Planung und Entwicklung eines VTK-Pipeline-Editors

Jonas Gilg

Bachelor-Arbeit

Hochschule Hannover
Fakultät IV – Wirtschaft und Informatik
Studiengang B. Sc. Angewandte Informatik

14. 10. 2016

Autor

Jonas Gilg
Matrikelnummer: 1271339
E-Mail: jonas@gilg-hannover.de

Erstprüfer

Prof. Dr. Volker Ahlers
Hochschule Hannover
Fakultät IV, Abteilung Informatik
Ricklinger Stadtweg 120
30459 Hannover

Zweitprüfer

Dipl.-Ing. Wito Engelke
Deutsches Zentrum für Luft- und Raumfahrt
Simulations- und Softwaretechnik
Software für Raumfahrtsysteme und interaktive Visualisierung
Lilienthalplatz 7
38108 Braunschweig

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Bachelor-Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum Unterschrift

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Hintergrund und Motivation . . . 1
- 1.2 Rahmen dieser Arbeit . . . 1
  - 1.2.1 Aufgaben . . . 1
  - 1.2.2 Ziele . . . 1
  - 1.2.3 Rahmenbedingungen . . . 2

2 Einführung und Grundlagen **3**
- 2.1 Wissenschaftliche Visualisierung . . . 3
  - 2.1.1 Visualisierungspipeline . . . 3
  - 2.1.2 Mögliches Einsatzgebiet - Wettervorhersagen . . . 4
- 2.2 Strömungssimulationen . . . 5
- 2.3 Strömungsvisualisierungen . . . 5
  - 2.3.1 Direkte Verfahren . . . 5
  - 2.3.2 Bildbasierte Verfahren . . . 7
  - 2.3.3 Geometriebasierte Verfahren . . . 8
  - 2.3.4 Topologische Verfahren . . . 8
- 2.4 Virtual Reality . . . 9
  - 2.4.1 Definition . . . 9
  - 2.4.2 Technologien . . . 9
  - 2.4.3 Powerwall . . . 10
  - 2.4.4 Verwendungsmöglichkeiten . . . 11
- 2.5 Knotenbasierte Editoren . . . 11
  - 2.5.1 Anwendungsgebiet - Unreal Engine 4 . . . 12

3 Anforderungsanalyse **15**
- 3.1 Untersuchung der bisher verwendeten Technologie UVDV . . . 15
  - 3.1.1 Allgemein . . . 15
  - 3.1.2 Arbeitsablauf . . . 16
  - 3.1.3 Funktionsweise . . . 17
  - 3.1.4 Bewertung . . . 17
- 3.2 Anforderungen and die Software . . . 18
  - 3.2.1 Use Cases . . . 18
- 3.3 Anforderungen and die Hardware und Zielplattformen . . . 20
  - 3.3.1 Anforderungen an die Hardware . . . 20
  - 3.3.2 Zielplattformen . . . 20
- 3.4 Anforderungen an die Softwarequalität . . . 21
  - 3.4.1 Unit Tests . . . 21
  - 3.4.2 Dokumentation . . . 21

3.5 Anforderungen Zusammengefasst . . . 21
3.6 Vorgehen . . . 23

4 Entwurf **25**
4.1 Überblick . . . 25
4.2 Visualisierungsanwendung . . . 25
4.2.1 Eingesetzte Technologien . . . 25
4.2.2 Alternative Frameworks . . . 27
4.2.3 Alternative Tools zur Strömungsvisualisierung . . . 30
4.2.4 Architektur . . . 31
4.3 Kommunikation . . . 34
4.3.1 Vergleich von Netzwerkbibliotheken . . . 35
4.3.2 Vergleich von Datenformaten zur Nachrichtenübertragung . . . 39
4.3.3 Architektur . . . 41
4.4 Editoranwendung . . . 45
4.4.1 Vergleich von GUI-Frameworks . . . 45
4.4.2 Architektur . . . 48
4.5 Unit Tests . . . 54
4.5.1 Google C++ Testing Framework . . . 54
4.5.2 JUnit . . . 54

5 Implementierung **55**
5.1 Allgemein . . . 55
5.2 Visualisierungsanwendung . . . 55
5.2.1 Werkzeuge zur Entwicklung . . . 55
5.2.2 Entwicklungsverlauf . . . 56
5.3 Netzwerk Kommunikation . . . 57
5.4 Editoranwendung . . . 57
5.4.1 Werkzeuge zur Entwicklung . . . 57
5.4.2 Entwicklungsverlauf . . . 58
5.5 Tests . . . 60

6 Ergebnisse **61**
6.1 Erfüllung der Anforderungen . . . 61
6.2 Anwendungsbeispiel . . . 62

7 Fazit und Ausblick **65**
7.1 Fazit . . . 65
7.2 Ausblick . . . 65

A Anhang **67**
A.1 XML-Reply-Nachricht . . . 67

Literaturverzeichnis **68**

1 Einleitung

1.1 Hintergrund und Motivation

Strömungssimulationen und deren Visualisierungen sind wichtige Werkzeuge bei der Entwicklung von Produkten. Sie ermöglichen ein schnelles Testen von Ideen mit minimalem Aufwand, da keine Material- und Produktionskosten für Prototypen anfallen. Gekoppelt mit Virtual Reality Technologie können Ingenieure ihre Produkte analysieren und Fehlerquellen entdecken oder Optimierungen vornehmen. Dabei wird es im Vergleich zu physischen Prototypen erleichtert Strömungsfelder darzustellen und Bereiche des Produktes sichtbar zu machen, in welche man sonst keinen Einblick hätte.

Das Deutsche Zentrum für Luft und Raumfahrt (DLR) nutzt ein solches Werkzeug für Demonstrationen zur Visualisierung von Strömungsdaten mit dem Namen Unsteady Vtk Data Visualizer (kurz UVDV), welches allerdings schwierig zu bedienen ist, da umfangreiche Kenntnisse über das Werkzeug notwendig sind, aber keine Dokumentation vorhanden ist. Somit soll für das DLR ein Werkzeug entwickelt werden, mit dem es einfach und intuitiv ist Strömungsvisualisierungen zu erstellen.

1.2 Rahmen dieser Arbeit

1.2.1 Aufgaben

Im Rahmen dieser Bachelorarbeit soll in Zusammenarbeit mit dem Deutschen Zentrum für Luft und Raumfahrt (DLR) eine erweiterbare Plattform geschaffen werden, welche auf vorhandener VR-Technologie aufbaut, um einfach und ohne Vorkenntnisse Strömungsvisualisierungen zu erstellen. Es soll die Möglichkeit bestehen die Strömungsvisualisierung getrennt von dem Editor laufen zu lassen, um gegebenenfalls Echtzeit-Änderungen von einem weiteren Rechner oder Tablet zu steuern. Folgende Teilaufgaben können aus der Aufgabenstellung abgeleitet werden:

- Untersuchung des Werkzeuges UVDV und ableiten von Use Cases aus dessen Funktionsweise
- Konzeption einer Rendersoftware, welche Strömungsvisualisierungen darstellen kann
- Konzeption eines Editors, welcher Strömungsvisualisierungen erstellen und in Echtzeit bearbeiten kann
- Konzeption einer Netzwerkarchitektur für die Kommunikation zwischen den zwei Anwendungen

1.2.2 Ziele

Folgende Ziele sind aus der Aufgabenstellung zu entnehmen:

- Implementierung einer Teilanwendung, welche Strömungsvisualisierungen in der, vom DLR verwendeten VR-Umgebung darstellen kann

- Implementierung einer Teilanwendung mit graphischer Oberfläche zur Erstellung und Modifikation von Strömungsvisualisierungen
- Implementierung einer Netzwerkarchitektur zur Echtzeitkommunikation zwischen den Teilanwendungen

1.2.3 Rahmenbedingungen

Das DLR hat zum erreichen dieser Ziele Rahmenbedingungen gestellt, welche im Folgenden erläutert werden.

Strömungsdaten werden aus dem VTK [1] Format importiert und somit muss die Visualisierungsanwendung VTK-Dateien importieren und darstellen können. VTK bietet eine umfangreiche Bibliothek zum Arbeiten mit VTK-Dateien an.

Die Strömungsvisualisierung muss auf einer Powerwall (siehe Kapitel 2.4.3) laufen, welche von einem Rechner-Cluster gesteuert wird. Hierfür verwendet das DLR bereits ein Framework mit dem Namen ViSTA [2], welches genau dieses Szenario unterstützt. Des Weiteren ist eine auf ViSTA aufbauende Bibliothek mit dem Namen FlowLib [2] verfügbar, die Strömungsvisualisierungen unter Verwendung des VTK-Dateiformates ermöglicht.

Aus den oben genannten Rahmenbedingungen lässt sich schließen, dass folgende Technologien verwendet werden müssen:

- VTK
- ViSTA Virtual Reality Toolkit
- ViSTA FlowLib

Näheres zu diesen Technologien in Kapitel 4.2.1.

2 Einführung und Grundlagen

In den folgenden Kapiteln werden Grundlagen betrachtet, die nötig sind, um den Rest der Arbeit und deren Motivation nachvollziehen zu können.

2.1 Wissenschaftliche Visualisierung

Wissenschaftliche Visualisierung ist in den vergangenen Jahrzehnten immer stärker in den Fokus gerückt, da sie die Veranschaulichung von komplexen und undurchschaubaren Prozessen ermöglicht. Gerade bei, für den Menschen nicht sichtbaren Abläufen, wie Luft- oder Wasserströmungen, können moderne Simulationstechnologien einen Einblick gewähren, aber auch Prozesse in komplexen Maschinen, wo ein Einblick im Betrieb nicht möglich ist, können durch Visualisierungstechnologien dargestellt und analysiert werden.

Ein weiterer Vorteil, den moderne, hochperformante Visualisierungssysteme mit sich bringen, ist der geringere Testaufwand, der betrieben werden muss, um eine schnelle und zuverlässige Entwicklung von Produkten zu gewährleisten. Es müssen keine teuren Prototypen gebaut werden und kleine Änderungen können schnell umgesetzt und erneut getestet werden. Dies fördert agile und iterative Entwicklungsprozesse.

2.1.1 Visualisierungspipeline

Um aus Rohdaten eine visuelle Darstellung zu generieren, durchlaufen die Daten eine sogenannte Visualisierungspipeline.

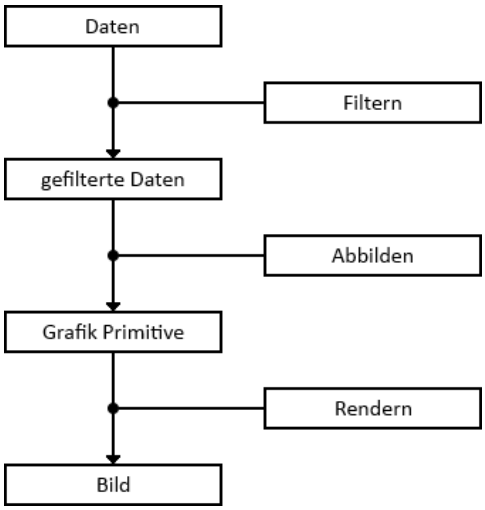

Abbildung 2.1: Die Visualisierungspipeline

Diese ist in Abbildung 2.1 zu sehen. Auf der linken Seite ist der Zustand der Daten abzulesen, auf der rechten Seite sind die auf die Daten angewandten Verfahren zu sehen. Das folgende Beispiel, welches Daten aus einem Windkanal visualisieren soll, durchläuft die Pipeline einmal:

1. Daten filtern:
   Windgeschwindigkeit und Windrichtung werden aus dem Datensatz herausgefiltert.

2. Abbilden:
   Die Daten werden auf Polygone abgebildet.

3. Rendering:
   Ein digitales Bild des Polygon-Modells wird erstellt.

2.1.2 Mögliches Einsatzgebiet - Wettervorhersagen

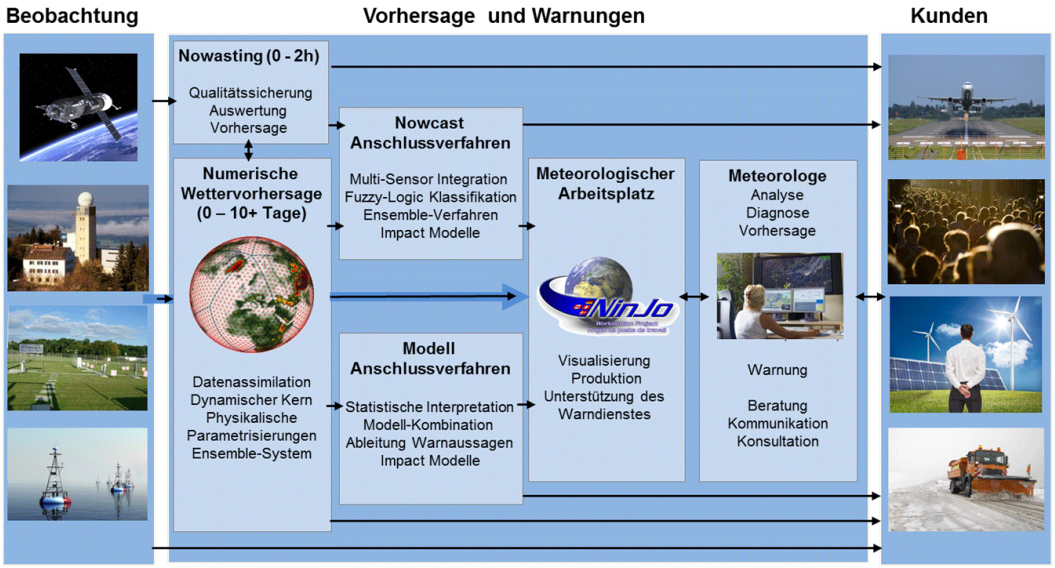

Abbildung 2.2: Der Wettervorhersageprozess [3].

Ein bekanntes Beispiel für eine wissenschaftliche Visualisierung ist der Wetterbericht. Hier werden unüberschaubar große Datenmengen aus der ganzen Welt erfasst. Daten wie Lufttemperatur, Luftdruck, Windrichtung, Satellitenbilder, Geographie und noch viele weitere, werden erfasst und analysiert, um zuverlässige Wettermodelle zu erstellen.

Allein aus den Zahlen kann selbst ein geübter Meteorologe nur wenig schließen. Deswegen werden unterschiedliche visuelle Modelle genutzt, um die Daten auszuwerten.

Wie in Abbildung 2.2 zu sehen ist, nutzt der Deutsche Wetterdienst auch eine Variante der Visualisierungspipeline. Die Pipeline beginnt auf der linken Seite mit dem Gewinnen der Daten und diese durchlaufen dann verschiedene wissenschaftliche Verfahren, um ein visuelles Modell zu generieren, welches am Ende auf der rechten Seite von Meteorologen und Kunden betrachtet und analysiert wird.

2.2 Strömungssimulationen

Strömungssimulationen oder auch Computational Fluid Dynamics (CFD) sind die numerische Simulation von Strömungen. Strömungssimulationen sind ein Instrument für die Versuchsplanung, -durchführung und -auswertung, ebenso wie ein Planungsinstrument für alle Vorhaben, die sich mit strömenden oder ruhenden Fluiden (Luft, Wasser, Schlamm uvm.) beschäftigen.

Die Erhaltungssätze der Physik für Masse, Impuls und Energie bilden eine wichtige Grundlage für Strömungssimulationen. Das gleiche gilt für empirische Grundsätze wie Turbulenz, Wärmestrahlung, poröse Medien etc. All dies wird mithilfe von numerischen Algorithmen in einen 2D- oder 3D-Raum projiziert, der optional auch eine zusätzliche, zeitliche Abhängigkeit haben kann. [4]

2.3 Strömungsvisualisierungen

Zum Auswerten von Strömungssimulationen werden, neben den numerischen Analyseverfahren, vor allem Visualisierungsverfahren genutzt, da diese die Simulationsergebnisse anschaulicher darstellen. Es ist deutlich einfacher visuelle Modelle auszuwerten, da Zusammenhänge vom Gehirn schneller erfasst werden können, als bei numerischen Verfahren.

Die räumliche Darstellung kann aufgrund von technischen Einschränkungen, wie begrenztem Arbeitsspeicher und begrenzter Rechenleistung, keine unendliche Auflösung haben. Aus diesem Grund werden die Daten in Rasterzellen angelegt und haben somit nur eine eingeschränkte Genauigkeit.

Um die einzelnen Rasterzellen zu visualisieren gibt es verschiedene Verfahren, welche in folgende Bereiche zusammenzufassen sind:

- Direkte Verfahren
- Bildbasierte Verfahren
- Geometriebasierte Verfahren
- Topologische Verfahren

[5, 6]

Diese Verfahren werden in den folgenden Kapiteln näher betrachtet.

2.3.1 Direkte Verfahren

Die direkten Verfahren – auch punktbasierte Verfahren genannt – visualisieren jede Rasterzelle direkt. Es werden im Vergleich zu anderen Verfahren keine Zusammenhänge zwischen Rasterzellen berücksichtigt. Diese Verfahren bieten sich vor allem für 2D-Darstellungen an, da jede Rasterzelle dargestellt wird, was in 3D-Räumen unübersichtlich sein kann. [7, 6] Es gibt aber auch Methoden um direkte Verfahren in 3D-Räumen zu visualisieren, wie zum Beispiel ein Schnittverfahren, welches eine 2D-Ebene im 3D-Raum darstellt, wobei nur die 2D-Ebene mit einem direkten Verfahren visualisiert wird. Ein Beispiel hierfür ist in Abbildung 2.3 zu sehen.

Die Farbkodierung und die Pfeildarstellung sind die am häufigsten verwendeten direkten Verfahren, da sie sich direkt aus den unterliegenden Daten erstellen lassen. Beide Verfahren werden in den folgenden Kapiteln näher betrachtet.

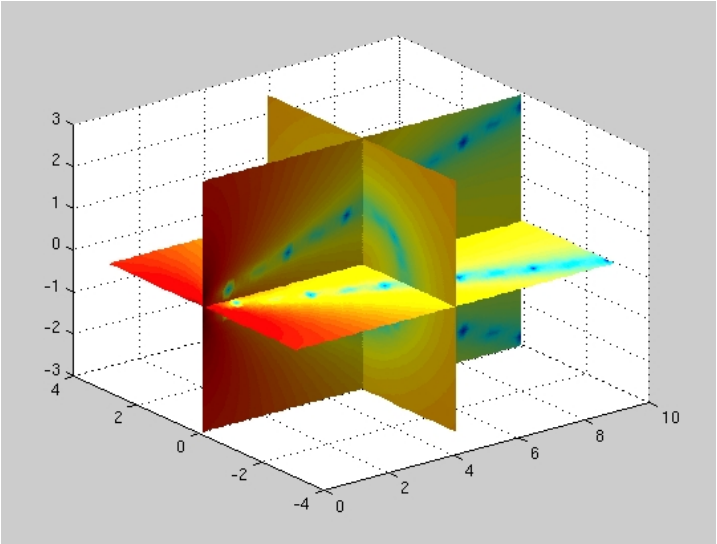

Abbildung 2.3: Ebenen werden verwendet, um punktbasierte Verfahren im 3D-Raum besser nutzen zu können. [8]

Farbkodierung

Bei der Farbkodierung werden aus den vorhandenen Strömungsdaten Skalarfelder extrahiert. Daten (wie Geschwindigkeit, Druck, Turbulenzen, Temperatur etc.) lassen sich als einfaches Skalar darstellen. Diese Skalare werden dann auf Farben projiziert, sodass jeder Rasterzelle ein Farbwert zugeteilt wird. Mithilfe von Farbkodierung können schnell Schlüsse über das Verhalten einer Komponente gezogen werden, da Maxima und Minima einfach zu erkennen sind. Ein Bild kann hier also tatsächlich mehr als tausend Worte sagen. [9]

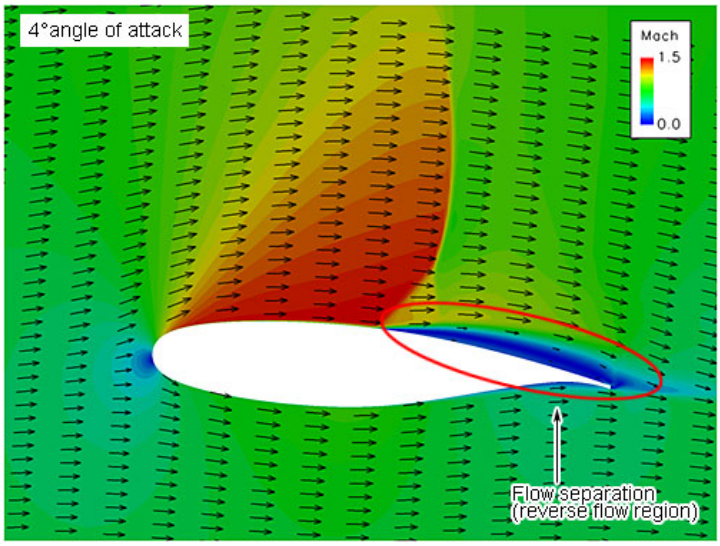

Abbildung 2.4: CFD Visualisierung, die Farbkodierung nutzt [10].

In Abbildung 2.4 kann man eine CFD Visualisierung der japanischen Raumfahrtagentur JAXA sehen. Diese zeigt den Flügel eines Passagierflugzeuges. Der Luftstrom ist durch Pfeile gekennzeichnet und der Winkel des Flügels zum Luftstrom beträgt 4°. Die Luftgeschwindigkeit relativ zum Flügel ist durch Farben kodiert, wobei die Kodierung in der oberen, rechten Ecke abzulesen ist. Anhand dieser Farbkodierung kann man schnell erkennen, dass im vorderen Bereich des Flügels hohe Geschwindigkeiten im Überschallbereich und im hinteren Bereich des Flügels niedrige Geschwindigkeiten herrschen. Physikalisch hätte dies einen Strömungsabriss zur Folge, sodass der Pilot mit einem potentiellen Kontrollverlust konfrontiert wäre. Die Farbkodierung kann dieses Problem einfach visualisieren.

Pfeildarstellung

Die Pfeildarstellung oder auch Vektordarstellung nutzt Pfeile mit Richtung und Größe/Länge als Variablen. Jeder Rasterzelle wird mithilfe der vorliegenden Daten ein solcher Pfeil zugeordnet. In der Regel wird diese Form der Visualisierung genutzt um Strömungsrichtung und -geschwindigkeit darzustellen, indem die Größe bzw. Länge des Pfeils die Geschwindigkeit indiziert.

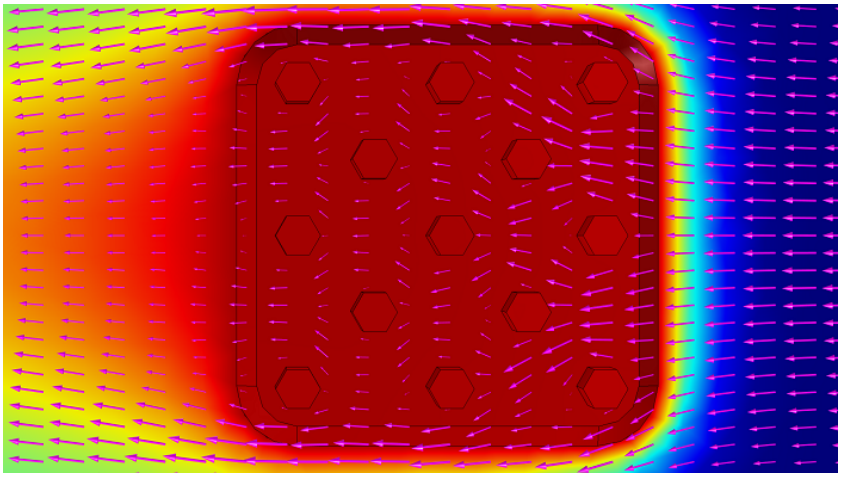

Abbildung 2.5: CFD Visualisierung, welche eine Pfeildarstellung nutzt [11].

In Abbildung 2.5 ist ein Beispiel für eine Pfeildarstellung zur Visualisierung der Luftströmungen in einem Kühlkörper dargestellt. Anhand der Pfeile ist zu erkennen, in welche Richtung die Luft strömt. Außerdem zeigt die Länge der Pfeiler die Geschwindigkeit an.

2.3.2 Bildbasierte Verfahren

Bildbasierte Verfahren generieren Texturen aus den Strömungsdaten, welche eine Repräsentation der Strömung widerspiegeln. Die Texturgenerierung erfolgt durch filtern eines Bildes unter Nutzung der lokalen Strömungsvektoren [6]. In Abbildung 2.6 ist ein das Ergebnis eines solchen Verfahrens zu sehen, welches die Luftströmung an einem Auto visualisiert.

Die am häufigsten verwendeten Verfahren sind:

- Punktrauschen
- Line Integral Convolution

Abbildung 2.6: Texturbasierte Visualisierung der Luftströmungen an einem Auto [12].

2.3.3 Geometriebasierte Verfahren

Geometriebasierte Verfahren werden eingesetzt, um das Langzeitverhalten von Strömungen zu untersuchen. Hierzu werden integrationsbasierte Ansätze genutzt. Zuerst werden die Strömungsdaten integriert und dann unter Zuhilfenahme von geometrischen Objekten visualisiert. Die entstandenen Geometrien spiegeln dann die Eigenschaften der Strömung wieder. Isoflächen hingegen nutzen einen anderen Ansatz, in dem nur selektiv Rasterzellen mit einer vorher bestimmten Eigenschaft dargestellt werden. So können zum Beispiel Übergangsbereiche im Vektorfeld besser darstellen, wenn nur Vektoren mit bestimmter Länge visualisert werden. [6, 13]

Folgende geometriebasierte Verfahren werden häufig angewendet:

- Isoflächen
- Stromlinien
- Stromtuben

Stromlinien

Stromlinien werden, ähnlich wie die Pfeildarstellung, für die Richtung von Strömungen genutzt. Die Stromlinie ist an jeder Stelle parallel zum Geschwindigkeitsvektor des Datensatzes [14]. Man kann eine Stromlinie mit der Bewegung eines Partikels im Vektorfeld vergleichen. [13]

In Abbildung 2.7 kann man Stromlinien sehen, welche verwendet werden, um Strömungen um ein Spaceshuttle zu visualisieren.

2.3.4 Topologische Verfahren

Topologische Verfahren basieren darauf, dass vor der Visualisierung für Wissenschaftler wichtige Teile aus den Strömungsdaten extrahiert werden. So werden nur ausgewählte Daten visualisiert [16]. Ein Beispiel für topologische Verfahren sind topologische Skelette [5].

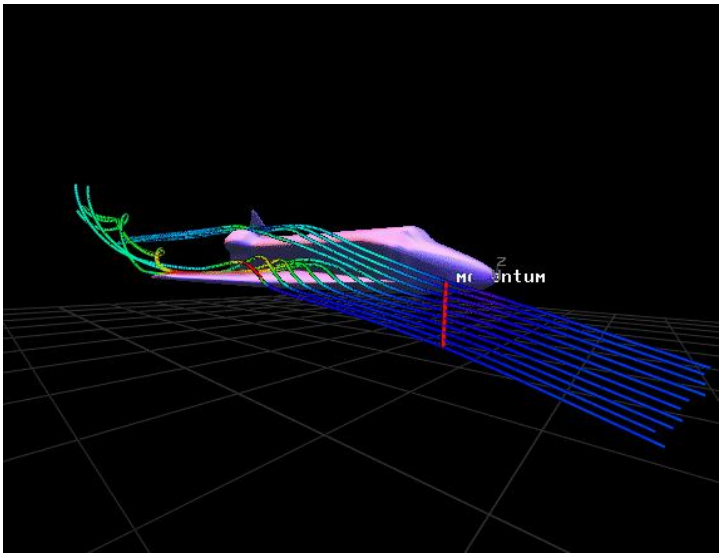

Abbildung 2.7: Stromlinien die Strömungen um ein Spaceshuttle darstellen. [15]

2.4 Virtual Reality

Im Folgenden gibt es eine kurze Einführung zur Virtual Reality Technologie. Insbesondere wird (in Kapitel 2.4.3) auf Powerwalls eingegangen, da die Zielplattform der zu entwickelnden Applikation eine solche ist.

2.4.1 Definition

Virtual Reality ist eine computergenerierte Simulation einer dreidimensionalen Umgebung mit der man nahezu nahtlos physisch interagieren kann, indem spezielle elektronische Geräte genutzt werden. Zum Beispiel können Handschuhe mit Sensoren Interaktionen simulieren [17]. Dies führt zu einer sehr hohen Immersion, da das Gefühl entsteht mit seinen Händen in der Simulation zu sein.

Virtual Reality ist nicht mit Augmented Reality zu verwechseln, welche computergenerierte Bilder in die reale Welt projiziert.

2.4.2 Technologien

Es gibt verschiedene Technologien sowohl zur Darstellung als auch zur Interaktion, die in den zwei folgenden Kapiteln kurz angeschnitten werden.

Technologien zur Darstellung

Virtual Reality Technologien machen Gebrauch von separaten Bildern für jedes Auge. Es wird versucht den Augen, wie in der Realität, zwei verschiedene Blickwinkel zu geben, sodass das Gehirn ein dreidimensionales Bild generieren kann. Damit eine flüssige Wahrnehmung erreicht werden kann, muss jedes Auge mindestens 30 Bilder pro Sekunde sehen. Mehr Bilder pro Sekunde sorgen allerdings für höhe-

re Immersion und geringere Kinetose[18] (Seekrankheit). Um dies zu erreichen gibt es verschiedene Ansätze:

- VR Brillen wie das Oculus Rift[19] oder HTC Vive[20] nutzen ein Headset, welches jeweils ein Bild für jedes Auge projiziert. Es werden dafür speziell angefertigte Bildschirme verwendet. Sie nutzen einen externen Computer zu Berechnung der Bilder.
- VR Brillen wie das Gear VR[21] oder Google Cardboard[22] nutzen im Vergleich zu den oben genannten Brillen ein Smartphone als Bildschirm und als Recheneinheit.
- Bildschirme und Shutterbrillen wie das NVIDIA 3D Vision System[23] nutzen den selben Bildschirm für jedes Auge, wobei abwechselnd Bilder für das linke und rechte Auge dargestellt werden, gleichzeitig verdunkelt die Brille das jeweils andere Auge.
- Polarisationsfähige Bildschirme und Polarisationsfilterbrillen nutzen auch einen Bildschirm für beide Augen, allerdings werden für jedes Auge Bilder mit entgegengesetztem, polarisiertem Licht ausgestrahlt. Die Brille enthält passende Polarisationsfilter für jedes Auge, sodass zu jedem Auge nur eines der beiden Bilder durchdringt[24]. Diese Technologie wird vor allem in Kinos eingesetzt.

Technologien zur Interaktion

Die Interaktion mit der Virtual Reality Simulation kann je nach Darstellungstechnologie und Anwendungsgebiet ganz unterschiedlich stattfinden.

- Für Flugsimulatoren werden Joysticks oder sogar Cockpitnachbauten verwendet.
- Oculus[19] und Vive[20] bieten spezielle Controller an mit denen die Hände im VR-Raum simuliert werden.
- Spezielle Handschuhe wie Manus VR[25] simulieren mithilfe von Sensoren im Handschuhe jede Hand- und Fingerbewegung.
- Controller wie der Flysticks2[26], die keine Spezialisierung für bestimmte Anwendungsgebiete mit sich bringen, sondern eher eine Vielzahl von Funktionalitäten anbieten.
- Motiontracking und Gesture-Control wie Orion[27], welche Bildanalyseverfahren nutzen, um die Bewegungen von der realen Welt in die simulierte Welt zu transferieren.

Es gibt noch viele weitere Interaktionstechnologien, die allerdings seltener Anwendung finden und hier keine Erwähnung finden sollen.

2.4.3 Powerwall

Da die Zielplattform dieser Arbeit eine Powerwall ist, wird diese Technologie in diesem Kapitel näher betrachtet.

Powerwalls nutzen mehrere Projektorenpaare, um große Flächen zur VR-Umgebung zu machen. Dazu werden die Projektorpaare an einen Rechnercluster gebunden, wo je Projektorpaar ein Rechner zur

Verfügung steht. Jeder Rechner ist für eine stereoskopische Ansicht eines Bildausschnittes zuständig. Dadurch wird erreicht, dass die Rechner synchron eine große Fläche nahtlos stereoskopisch mit der gerenderten Szene abdecken.

Um nun einen 3D Effekt zu erhalten werden Shutterbrillen eingesetzt. Diese werden mithilfe von Motiontracking verfolgt, sodass die Szene sich an die Bewegungen des Benutzers anpassen kann.

Des Weiteren werden Flysticks verwendet, um mit der Applikation zu interagieren.

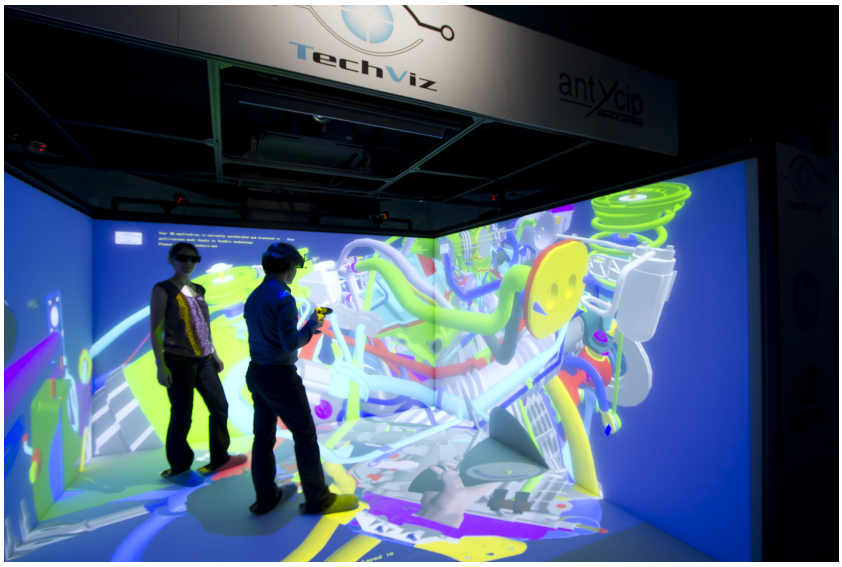

Abbildung 2.8: Ein CAVE System [28]

In Abbildung 2.8 ist ein CAVE System abgebildet. Dies ist eine erweiterte Powerwall mit Projektorpaaren für den Boden und die seitlichen Wände, sodass ein tieferes VR-Erlebnis entsteht.

2.4.4 Verwendungsmöglichkeiten

Die Anwendungsgebiete der Virtuellen Realität sind endlos. Es geht von simplen Spielen über wissenschaftliche Visualisierung und Analyse, bis hin zum Training von Piloten und Astronauten. Vor allem in Gebieten zur wissenschaftlichen Visualisierung finden VR-Technologien Verwendung, da gerade 3D Strömungen in 2D Umgebungen schwer nachvollziehbar sind.

2.5 Knotenbasierte Editoren

Diese Arbeit hat als Ziel, einen einfach zu bedienenden Editor zu bauen, der komplexe Aufgaben löst. Dazu wird hier kurz das Prinzip hinter knotenbasierten Editoren erklärt.

Knotenbasierte Editoren nutzen ein sehr einfaches Prinzip zum Erstellen komplexer Systeme: Beziehungen. Es gibt Komponenten oder auch Knoten genannt, welche in Beziehung zueinander stehen. Diese Beziehungen heißen auch Kanten. Das ganze wird dann visualisiert, indem Knoten als Boxen dargestellt werden und Kanten als Linien zwischen diesen Boxen.

Dann kann der Benutzer Linien zwischen Boxen ziehen und somit Beziehungen definieren. Ein einfaches Beispiel hierfür ist ein Förderband in einer Fabrik.

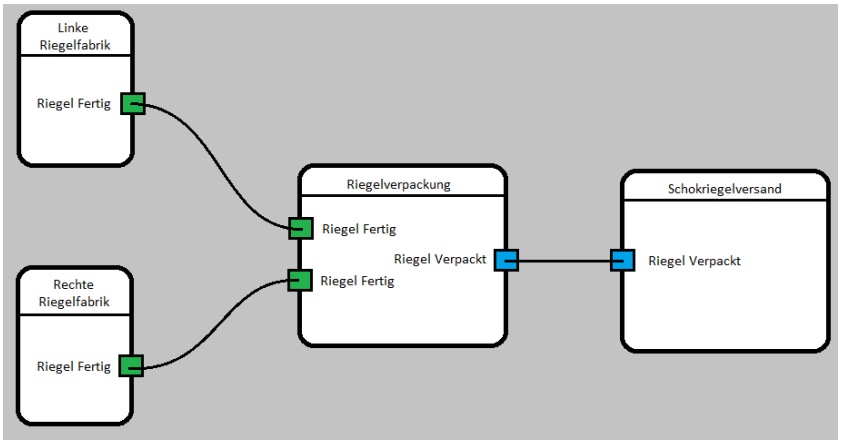

Abbildung 2.9: Ein Beispiel für einen knotenbasierten Editor

In Abbildung 2.9 ist ein solches Anwendungsbeispiel dargestellt. Es geht um eine Produktionskette. Die Fabriken produzieren Schokoriegel. Jeweils zwei Riegel werden in eine Verpackung eingepackt. Verpackte Riegel werden ausgeliefert.

Jeder Knoten hat einen Namen, dieser wird oben angezeigt, außerdem haben Knoten ein und ausgehende Anschlusspunkte. Diese werden mit der Bezeichnung der Beziehung beschrieben. Zuletzt gibt es noch die Kanten. Sie verbinden zwei Anschlusspunkte mit der gleichen Beziehung.

So können größere und komplexere Systeme einfach erstellt werden und die Funktionsweise ist visuell gut nachvollziehbar.

2.5.1 Anwendungsgebiet - Unreal Engine 4

In Abbildung 2.10 ist der Editor der Unreal Engine 4 zu sehen. Dieser wird unter anderem verwendet um die Spiellogik zu implementieren. Dieser grafische Ansatz hat den Vorteil, dass weniger programmiert werden muss und somit Personen an der Spielentwicklung mitwirken können, die keine Programmierkenntnisse haben. So können zum Beispiel Grafiker ihre Texturen einfach selber in das Spiel integrieren, ohne vorher einen Softwareentwickler darum zu bitten. Auch sind Programmabläufe und Logik einfacher nachzuvollziehen, was Fehler hervorhebt oder ganz vermeidet. Ebenfalls wird die Modularität erhöht, da man versucht Knoten wiederverwendbar zu machen.

Es kann aber passieren, dass zu viele Knoten mit zu vielen Verbindungen zu sehr unübersichtlichen Szenarien führen. Deshalb ist es wichtig auf die richtige Nutzung zu achten.

Zusammenfassend können Spiele so schneller und zuverlässiger entwickelt werden, solange die Funktionen des Editors mit Bedacht eingesetzt werden.

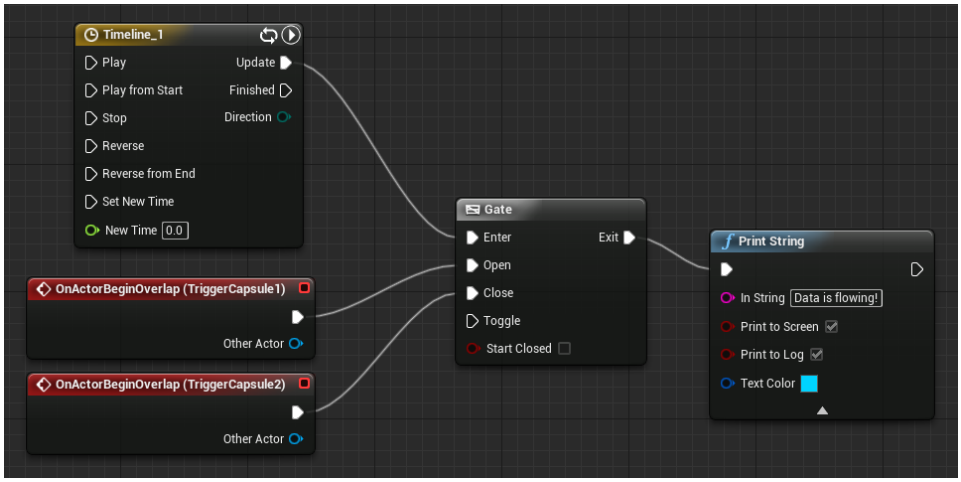

Abbildung 2.10: Der Editor der UnrealEngine [29]

3 Anforderungsanalyse

Im Folgenden wird eine Anforderungsanalyse erstellt, in welcher ermittelt werden soll welche Anforderungen an die zu bauende Software gestellt werden. Es werden Use Cases, Ziele und Anforderungen erhoben. Anforderungen werden in folgende Typen eingeordnet:

- funktionale Anforderungen – welche fachlichen Dienste soll das System bereitstellen
- nichtfunktionale Anforderungen – Qualitätseigenschaften, Ergonomie, Verfügbarkeit
- technische Anforderungen – technische Infrastruktur, Schnittstellen zu anderen Systemen

[30]

Nach einem kurzen einführenden Gespräch mit dem betreuenden Mitarbeiter beim DLR ist die grobe Zielsetzung das bisherig verwendete Programm Unsteady VTK Data Visualizer (UVDV) zu ersetzen. Mit dieser Information sind folgende Arbeitsschritte für die Anforderungsanalyse entstanden:

- Untersuchen von UVDV
- Finden von Use Cases über Gespräche mit Mitarbeitern
- Finden von Hardware und Betriebssystemanforderungen über Gespräche mit Mitarbeitern

3.1 Untersuchung der bisher verwendeten Technologie UVDV

Als Teil der Anforderungsanalyse wird die Software UVDV untersucht. Zunächst gibt es eine kleine Einleitung für welche Anwendungsgebiete diese Software existiert. In einem zweiten Schritt wird der Arbeitsablauf untersucht in dem UVDV verwendet wird. Danach wird der Quellcode vom UVDV betrachtet, um die genaue Funktionsweise dieser Software zu erfahren und gegebenenfalls darauf aufzubauen. Zuletzt soll die Auswertung Stärken und Schwächen von UVDV offenbaren.

3.1.1 Allgemein

Unsteady VTK Data Visualizer (UVDV) ist entwickelt worden, um VTK basierte Strömungssimulationen visuell darzustellen. Näheres über VTK im Kapitel 4.2.1. Im DLR werden diese Visualisierungen auf einer Powerwall (siehe Kapitel 2.4.3) dargestellt und mit einem Flystick (siehe Kapitel 2.4.2) bedient. UVDV bietet die Möglichkeit beide dieser Technologien mithilfe der ViSTA und ViSTA FlowLib Frameworks (siehe Kapitel 4.2.1) zu unterstützen. Somit können komplexe Strömungssimulationen in 3D visualisiert und Interaktionen möglich gemacht werden.

Es besteht die Möglichkeit mithilfe des Flysticks Partikel in die Visualisierung einzusetzen und den Verlauf dieser Partikel zu folgen. Somit können Strömungen besser nachvollzogen und Konstruktionsfehler schnell erkannt werden.

3.1.2 Arbeitsablauf

Um die Funktionsweise besser zu verstehen, wurde der Arbeitsablauf einer Nutzung nachgestellt und untersucht. Folgendes Szenario wurde hierfür verwendet:

Ein Wissenschaftler kommt mit einem VTK Datensatz, der einen Zyklon zur Trennung von Partikeln darstellt. Es soll herausgefunden werden ob dieses Produkt funktionsfähig ist.

Als Erstes wird der Datensatz auf seine Komponenten untersucht. Es gibt VTK Dateien die nur zu Visualisierung vorhanden sind, wie eine aufgeschnittene Hälfte des Zyklons, VTK Dateien die Steamlines darstellen oder VTK Dateien die Vektorfelder beschreiben.

```
[GLOBAL]
DEBUG       = true
SCENES      = SCENE
MASTER_TIMING   = TIMING_ZYKLON
LOOP_TIME   = 60
BACKGROUND  = 0.1, 0.1, 0.1
STATES      = Standard, Timenavigation, Particletracing, VolumeSlicer, VolumeRendering
SHOW_TIMING = TRUE
[SCENE]
MASTER_TIMING   = TIMING_ZYKLON
LOOP_TIME   = 60
COMPONENTS  = Geometry, VelocityParticleTraces, VelocityParticles, VelocityVolume
AUTOADJUST  = FALSE
SCALE       = 1.0
CENTER      = 0.0, 0.0, 0.0
ROTATION    = 0.0, 0.0, 0.0, 1.0
BACKGROUND  = 0.1, 0.1, 0.1

[Geometry]
FILE_NAME   = /Zyklon/vtk_geom/half_geom.vtk
TIMING_SECTION  = TIMING_ZYKLON
DATA_FORMAT     = VTK_POLYDATA
OPACITY     = 1.0
COLOR       = 1.0, 1.0, 0.0

#----------------------------
#=== INTEGRATION SETTINGS  ===
#----------------------------
[RungeKutta4]
MAX_NUM_STEPS       = 1000
TERMINATION_SPEED   = 0.00000000001
INTEGRATION_DIRECTION   = BOTH
MAX_PROPAGATION_UNIT    = LENGTH_UNIT
MAX_PROPAGATION_TIME    = 185
INIT_INT_STEP_UNIT  = CELL_LENGTH_UNIT
INTEGRATION_LENGTH  = 0.2
INTEGRATOR_TYPE     = RK4
PARTICLE_TIME_TYPE  = 0
```

Abbildung 3.1: Initialisierungsdatei für eine UVDV Strömungsvisualisierung.

Dann muss eine Initialisierungsdatei geschrieben werden, die die Simulation beschreibt. Ein Ausschnitt ist in Abbildung 3.1 zu sehen. Eine solche Initialisierungsdatei erfordert einen DLR Mitarbeiter, der sich mit der Software gut auskennt, da es keine Dokumentation zum Schreiben einer solchen Datei gibt.

Jedes mal, wenn ein Teil der Initialisierungsdatei fertig ist, wird UVDV gestartet und es wird bewertet, ob die Initialisierungsdatei korrekt geschrieben wurde oder ob Werte verändert werden müssen. Dies ist ein sehr zeitaufwändiger Prozess, der sowohl die Expertise des Wissenschaftlers benötigt, als auch die des UVDV-kundigen Mitarbeiters.

Wenn beide mit der, aus der Initialisierungsdatei generierten, Visualisierung zufrieden sind, kann der Wissenschaftler diese untersuchen. Wenn beim Untersuchen wieder Werte angepasst werden müssen, muss zuerst die Visualisierung unterbrochen werden, danach ist es möglich die Werte in der Initialisierungsdatei anzupassen. Zuletzt kann die Visualisierung neu gestartet und erneut untersucht werden.

Am Ende ist der Wissenschaftler zu dem Ergebnis gekommen, dass die Komponente nicht ihren Zweck erfüllt.

3.1.3 Funktionsweise

Da UVDV keinerlei Dokumentation mit sich bringt, aber der Quellcode zur Verfügung steht wird dieser untersucht, um die Funktionsweise der Software nachzuvollziehen. Vor allem Informationen zur Nutzung von ViSTA FlowLib innerhalb von UVDV sind von Interesse, da dieses Framework ebenfalls schlecht bis gar nicht dokumentiert ist.

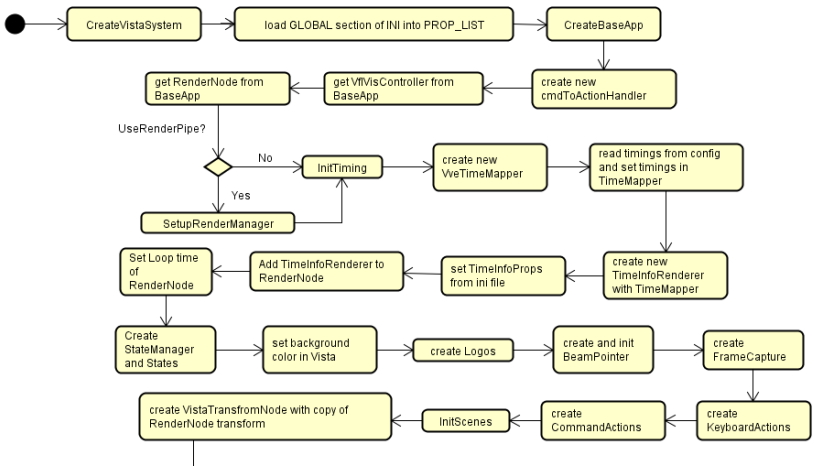

Abbildung 3.2: Aktivitätsdiagramm der UVDVs Programmablauf beschreibt.

Um die Funktionsweise für den späteren Entwicklungsprozess zu dokumentieren, wurde ein Aktivitätsdiagramm erstellt, das den Ablauf der Anwendung beschreibt. In Abbildung 3.2 ist ein Ausschnitt dieses Aktivitätsdiagramms zu sehen. Wie zu erkennen ist, hat das Programm einen sehr linearen Ablauf. In diesem wird hauptsächlich die Initialisierungsdatei gelesen, aus den daraus gewonnen Informationen werden ViSTA FlowLib Objekte erstellt und ihre Parameter angepasst. Anschließend werden sie an den Szenegraphen angehängt und zum Schluss wird die ViSTA FlowLib Visualisierung gestartet. Das Erstellen neuer Objekte und das Anpassen von Parametern ist deshalb nicht während der Laufzeit möglich.

3.1.4 Bewertung

UVDV ermöglicht es komplexe Strömungssimulationen zu visualisieren, jedoch ist die Bedienung unnötig komplex und zeitaufwändig.

Zum einen benötigt jede Visualisierung einen erfahrenen, UVDV-kundigen Mitarbeiter, welcher die Funktionsweise und Syntax der UVDV-Initialisierungsdateien kennt, d.h. ein wissenschaftlicher Mitarbeiter kann eine Visualisierung nicht allein starten, sondern muss unter Umständen sogar auf einen Termin warten.

Zum anderen müssen selbst bei kleinsten Änderungen die Visualisierungen neu gestartet werden, da nur über die Initialisierungsdateien Anpassungen vorgenommen werden können.

3.2 Anforderungen and die Software

Die Anforderungen für die Software ergeben sich aus Gesprächen mit DLR Mitarbeitern und UVDVs Funktionalität. Ziel dieser Arbeit soll nicht sein UVDV komplett zu ersetzen, sondern eine erweiterbare Software zu bauen, die UVDVs Probleme aufgreift und diese löst. Diese Erweiterbarkeit soll aber in Zukunft dazu genutzt werden können, eine Alternative zu schaffen. Wenn dies geschehen ist, können UVDVs Funktionalitäten reimplementiert werden.

Die vom DLR gestellten Anforderungen sind wie folgt:

Verwendung von ViSTA und ViSTA FlowLib zum Visualisieren von VTK basierten Strömungssimulationen. Die Visualisierungen sollen zur Laufzeit erstellt und verändert werden, indem ein grafischer Editor bereitgestellt wird, welcher über das Netzwerk mit der Visualisierung kommuniziert. Dies soll gewährleisten, dass die Visualisierung auf einem anderen Gerät läuft als der Editor, sodass zum Beispiel Modifikationen von einem Tablet vorgenommen werden können, während die Visualisierung auf einer Powerwall läuft. Der Editor soll knotenbasiert Komponenten miteinander Verknüpfen können.

Des Weiteren soll das gleichzeitige Verbinden von einem Editor zu mehreren Visualisierungen funktionieren. Dies wird benötigt, um Visualisierungen im Cluster durchzuführen und auf der, im DLR befindlichen, Powerwall darzustellen. Außerdem kann so parallel die selbe Visualisierung auf verschiedenen Geräten durchgeführt werden.

3.2.1 Use Cases

Da nicht alle von UVDVs Funktionalitäten implementiert werden können, wurden Mindestanforderungen gestellt. Diese sind in den folgenden Use Cases zusammen mit allen anderen Anforderungen dargestellt:

Use Case: Editorstart

Vorbedingung: Der Benutzer hat die Anwendung gestartet

Ereignisfluss:

1. Ein Fenster öffnet sich, welches den Benutzer nach einer IP Adresse fragt.
2. Der Benutzer gibt die IP Adresse einer Visualisierungsanwendung an.
3. Der Benutzer klickt auf OK und der Use Case endet.

Alternativen:

- Der Benutzer klickt auf Abbrechen und die Anwendung schließt.

Use Case: Komponente Erstellen

Vorbedingung: Der Editor ist mit einer Visualisierungsanwendung verbunden

Ereignisfluss:

1. Der Benutzer zieht sich aus dem Komponentenfeld eine Komponente in die Arbeitsfläche.
2. Die Komponente erscheint auf der Arbeitsfläche und ist in der Visualisierungsanwendung eingebunden.

Mögliche Komponenten:

1. VTK Data (spezieller Use Case)
2. Scalarbar
3. Render Node
4. Time Info

Use Case: VTK Datensatz Komponente erstellen

Vorbedingung: Der Editor ist mit einer Visualisierungsanwendung verbunden

Ereignisfluss:

1. Der Nutzer zieht eine oder mehrere VTK Dateien aus einem Dateiexplorer in die Arbeitsfläche
2. Die Komponente erscheint auf der Arbeitsfläche und ist in der Visualisierungsanwendung eingebunden.

Alternativ:

1. Der Benutzer zieht sich aus dem Komponentenfeld eine VTK Data Komponente in die Arbeitsfläche.
2. Ein Dateiexplorer erscheint, aus dem der Benutzer eine oder mehrere VTK Dateien auswählt.
3. Die Komponente erscheint auf der Arbeitsfläche und ist in der Visualisierungsanwendung eingebunden.

Use Case: Komponenteneigenschaften bearbeiten

Vorbedingung: Eine Komponente ist auf der Arbeitsfläche

Ereignisfluss:

1. Der Benutzer klickt auf eine Komponente in der Arbeitsfläche.
2. Eine Ansicht der Komponenteneigenschaften öffnet sich.
3. Der Benutzer ändert eine der Eigenschaften und die Änderung wirkt sich auf die Visualisierung aus. Der Use Case endet.

Use Case: Komponenten Verknüpfen
Vorbedingung: Der Editor ist mit einer Visualisierungsanwendung verbunden
Ereignisfluss:

1. Der Benutzer beginnt von einer Komponente die Maus zu einer anderen zu ziehen. Eine Linie erscheint von der Komponente zur Maus.
2. Der Benutzer zieht die Maus über eine andere Komponente. Ihm wird angezeigt, ob eine Verbindung zwischen diesen Komponenten möglich ist.
3. Der Benutzer lässt die Maus los und die Komponenten werden mit einer Linie verbunden und die Beziehung wird in der Visualisierung widergespiegelt. Der Use Case endet.

Alternativ:

- zu 2. Der Benutzer lässt die Maus los und der Use Case endet.

Use Case: Mit weiteren Visualisierungen Verbinden
Vorbedingung: Der Editor ist bereits mit einer Visualisierungsanwendung verbunden
Ereignisfluss:

1. Der Benutzer öffnet ein Verbinden-Menü, er wird aufgefordert eine IP Adresse einzugeben.
2. Der Benutzer gibt eine IP Adresse ein und drückt auf OK.
3. Der Editor baut eine Verbindung zu dieser IP Adresse auf und ist mit der Visualisierung verbunden. Der Use Case endet.

Alternativ:

- zu 2. Der Benutzer drückt Abbrechen und der Use Case endet.

3.3 Anforderungen and die Hardware und Zielplattformen

Zusätzlich zu den Anforderungen an die Software gibt es Anforderungen an Hardware und Betriebssysteme, welche in den folgenden Kapiteln erläutert werden.

3.3.1 Anforderungen an die Hardware

Die Anforderungen an die Hardware können nicht genau spezifiziert werden, da die Komplexität der Visualisierung von dem Datensatz abhängt. Jedoch ist festzuhalten, dass die Performanz von ViSTA und ViSTA FlowLib nicht beeinträchtigt werden sollte. Die Applikation sollte eine gleiche Performance wie UVDV aufweisen und auch auf den selben Endgeräten laufen. Vor allem die Powerwall sollte unterstützt werden.

3.3.2 Zielplattformen

Die Visualisierung muss Betriebssystemunabhängig laufen. Es müssen Windows 7, und SUSE Enterprise unterstützt werden. Der Editor sollte die gleichen Bedingungen erfüllen.

3.4 Anforderungen an die Softwarequalität

Es wurden keine Anforderungen an die Softwarequalität gestellt, dennoch werden Maßnahmen ergriffen, um diese zu sichern.

3.4.1 Unit Tests

Jede Klasse und jede Methode muss mit einem White Box Test geprüft werden. Dabei muss jeder mögliche Kontrollfluss berücksichtigt werden.

Wenn eine Klasse oder Methode nachträglich verändert wird, muss der dazugehörige Test überprüft und gegebenenfalls erneuert oder erweitert werden.

3.4.2 Dokumentation

Jede Klasse und jede Methode muss dokumentiert werden. In der Dokumentation muss eine kurze Erklärung enthalten sein, welche die Funktionsweise und eventuelle Anwendungsfälle erklärt. Des Weiteren müssen alle Methodenparameter und die Rückgabewerte dokumentiert sein. Geworfene Exceptions und ihre Auslöser sollen notiert werden.

Wenn eine Klasse oder Methode modifiziert wird, muss die dazugehörige Dokumentation überprüft und gegebenenfalls erneuert oder erweitert werden.

3.5 Anforderungen Zusammengefasst

Zur Übersicht noch einmal eine zusammengefasste Aufzählung aller Anforderungen, kategorisiert nach funktionalen, nichtfunktionalen und technischen Anforderungstypen. Diese Anforderungen werden am Ende dieser Arbeit auf ihre Erfüllung überprüft.

Funktionale Anforderungen

F1: Der Benutzer kann die Visualisierungsanwendung und die Editoranwendung seperat auf zwei sich im Netzwerk befindlichen Geräten starten und diese unter Verwendung der IP-Adresse der Visualisierungsanwendung aus der Editoranwendung heraus verbinden.

F2: Der Benutzter kann mindestens die unten aufgelisteten Komponenten per Drag&Drop in der Editoranwendung erstellen. Diese Aktion bewirkt, dass die Komponenten in der Visualisierungsanwendung erstellt werden und dort ihren jeweiligen in ViSTA FlowLib definierten Zweck erfüllen. Gleichzeitig entsteht in der Editoranwendung ein Knoten, welcher diese Komponente auch grafisch repräsentiert.

- VTK Data
- Scalarbar
- Render Node
- Time Info

F3: Der Benutzer kann von VTK Data Komponenten ausgehende Verbindungen mit einer Linie zu folgenden Komponenten erstellen:

- Scalarbar: Die Verbindung bewirkt, dass die Scalarbar die Skalare der VTK Data darstellen kann.
- Render Node: Die Verbindung bewirkt, dass die VTK Data gerendert wird.
- Time Info: Die Verbindung bewirkt, dass die Time Info die zeitlichen Informationen der VTK Data, zur Darstellung, bereitgestellt bekommt.

F4: Der Benutzer kann Scalarbar und Time Info Komponenten mit Render Node Komponenten verbinden, sodass sie gerendert werden.

F5: Der Benutzer kann Verbindungen löschen. Dies bewirkt, dass die in F3 und F4 spezifizierten Ereignisse rückgängig gemacht werden.

F6: Der Benutzer kann Komponenten löschen. Dies bewirkt, dass sie aus der Visualisierungsanwendung entfernt werden. Alle Verbindungen, die diese Komponenten hatten, werden gelöscht.

F7: Der Benutzer kann sich die Eigenschaften einer Komponente ansehen und verändern.

F8: Der Benutzer kann sich mit zusätzlichen Visualisierungsanwendungen verbinden, welche die gleiche visuelle Repräsentation der ersten Anwendung widerspiegelt.

Nichtfunktionale Anforderungen

NF1: Die Editoranwendung sollte einfach und intuitiv zu bedienen sein.

NF2: Die Editoranwendung sollte ein modernes Look&Feel haben, welches einfach zu ersetzen ist, sodass es eventuell auf den neuesten Stand gebracht werden kann.

NF3: Unit Tests müssen im White-Box-Verfahren jede Klasse und Methode auf Fehlerfreiheit überprüfen. Jeder mögliche Logikpfad muss von mindestens einem Test durchlaufen werden.

NF4: Jede Klasse und Methode muss an der Stelle ihrer Definition auch dokumentiert sein. Diese beinhaltet eine kurze Zusammenfassung der Funktionen, sowie eine Erklärung aller Parameter, Rückabewerte und geworfene Exceptions.

Technische Anforderungen

T1: Die Visualisierungsanwendung muss mindestens auf folgenden Plattformen laufen:

- Windows 7 (oder neuer)
- SUSE Linux Enterprise Desktop 12 (oder neuer)

T2: Die Editoranwendung muss mindestens auf folgenden Plattformen laufen:

- Windows 7 (oder neuer)
- SUSE Linux Enterprise Desktop 12 (oder neuer)

T3: Die Visualisierungsanwendung muss folgende Bibliotheken verwenden:

- ViSTA Virtual Reality Toolkit
- ViSTA FlowLib
- VTK

T4: Die Visualisierungsanwendung und die Editoranwendung müssen über das Netzwerk kommunizieren können.

T5: Die Visualisierungsanwendung muss die Powerwall des DLR unterstützen. Dies beinhaltet Unterstützung für einen Rechnercluster und einen Flystick.

3.6 Vorgehen

Nachdem alle Anforderungen erfasst wurden, muss ein Zeitplan festgelegt und das weitere Vorgehen spezifiziert werden. Da sich die Anwendung in drei Schichten einteilen lässt,

- Visualisierungsschicht
- Netzwerkkommunikationsschicht
- Editorschicht

wird eine aufeinander aufbauende Entwicklung durchgeführt.

Zum Zeitpunkt der Vorgehensplanung verbleiben noch 51 Personenarbeitstage zum Abschluss der Softwareentwicklung. Folgender Zeitplan legt fest, wann die einzelnen Schichten entwickelt werden:

- 20 Personenarbeitstage – Visualisierungsschicht
- 10 Personenarbeitstage – Netzwerkschicht
- 15 Personenarbeitstage – Editorschicht
- 06 Personenarbeitstage – Optimierungen und Erweiterungen

4 Entwurf

Dieses Kapitel befasst sich mit dem Entwurf der einzelnen Schichten.

4.1 Überblick

In den folgenden Sektionen werden Architekturen für die einzelnen Schichten erstellt und Technologien verglichen und ausgewählt.

4.2 Visualisierungsanwendung

Für den Entwurf der Visualisierungsschicht werden zuerst die verwendeten Technologien ViSTA Virtual Reality Toolkit und ViSTA FlowLib näher betrachtet und mögliche Alternativen vorgestellt. Danach wird die Softwarearchitektur näher beleuchtet.

4.2.1 Eingesetzte Technologien

Die im Folgenden vorgestellten Technologien wurden genutzt, weil sie vom DLR vorgegeben wurden. Dennoch werden Alternativen vorgestellt.

VTK

Das Visualization Toolkit (VTK) ist ein Open-Source Softwaresystem für 3D-Computergraphik, Modellierung, Bildverarbeitung, Volumenrendering, Wissenschaftlicher Visualisierung und Informationsvisualisierung. VTK ist eine C++ Bibliothek, bietet aber auch Wrapper für Python und Java an. Die Bibliothek ist plattformunabhängig. [31]

Datenmodell: VTK bietet ein eigenes Datenmodell an, mit dem nahezu jedes Problem, ob fiktiv oder real, dargestellt werden kann. Das Datenmodell bietet eine Datenschnittstelle basierend auf Punkten im dreidimensionalen Raum und Zellen, welche den Bereich zwischen Punkten abdecken. In Abbildung 4.1 sind VTKs Datenklassen dargestellt. Diese können sowohl mit der Bibliothek erstellt werden, als auch aus verschiedenen Dateiformaten ausgelesen werden. Hierzu bietet VTK eine Vielzahl von Lese- und Schreibklassen die sowohl VTKs eigene Dateiformate (vtk und vtp) unterstützen als auch andere Dateiformate wie zum Beispiel OpenFOAM, .OBJ und PLY.

Außerdem bietet VTK auch eine eigene Pipeline an, welche eine zeitliche Dimension zum Datenmodell hinzufügt, sodass auch zeitabhängige Visualisierungen möglich sind. [32]

Rendering: Für dreidimensionales rendering nutzt VTK eine Abstraktionsschicht über die Grafikbibliothek des Systems, welche hauptsächlich OpenGL nutzt. Es wird ermöglicht sowohl Oberflächen, Volumen und Modelle als auch Annotationen darzustellen. [33]

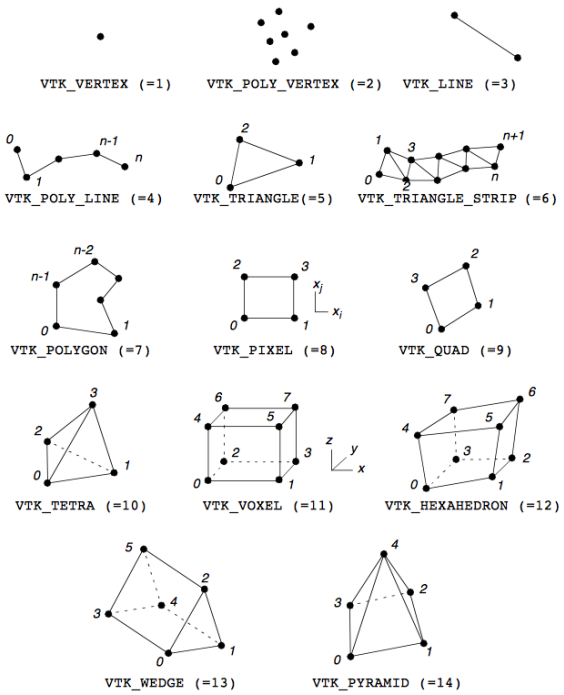

Abbildung 4.1: VTKs Datenklassen [32]

ViSTA Virtual Reality Toolkit

Das ViSTA Virtual Reality Toolkit ist ein Framework, welches von der Virtual Reality Group der RWTH Aachen entwickelt wird. Es dient zur interaktiven 3D-Visualisierung in technischen und wissenschaftlichen Bereichen.

ViSTA bietet grundlegende und allgemeine Nutzungsfunktionen der Vitual Reality zur Verfügung. Diese beinhalten:

- Anzeige-Management
- Input/Output für VR-Geräte, wie Flysticks oder Tracking-Kameras
- Schnittstelle zur OpenSG Szenegraph Bibliothek
- Cluster-Support für verteilte Anwendungen in Mehrfachanzeige-Umgebungen wie Powerwalls
- Systemunabhängige Dienste für Multithreadding, Dateisystemzugriffe usw.

ViSTA unterstützt die Großzahl aller VR-Anwednungsgebiete, was es zu einem guten VR-Framework macht. Allerdings ist es ungenügend dokumentiert, was die Entwicklung mit ViSTA sehr erschwert. [2]

ViSTA FlowLib

ViSTA FlowLib ist ein Framework zur wissenschaftlichen Visualisierung. Es baut auf dem ViSTA Virtual Reality Toolkit auf. Somit ist es sehr gut ausgelegt für Anwendungen im Virual Reality Bereich. Es bietet folgende Methoden:

- Daten- und Zeitmanagement
- Interaktionsmethoden
- Rendering-Methoden
- Partikelverfolgung

[2]

Das Datenmodell von FlowLib kann auch VTK-basierte Daten nutzen. Dafür bietet FlowLib auch Leseklassen an, die es ermöglichen VTK-Dateien einzulesen.

Leider ist ViSTA FlowLib genau so schlecht dokumentiert wie ViSTA, was die Entwicklung sehr zeitaufwändig macht.

4.2.2 Alternative Frameworks

Zwar sind die zuvor genannten Technologien schon zur Nutzung in der Visualisierungsanwendung festgelegt worden, dennoch werden hier noch ein paar Alternativen vorgestellt, mit denen die Anwendung eventuell auch entwickelt werden könnte. Für hochperformante VR-Visualisierungen bieten sich vor allem Game Engines an, da diese heutzutage sowohl VR-Umgebungen unterstützen, als auch Wert auf Performanz legen.

Unity

Unity ist eine Plattformübergreifende Game-Engine von Unity Technologies. Unity nutzt ein hochperformantes Backend welches in C++ geschrieben ist und sowohl von OpenGL, als auch Direct3D Gebrauch macht.

Programmiersprache: Nutzer der Engine müssen C# oder JavaScript als Programmiersprache nutzen. Dies kann natürlich zu eingeschränkter Leistung führen, da Garbage Collection erhebliche CPU Zeit in Anspruch nimmt. Allerdings sind Applikationen, die nicht viele Heap-Allokationen machen, davon kaum betroffen. Die meisten Strömungsvisualisierungen nutzen ein sich selten veränderndes Szenario, was Garbage Collection zu einem kleineren Problem macht.[34] Das C# eine zur Laufzeit optimierte Sprache ist, kann sogar Vorteile haben, da nicht-verändernde Szenarien gut optimiert werden. In manchen fällen kann C# sogar genau so schnell oder schneller sein als natives C++.[35] Ob dies für Strömungsvisualisierungen auch zutrifft kann aber nur ein umfangreicher Performance-Benchmark auf diesem Anwendungsgebiet belegen.

VR Unterstützung: Unity unterstützt standardmäßig einige VR-Geräte, wie das HTC Vive und das Oculus Rift[36]. Powerwalls sind nicht offiziell unterstützt, dennoch gibt es die Möglichkeit über den in Unity vorhandenen Asset Store Erweiterungen zu erwerben, die diese Funktionalität hinzufügen. So gibt es die getReal3D for Unity[37] und .middleVR[38] Erweiterungen, die eine solche Unterstützung hinzufügen.

VTK Unterstützung: VTK kann in C# unter Verwendung der ActiViz[39] Bibliothek verwendete werden. Mithilfe von VtkUnity[40], was auf GitHub zur Verfügung steht, gibt es eine Möglichkeit Unity und VTK zu verbinden, allerdings ist VtkUnity nur mit Windows kompatibel.

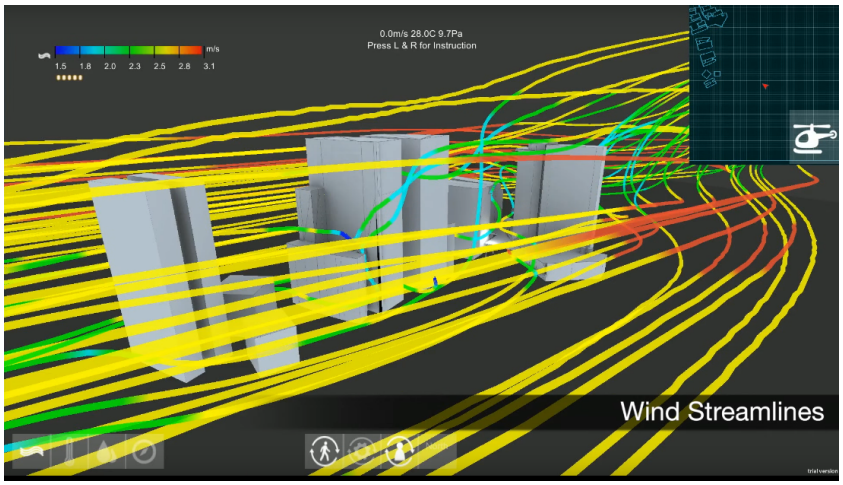

Abbildung 4.2: Eine Strömungsvisualisierung in Unity die Windbewegungen in einer Stadt darstellt [41]

Fazit: Unity kann durchaus für die Verwendung von Strömungsvisualisierungen mit VTK verwendet werden. Es gäbe aber sicherlich andere Wege Strömungsvisualisierungen in Unity umzusetzen. MegaFlow[42] zum Beispiel ist eine Unity Erweiterung, die schon Strömungsvisualisierungen auf Vektorfeldbasis anbietet.

Unreal Engine 4

Unreal Engine 4 ist eine plattformübergreifende Game-Engine von Epic-Games. Unreal Engine besitzt ein hochperformantes C++ Backend mit komplettem Quellcodezugriff für jeden. Es nutzt sowohl Direct3D als auch OpenGL zum Rendern. Seit neustem unterstützt Unreal Engine auch die Vulkan API [43].

Programmiersprache: C++ kann von Nutzern für die Entwicklung von Unreal Engine Applikationen verwendet werden, was hohe Performanz bei erfahrenen Entwicklern garantiert. [44]

VR Unterstützung: Die Unreal Engine bietet auch VR Unterstützung für einige Geräte an:

- Oculus Rift
- HTC Vive
- PlayStation VR
- Samsung Gear VR
- Google VR
- OSVR

[45]

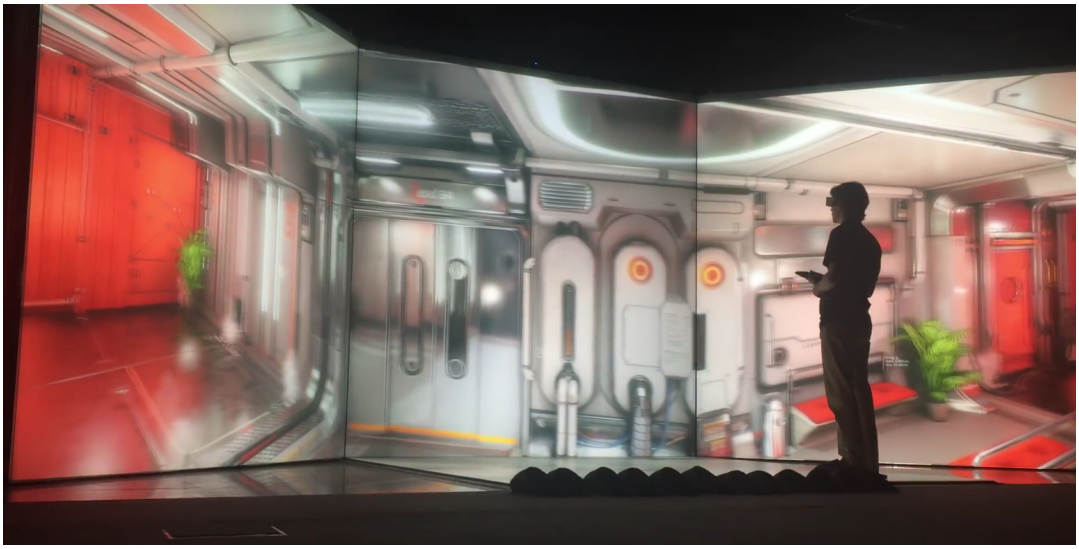

Abbildung 4.3: Eine Powerwallanwendung in der Unreal Engine mit VRCluster [46]

Powerwalls werden leider nicht standardmäßig unterstützt, aber es gibt die Erweiterung VRCluster[46], welche Unterstüzung für CAVE, Powerwallsysteme und Cluster anbietet (siehe Abbildung 4.3).

VTK Unterstützung: Da die Unreal Engine C++ verwendet und VTK eine C++ Bibliothek ist, gibt es in dieser Hinsicht keine Barrieren. Allerdings konnten keine Implementationen gefunden werden, die die Unreal Engine mit VTK erfolgreich verwenden.

Fazit: Die Unreal Engine 4 scheint alle Voraussetzungen für Strömungsvisualisierungen zu erfüllen. Allerdings müsste für ein solches Projekt vorher noch überprüft werden, ob VTK in Zusammenarbeit mit der Unreal Engine funktionsfähig ist. Falls dies nicht der Fall ist, müsste eine eigene Implementierung umgesetzt werden, die wahrscheinlich sehr Zeitaufwändig wäre.

4.2.3 Alternative Tools zur Strömungsvisualisierung

Im Folgenden werden alternative Tools zur Strömungsvisualisierung betrachtet. Es gibt einen großen Markt für solche Werkzeuge. Zwei weitverbreitete Werkzeuge werden an folgenden Punkte auf die Erfüllung der Anforderungen bewertet:

- Plattformunabhängigkeit: Es muss auf Windows und Linux verfügbar sein.
- VTK-Dateiformat: Das VTK-Dateiformat muss unterstützt werden.
- VR-Unterstützung: Powerwalls müssen unterstützt werden.
- Erweiterbarkeit: Fehlende Features sollten über Plugins implementierbar sein.

EnSight

EnSight ist ein kommerzielles Visualisierungswerkzeug für Strömungssimulationsdaten.

Plattformunabhängigkeit: EnSight ist sowohl auf Windows, als auch auf Linux verfügbar. [47]

VTK-Dateiformat: VTK-Dateiformate werden von EnSight offiziell nicht unterstützt. [48]

VR-Unterstützung: EnSight unterstützt viele VR-Technologien. Powerwalls und CAVES sind auch darunter. [49]

Erweiterbarkeit: EnSight bietet keinerlei Möglichkeit durch Erweiterungen Features hinzuzufügen. So kann auch der VTK-Support nicht nachgerüstet werden.

ParaView

ParaView ist ein Open Source WErkzeug zur Datenanalyse und Visualisierung.

Plattformunabhängigkeit: ParaView ist für Linux und Windows erhältlich. [50]

VTK-Dateiformat: VTK-Daten werden von ParaView unterstützt. [51]

VR-Unterstützung: Powerwalls, sowie Render-Cluster werden von ParaView unterstützt. [52]

Erweiterbarkeit: ParaView bietet umfangreiche Möglichkeiten zur Plugin-Entwicklung an. Dementsprechend können fehlende Features selbst implementiert werden oder sie sind schon in anderen Plugins implementiert. [53] Da ParaView Open-Source ist, kann man dies auch beliebig erweitern und weiterentwickeln.

Fazit:

Da EnSight das VTK-Format nicht unterstützt, entspricht es den Anforderungen nicht. ParaView hingegen, kann alle Anforderungen erfüllen und würde somit zur Nutzung von Strömungsvisualisierungen in Frage kommen.

4.2.4 Architektur

Basierend auf den Anforderungen und den eingesetzten Technologien wurde eine Architektur für die Visualisierungsschicht entworfen. Diese wird in den folgenden Kapiteln erläutert.

Knoten und Kanten

Da ein knotenbasierter Ansatz gefordert ist, müssen Klassen für Knoten und Kanten erstellt werden. Diese müssen die Möglichkeit bieten zwei Knoten mithilfe einer Kante zu Verbinden.

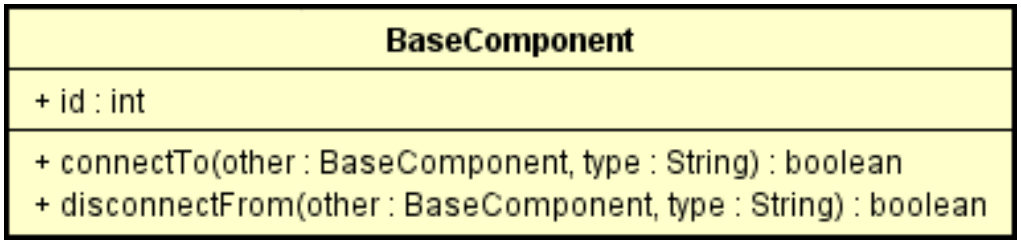

Abbildung 4.4: Die BaseComponent Klasse repräsentiert einen Knoten

In Abbildung 4.4 ist die BaseComponent Klasse definiert. Sie repräsentiert einen Knoten in der Anwendung. Jede Komponente bekommt eine einzigartige ID, um sie später leichter identifizieren zu können. Jede Komponente muss in der Lage sein sich mit anderen Komponenten zu verbinden. Dies wird durch die definierten Methoden ermöglicht.

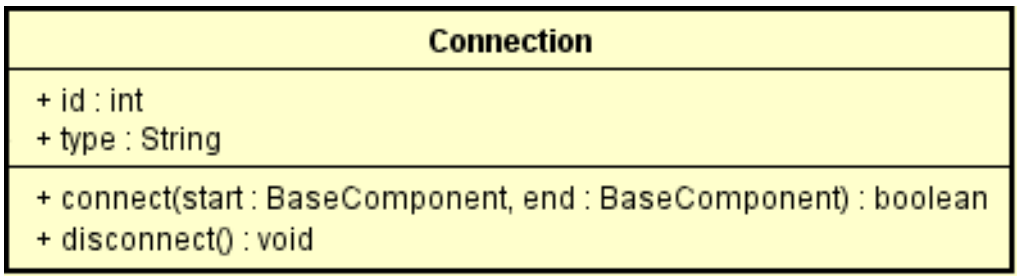

Abbildung 4.5: Die Connection Klasse repräsentiert eine Kante

In Abbildung 4.5 ist die Connection Klasse definiert. Auch jede Kante besitzt eine einzigartige ID um sie leichter zu identifizieren. Außerdem hat jede Kante einen Typ. Dieser soll verhindern, dass Verbindungen wahllos gemacht werden können. Die connect und disconnect Methoden stellen die Funktionalität zum Verbinden und Lösen von Verbindungen bereit.

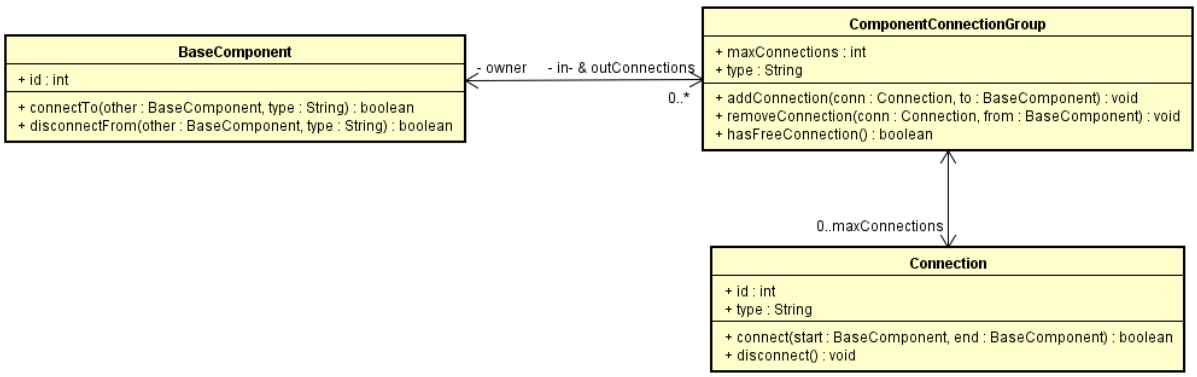

Abbildung 4.6: Die Beziehung zwischen Kanten und Knoten mithilfe der ComponentConnectionGroup Klasse

In Abbildung 4.6 ist die Beziehung zwischen den Knoten und Kanten beschrieben. Sie bedient sich der Hilfsklasse ComponentConnectionGroup. Jeder Knoten besitzt sowohl für eingehende als auch für ausgehende Verbindungen eine Gruppen. Die Aufgabe soclher Gruppen ist es alle Verbindungen eines Typs, die einen Knoten besitzt, zu verwalten. Hierfür besitzt die Klasse zwei Attribute. Das erste Attribut spezifiziert welche Verbindungstypen die Gruppe verwaltet und das zweite Attribut legt die maximale Anzahl erlaubter Verbindungen für diese Gruppe fest.

In Abbildung 4.7 ist das Ganze grafisch dargestellt. Die einzelnen ComponentConnectionGroups sind als farbige Kästen dargestellt. Wie man sehr gut erkennen kann besitzt eine Gruppe nur Verbindungen von einem Typen.

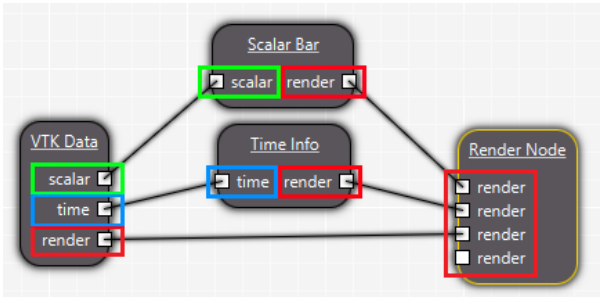

Abbildung 4.7: Grafische Darstellung der in Abbildung 4.6 beschriebenen Beziehungen

Abbildung von FlowLib-Komponenten auf Knoten

Da FlowLib-Komponenten miteinander verbunden werden müssen, muss die Architektur das berücksichtigen. Die verwendete Programmiersprache C++ erlaubt einen polymorphen Ansatz zur Lösung dieses Problems.

Abbildung 4.8 stellt einen Ausschnitt aus einer solchen Architektur dar. Es gibt zwei von BaseComponent erbende Klassen, die eine Beziehung zueinander haben.

RenderNodeComponent verpackt einen VflRenderNode (eine Klasse der VistaFlowLib), dieser hat die Funktion IVflRenderables (ebenfalls eine Klasse der VistaFlowLib) zu rendern. IVflRenderables

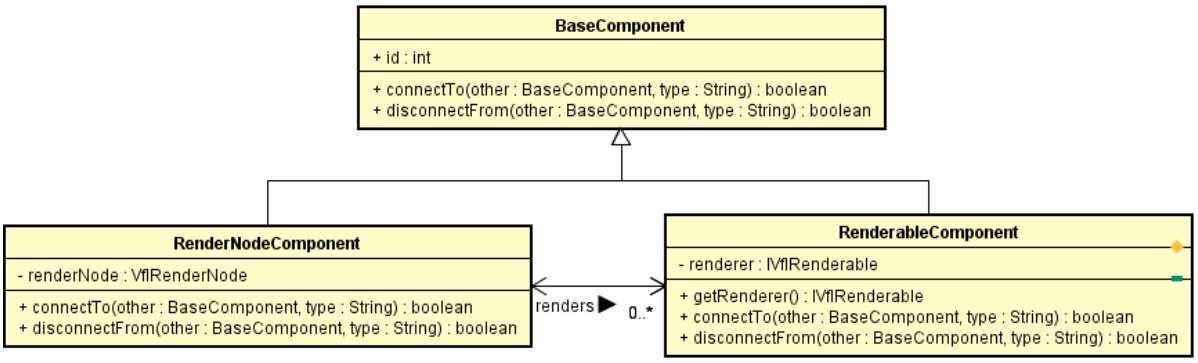

Abbildung 4.8: Polymorphe Softwarearchitektur zum abstrahieren der FlowLib-Komponenten

sind in die Klasse RenderableComponent verpackt.

Beide Klassen überschreiben die connect und disconnect Methoden der BaseComponent Klasse, sodass die dazugehörige FlowLib Logik ausgeführt wird, sobald diese Komponenten miteinander verbunden oder voneinander getrennt werden. Dadurch ist es möglich zwei Komponenten ohne Wissen über ihre Implementierung zu verbinden. Die einzige Voraussetzung für eine solche Verbindung ist die Akzeptanz gleicher Verbindungstypen.

Schnittstelle

Um später möglichst einfach aus der Netzwerkschicht auf die Visualisierungsschicht zugreifen zu können, muss eine Schnittstelle bereitgestellt werden, die alle benötigten Funktionen möglichst nach außen vertritt.

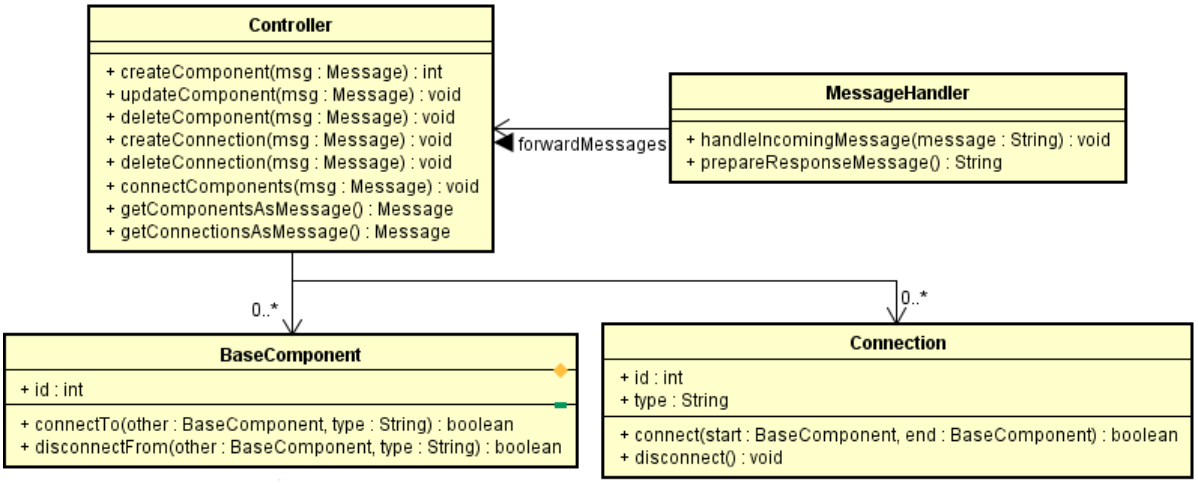

Abbildung 4.9: Eine simple Schnittstelle, die alle notwendigen Funktionen für ein knotenbasiertes System bereitstellt

Um diese Anforderung zu erfüllen werden zwei Klassen bereitgestellt (siehe Abbildung 4.9). Einen MessageHandler und ein Controller. Der MessageHandler wertet eingehende Nachrichten aus und je

nach Nachricht leitet er diese an den Controller weiter. Der Controller bietet dafür ein sehr übersichtliches Interface an. Folgende Methoden sind für den Aufruf über das Netzwerk bestimmt:

- Knoten erstellen (createComponent)
- Knoten updaten (updateComponent)
- Knoten löschen (deleteComponent)
- Kante erstellen (createConnection)
- Kante löschen (deleteConnection)
- Knoten verbinden (connectComponents)

Die Methoden getComponentsAsMessage und getConnectionsAsMessage werden für die Antworten verwendet. Der Controller verwaltet somit alle Knoten, Verbindungen und die Beziehungen zwischen ihnen.

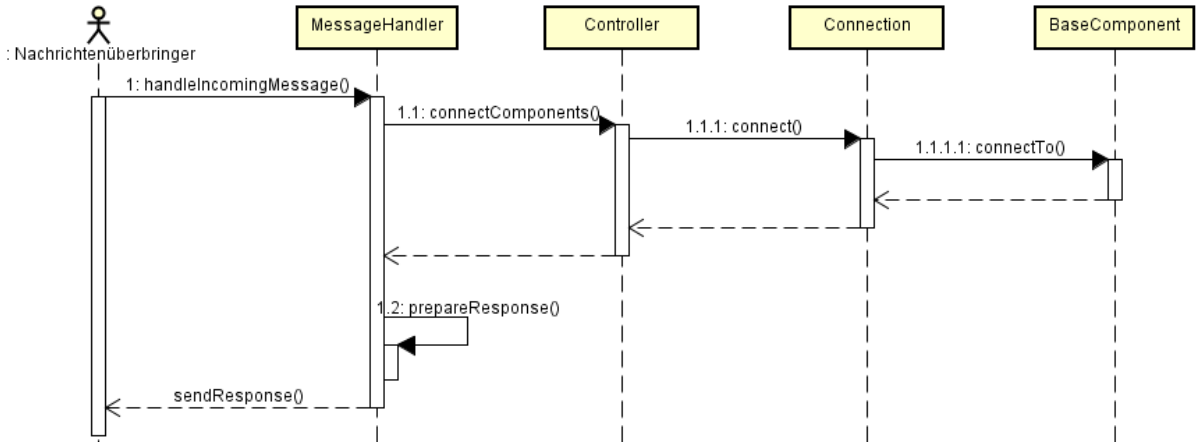

Abbildung 4.10: Ein einfacher Programmablauf zum Abarbeiten einer Nachricht

Abbildung 4.10 beschreibt einen einfachen Ablauf, den eine eingehende Nachricht auslöst. Aus der Netzwerkschicht kommt eine Nachricht beim MessageHandler an. Dieser schaut sich die Nachricht an und ruft eine entsprechende Methode beim Controller auf. In diesem Fall enthält die Nachricht Anweisungen zwei Knoten miteinander zu verbinden. Dies wird an eine Connection weitergeleitet, welche dann die zwei BaseComponents miteinander verbindet. Nachdem dies erfolgreich abgearbeitet wurde, bereitet der MessageHandler eine Antwort mit Informationen über die neue Struktur vor und gibt diese an die Netzwerkschicht zurück.

4.3 Kommunikation

Da Visualisierungsanwendung und Editoranwendung rechnerübergreifend kommunizieren müssen, muss ein Framework gefunden werden, welches dies zuverlässig garantiert. Hierzu werden in den folgenden Kapiteln Netzwerkbibliotheken und Datenformate zur Nachrichtenkommunikation verglichen, um die für diesen Anwendungsfall beste Bibliothek, auszuwählen.

4.3.1 Vergleich von Netzwerkbibliotheken

In diesem Kapitel werden verschiedene Netzwerkbibliotheken zum Nachrichtenaustausch untersucht. Bewertet wird anhand folgender Punkte:

- Sprachsupport: C muss unterstützt werden, andere Sprachen erwünscht
- Plattformübergreifend: Windows und Linux müssen unterstützt werden; weitere Plattformen wie Android und iOS sind erwünscht
- Standalone: Es sollten keine Abhängigkeiten zu anderen Bibliotheken geben
- Dokumentation: Es sollte eine gute Dokumentation vorliegen
- Benutzerfreundlichkeit: Die Bibliothek sollte einfach zu benutzen sein, aber darf dabei keine Kompromisse machen
- Support: Es sollte gute Unterstützung der Entwickler und der Community geben
- Kosten: Die Bibliothek sollte kostenlos sein

ZeroMQ

ZeroMQ ist eine Open-Source Messaging Library lizenziert unter der LGP Lizenz [54].

Sprachsupport: ZeroMQ ist in C geschrieben und unterstützt zur Zeit 50 Sprachen [55]. Darunter finden sich alle modernen Sprachen von C++ über C# bis hin zu Java.

Plattformübergreifend: ZeroMQ kapselt System Ressourcen ab und unterstützt Plattformen wie Windows, Linux, Android und iOS.

Standalone: ZeroMQ hat keine Abhängigkeiten zu anderen Bibliotheken.

Dokumentation: ZeroMQ ist sehr gut dokumentiert. Es liegt vor allem Dokumentation für das C-Binding vor, allerdings bedienen sich alle anderen Sprachen einer nahezu identischen Syntax, weshalb die C-Dokumentation völlig ausreicht. Zudem gibt es einen umfangreichen Guide [56], der alle Bibliotheksfeatures mit Beispielen in mehreren Sprachen vorstellt.

Benutzerfreundlichkeit: ZeroMQ bietet ein extrem simples aber dennoch mächtiges Interface an. Mit lediglich drei Datenstrukturen und einigen wenigen Funktionen lassen sich einfach sehr komplexe Systeme bauen.

ZeroMQ übernimmt dabei eine Vielzahl an Aufgaben im Hintergrund. Die Bibliothek kümmert sich um das Versenden und Empfangen von Nachrichten und garantiert, dass alle Nachrichten in der korrekten Reihenfolge ankommen. Außerdem stellt sie sicher, dass kurze Verbindungsabbrüche nicht zum Komplettausfall der Kommunikation führen, indem Verbindungen nach einem Abbruch automatisch wiederhergestellt werden.

Allerdings bietet ZeroMQ zur Nachrichtenübertragung nur wenige Funktionen. Es kann lediglich Bytes versenden und empfangen. Für die Interpretation der Byte-Nachrichten ist der Programmierer selbst verantwortlich.

Die Mächtigkeit von ZeroMQ kommt von den sogenannten Pattern. Es folgt eine Vorstellung von zwei dieser Pattern.

Request-Reply Pattern: Das Request-Reply Pattern ist das wohl einfachste Pattern, welches ZeroMQ bietet. Es sichert die Kommunikation zwischen genau zwei Netzwerkendpunkten. Ein Endpunkt übernimmt die Rolle des Servers; der andere die Rolle des Clients. Zudem wird einer der beiden Endpunkte Requester; der andere Replier.

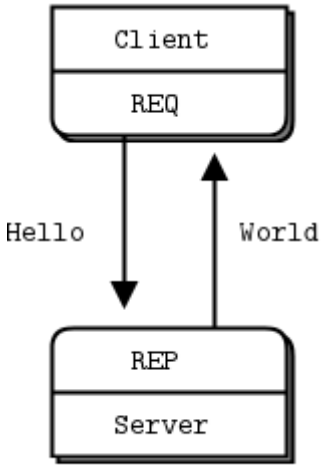

Abbildung 4.11: ZeroMQs Request-Reply Pattern [56]

In Abbildung 4.11 ist dieses Pattern zu sehen. Der Client verbindet sich mit dem Server und nach erfolgreichem Verbindungsaufbau sendet der Client eine Request-Nachricht an den Server. Danach kann der Client keine weiteren Nachrichten an den gleichen Server schicken, solange er keine Antwort erhalten hat. Sobald der Server geantwortet hat, steht diese Funktion aber wieder zur Verfügung. Hier findet die Kommunikation bidirektional und abwechselnd zwischen zwei Teilnehmern statt.

Publish-Subscribe Pattern: Das Publish-Subscribe Pattern kann mehr als zwei Teilnehmer haben. Es gibt einen Publisher und beliebig viele Subscriber. In der Regel ist der Publisher der Server und alle Subscriber Clients – davon kann aber abgewichen werden.

In Abbildung 4.12 ist das Publish-Subscribe Pattern dargestellt. Dort sind mehrere Subscriber zu einem Publisher verbunden. Die Kommunikation ist in diesem Pattern einseitig. Der Publisher sendet immer Nachrichten an die Subscriber. Er braucht nicht auf Antworten warten. Es gibt für ihn auch keine Möglichkeit herauszufinden wer mit ihm verbunden ist. Deshalb kann er auch dann Nachrichten versenden, wenn kein Subscriber mit ihm verbunden ist. Es ist Vergleichbar mit dem Observer-Pattern aus der Softwareentwicklung.

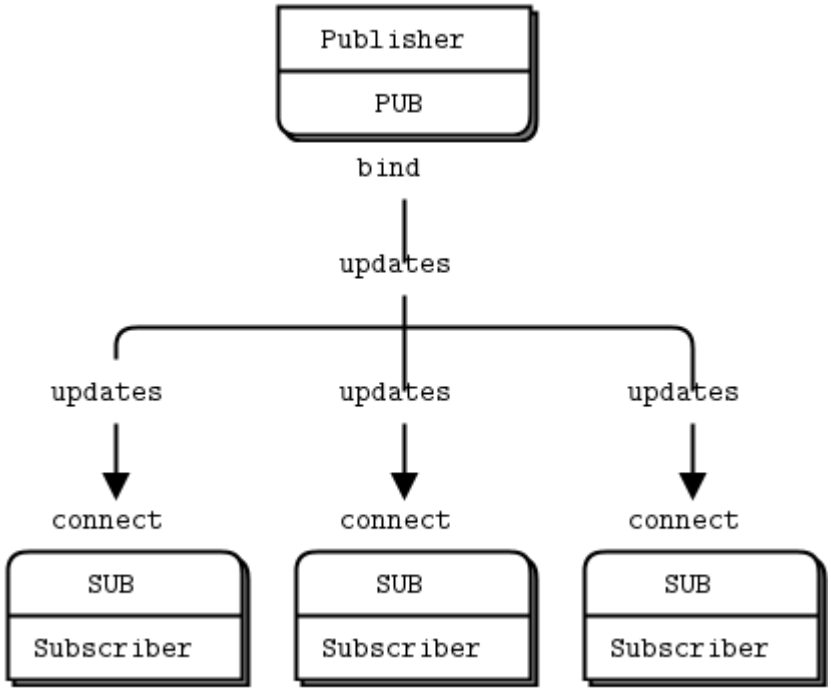

Abbildung 4.12: ZeroMQs Publish-Subscribe Pattern [56]

Support: Da die Bibliothek Open-Source entwickelt wird, ist die Community gleichzeitig Entwickler. Neben der umfangreichen Dokumentation ist es auch möglich über Mailinglisten, Chatrooms und Bugtracker Unterstützung zu erhalten.

Kosten: ZeroMQ ist Open-Source und somit frei verfügbar.

RabbitMQ

RabbitMQ ist eine Open-Source Message Broker Bibliothek, lizenziert unter der Mozilla Public Lizenz. Sie ist von Pivotal Software Entwickelt und in Erlang geschrieben.

Sprachsupport: RabbitMQ unterstützt zurzeit über 25 Sprachen, darunter sind alle modernen Sprachen, wie C, Java und C#. [57]

Plattformübergreifend: RabbitMQ ist auf vielen Plattformen verfügbar. Diese beinhalten Linux, Windows, Mac OS X und ein paar weitere Plattformen. Es gibt keinen offiziellen Support für Android und iOS. [58]

Standalone: RabbitMQ hat keine Abhängigkeiten an andere Bibliotheken, allerdings muss ein dedizierter RabbitMQ Server laufen um Nachrichten verschicken zu können.

Dokumentation: RabbitMQ hat eine umfangreiche Dokumentation für alle unterstützten Sprachen, inklusive Tutorials, mit vielen Anwendungsbeispielen. [59, 60]

Benutzerfreundlichkeit: RabbitMQ nutzt jede Sprache zu ihrem vollen Potential, indem es alle Sprachfeatures und Bestpractices für die jeweilige Sprache verwendet. Zum Beispiel wird in Java das Factory-Pattern benutzt, während in Go eher Low-Level-Bindings bevorzugt werden.

Es nutzt einen dedizierten Server als Middleware um Nachrichten weiterzuleiten. Die Komplexität der zu schreibenden Clientsoftware hängt ganz von Sprache und Anwendungsgebiet ab. RabbitMQ bedient sich sehr hohen Abstraktionsniveaus, welches für gute Skalierbarkeit sorgt.

Support: RabbitMQ hat eine sehr aktive Community, die es ständig weiterentwickelt und wird kommerziell durch Pivotal Software unterstützt.

Kosten: RabbitMQ ist Open-Source und damit frei zur Verfügung.

nanomsg

Nanomsg ist eine Open-Source Socket-Bibliothek lizenziert unter der MIT/X11 Lizenz. [61]

Sprachsupport: Nanomsg ist in C implementiert und unterstützt momentan 24 weitere Sprachen. Darunter sind unter anderem C++, C# und Java. [62]

Plattformübergreifend: Nanomsg ist auf einer Vielzahl von Plattformen verfügbar. Darunter: Linux, Windows, Android und iOS.

Standalone: Nanomsg hat keine weiteren Abhängigkeiten und kann somit als Standalone verwendet werden. [61]

Dokumentation: Nanomsg ist relativ gut dokumentiert, allerdings ist dies von der verwendeten Sprache abhängig. Da sich aber alle Sprachen an der C-Implementierung orientieren, reicht es völlig aus diese zu kennen. Desweiteren sind auch Tutorials verfügbar, welche einen Einstieg in die Nutzung der Bibliothek geben. [62]

Benutzerfreundlichkeit: Nanomsg orientiert sich an Posix-Sockets und es ist möglich mit wenigen Datenstrukturen komplexe Netzwerkarchitekturen zu entwickeln. Hierbei bedient es sich mehrerer Patterns:

- PAIR
- BUS
- Request-Reply
- Publish-Subscribe

- Pipeline
- SURVEY

[61]

Dies ermöglicht mit wenig Aufwand skalierbare Netzwerkkommunikation zu implementieren. Nanomsg kann jedoch nur Byte-Nachrichten verschicken, was die Interpretation von Nachrichten in die Hände des Entwicklers legt.

Support: Da Nanomsg Open-Source entwickelt wird hat es eine sehr aktive Community, welche über Mailinglisten und Chatrooms erreichbar ist. [63]

Kosten: Nanomsg ist Open-Source und somit frei verfügbar.

Fazit

RabbitMQ scheidet aus, da es einen dedizierten Server benötigt. Dies ist eine Komplexität, die für dieses Anwendungsgebiet nicht nötig ist.

ZeroMQ und nanomsg sind sehr ähnlich in ihrer Benutzung. Beide sind sehr minimal gehalten, bieten aber mächtige Patterns an, um skalierbare Anwendungen zu bauen. ZeroMQ allerdings unterstützt mehr Sprachen und hat eine umfangreichere Dokumentation inklusive eines Buches, welches ZeroMQ in der Tiefe behandelt und viele praktische Beispiele liefert. Auch wenn beide Bibliotheken die gleichen Plattformen unterstützen scheint ZeroMQ doch einen zuverlässigeren Support zu bieten als nanomsg.

ZeroMQ wird damit für die Netzwerkschicht als Bibliothek verwendet.

4.3.2 Vergleich von Datenformaten zur Nachrichtenübertragung

Da ZeroMQ nur Bytes verschicken kann, muss ein Datenformat gefunden werden, was den Anforderungen entspricht. Es wird nach folgenden Kriterien entschieden.

- Sprachunterstützung: C muss mindestens unterstützt werden
- Benötigte Bibliothek: Um Abhängigkeiten gering zu halten sind Formate bevorzugt, welche keine Bibliothek benötigen
- Komplexität: Wie komplex ist das Datenformat
- Dokumentation: Wie gut ist das Datenformat dokumentiert

JSON

JavaScript Object Notation ist ein leichtgewichtiges Datenformat. Es ist von Menschen les- und schreibbar. Es nutzt ein Textformat das durch den ECMA-404 The JSON Data Interchange Standard [64] definiert ist. [65]

Sprachunterstützung: JSON wird von über 50 Sprachen unterstützt; C, C++, Java und C# sind darin enthalten. [65]

Benötigte Bibliothek: Nur wenige Sprachen kommen mit JSON-Unterstützung in der Standardbibliothek. C/C++ gehören nicht dazu, weshalb die Visualisierungsschicht eine externe Bibliothek nutzen müsste.

Komplexität: JSON nutzt eine sehr einfache Syntax, welche auf zwei Datenstrukturen beruht.

- Eine Ansammlung von Schlüssel/Wert Paaren
- Eine geordnete Liste von Werten

[65]

Dokumentation: Das Datenformat ist sehr gut dokumentiert. Es ist so einfach strukturiert, dass man es innerhalb von wenigen Minuten lernen kann. [65]

XML

Die Extensible Markup Language (XML) ist ein einfaches vom Mensch les- und schreibbares Datenformat in Textform. Es ist von www.w3.org spezifiziert worden [66].

Sprachunterstützung: Nahezu alle Sprachen haben XML Unterstützung. C, C++, Java und C# sind darin enthalten.

Benötigte Bibliothek: C und C++ haben zwar keine XML Unterstützung in der Standardbibliothek, aber ViSTA bringt schon einen XML-Parser mit, der auf TinyXML [67] aufbaut. Auch viele andere moderne Sprachen unterstützen XML bereits in der Standardbibliothek, wie zum Beispiel C# und Java. Somit kann, unter Verwendung der meisten Sprachen, auf eine externe Abhängigkeit verzichtet werden.

Komplexität: XML bietet eine relativ simple Syntax an. Es besteht hauptsächlich aus zwei Datenstrukturen:

- Elemente – können Text oder andere Elemente enthalten
- Attribute – werden an Elementdefinitionen angehängt und bestehen aus Schlüssel/Wert-Paaren

XML ist somit sehr einfach zu verstehen und zu benutzen.

Dokumentation: XML ist sehr gut auf den Seiten des W3C dokumentiert und mit vielen Tutorials anwendungsbezogen erklärt. [68]

Protocol Buffers

Protocol Buffers ist eine von Google entwickelte Bibliothek zur Serialisierung von Datenstrukturen.

Sprachunterstützung: Protocol Buffers wird momentan von 9 Sprachen unterstützt. Darin enthalten sind Java, C++ (kein C) und C#. [69]

Benötigte Bibliothek: Protocol Buffers ist in keiner Sprache in der Standardbibliothek enthalten und benötigt deshalb immer eine externe Bibliothek.

Komplexität: Protocol Buffers ist sehr simpel und ähnelt in seiner Syntax einer modernen, objektorientierten Sprache. Nachrichten müssen in einer .proto-Datei definiert werden. Diese muss kompiliert werden. Das kompilieren erzeugt Datenklassen, welche die Nachrichten repräsentieren. Diese Klassen bietet dann Methoden zum Verändern, Serialisieren und Deserialisieren an. [70]

Dokumentation: Protocol Buffers ist sehr gut dokumentiert. Sowohl die .proto Syntax als auch die Syntax der unterstützten Sprachen ist sehr umfangreich und mit Tutorials erklärt. [70]

Fazit

Alle drei betrachteten Technologien könnten eingesetzt werden, um als Nachrichtenformat zu dienen. JSON punktet durch seine extrem einfache Syntax, hat dafür aber nahezu keine Unterstützung in Standardbibliotheken. XML ist auch nicht sonderlich komplex und würde wahrscheinlich keine externe Bibliothek benötigen. Protocol Buffers benötigt immer eine externe Bibliothek und benötigt zusätzlich noch einen Compiler für jede Sprache in der es eingesetzt wird. Es hat auch die schlechteste Sprachunterstützung von den drei Technologien. Protocol Buffers punktet vor allem mit Performanz, welche aber nicht relevant für ein Projekt dieser Größe ist.

Da weniger externe Abhängigkeiten, gerade bei C++ Anwendungen, von großem Vorteil sind, wird XML als Nachrichtenformat gewählt. Dies garantiert schnellere Compilierzeiten und bessere Plattformunabhängigkeiten.

4.3.3 Architektur

Da die Technologien zur Netzwerkkommunikation feststehen, kann die Architektur geplant werden. Dafür muss ein ZeroMQ-Design-Pattern ausgesucht und die XML-Struktur festgelegt werden.

Auswählen des Design-Patterns

Das Designpattern muss folgende Anforderungen erfüllen:

- Es ist möglich sich mit mehreren Visualisierungen zu verbinden.
- Jede Visualisierung muss Nachrichten erhalten können zum Erstellen, Updaten und Löschen von Komponenten und Verbindungen
- Jede Visualisierung muss daraufhin in einer Antwort mitteilen, welche Komponenten und Verbindungen es gibt, inklusive aller Eigenschaften

Das Publish-Subscribe Pattern würde sich anbieten, da jede Nachricht vom Editor zu allen Visualisierungen geschickt werden muss, allerdings haben die Visualisierungen keine Antwortmöglichkeit bei diesem Pattern.

Deshalb wird das Request-Reply-Pattern gewählt, da so der Editor jeder Visualisierung die selbe Nachricht schicken und separate Antworten erhalten kann.

XML-Struktur

Die XML Struktur muss folgende Anforderungen erfüllen:

- Es gibt Komponenten, welche Typ, ID und Eigenschaften besitzen
- Es gibt Verknüpfungen, welche Typ und ID besitzen und Komponenten Verbinden
- Komponenten können nur Verknüpfungen von bestimmten Typen akzeptieren
- Es muss propagiert werden, welche Komponententypen die Visualisierung unterstützt
- Komponenten müssen erstellt, aktualisiert und gelöscht werden können
- Verbindungen müssen erstellt und gelöscht werden und Komponenten verbinden können

Hierfür werden zwei Arten von Nachrichten definiert:

1. Nachrichten für (Editor –> Visualisierung)
2. Nachrichten für (Visualisierung –> Editor)

Requests: Folgende Request-Nachrichten werden definiert:

```
<createComponent type="COMPONENT_TYPE"/>
```

```
<updateComponent id="COMPONENT_ID">
  <properties>
    <property name="PROPERTY_NAME" value="VALUE"/>
    <property name="PROPERTY_NAME" value="VALUE"/>
    ...
    ...
  </properties>
</updateComponent
```

```
<deleteComponent id="COMPONENT_ID"/>
```

```
<createConnection type="CONNECTION_TYPE"/>
```

```
<deleteConnection id="CONNECTION_ID"/>
```

```
<connectComponents from="COMPONENT_ID" to="COMPONENT_ID" id="CONNECTION_ID"/>
```

Mit diesen sechs simplen Nachrichten kann der Editor die gesamte Visualisierung steuern.

Replies: Jeder Reply ist gleich aufgebaut. Er folgt folgendem Muster:

1. Sektion: Verfügbare Komponenten-Typen
2. Sektion: Existierende Komponenten
3. Sektion: Existierende Verbindungen

Für ein umfangreiches Beispiel siehe A.1 im Anhang. Ansonsten werden die einzelnen Sektionen hier nur getrennt betrachtet.

Sektion: Verfügbare Komponenten-Typen: Hier wird beschrieben, welche Komponenten zur Erstellung zur Verfügung stehen.

```
<componentTypes>
  <componentType type="TYPE"/>
  <componentType type="OTHER_TYPE"/>
  ...
</componentTypes>
```

Sektion: Existierende Komponenten: Komponenten bestehen aus drei Sektionen:

1. Sektion: Eingehende Verbindungen
2. Sektion: Ausgehende Verbindungen
3. Sektion: Eigenschaften

Für ein umfangreiches Beispiel siehe A.1 im Anhang.

Sektion: Existierende Verbindungen: Die Verbindungen werden in diese Sektion aufgelistet:

```
<connections>
  <connection id="CONNECTION_ID" type="TYPE"
              start="COMPONENT_ID"
              end="COMPONENT_ID"/>
  ...
</connections>
```

Netzwerkkommunikation: Visualisierungsschicht

Um die Netzwerkschicht mit der Visualisierungsschicht zu verknüpfen muss sowohl eine Schnittstelle definiert werden, als auch das Serialisieren in das XML-Format erfolgen.

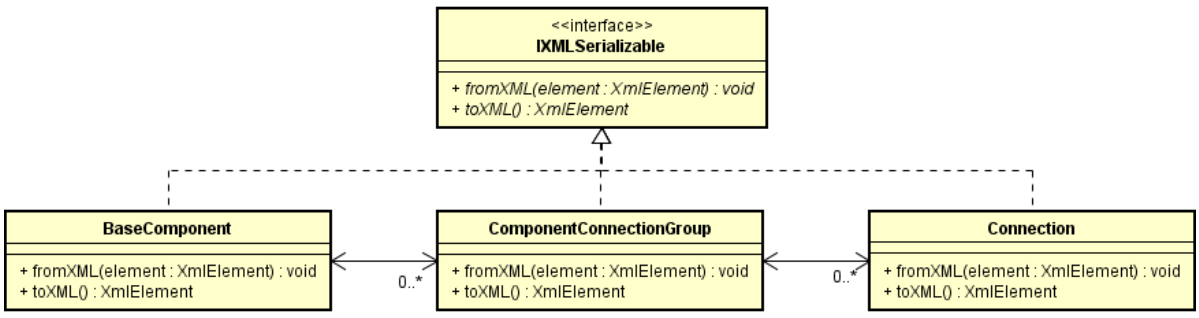

Abbildung 4.13: Ein Interface zum Serialisieren

Serialisieren: Es wird ein neues Interface in die Visualisierungsanwendung eingefügt, welches zwei Methoden bereitstellt.

In Abbildung 4.13 ist das neue Interface zur Serialisierung abgebildet. Es bietet abstrakte Methoden zur Serialisierung und Deserialisierung an. Die Klassen Connection, ComponentConnectionGroup und BaseComponent (sowie alle Kindklassen von BaseComponent) implementieren dieses Interface.

Schnittstelle: An der Schnittstelle wird eine neue Klasse definiert, die sich um das Empfangen und Absenden von Nachrichten kümmert.

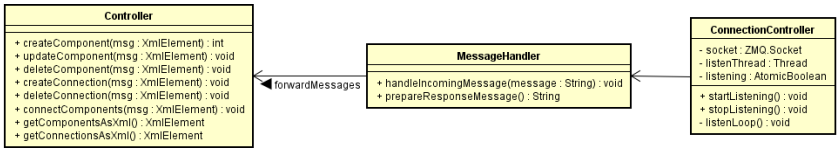

Abbildung 4.14: Die Netzwerkkommunikationsschnittstelle auf der Visualisierungsseite

In Abbildung 4.14 ist eine neue Klasse zur Netzwerkkommunikation hinzugekommen. Sie hat die Aufgabe Nachrichten zu empfangen, diese an den MessageHandler weiterzuleiten und eine Antwort vom MessageHandler zurückzuschicken.

Netzwerkkommunikation: Editorschicht

Die Editorschicht benötigt ebenfalls eine Schnittstelle zur Netzwerkschicht, um Nachrichten an mehrere Visualisierungen zu verschicken und Antworten zu erhalten.

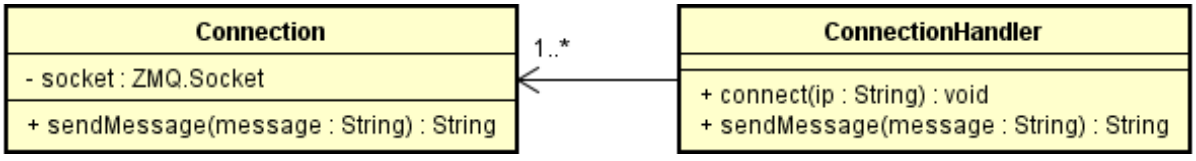

Abbildung 4.15: Die Netzwerkkommunikationsschnittstelle auf der Editorseite

In Abbildung 4.15 ist eine solche Architektur mit zwei Klassen implementiert.

Die ConnectionHandler-Klasse ist Schnittstelle zur Editorapplikation. Sie verwaltet alle Verbindungen, nimmt Nachrichten an und leitet diese zum Versenden an alle Verbindungen weiter. Sie sammelt auch alle Antworten zur Bearbeitung ein. Die Connection-Klasse ist die Schnittstelle zum Netzwerk. Sie sendet und empfängt Nachrichten.

4.4 Editoranwendung

Die Editoranwendung muss die grafischen Anforderungen an die Software erfüllen. Um dies zu gewährleisten muss ein GUI-Framework ausgewählt werden, mit dem diese Anforderungen implementiert werden können. Hier noch einmal die wichtigsten Anforderungen zusammengefasst:

- Plattformunabhängigkeit
- Mögliche Netzwerkschnittstelle
- Erstellen von Knoten
- Verknüpfen von Knoten über Kanten

4.4.1 Vergleich von GUI-Frameworks

Zur Auswahl eines Frameworks wird anhand folgender Kriterien bewertet:

- Sprachunterstützung: Es muss mindestens eine Sprache unterstützt werden, die ZeroMQ und XML zur Kommunikation unterstützt.
- Plattformunabhängigkeit: Das Framework sollte auf so vielen Plattformen wie möglich laufen. Windows und Linux müssen unterstützt werden, während Android und iOS wünschenswert sind.
- Modernes Design: Das Design der Benutzeroberfläche sollte modern sein und die Möglichkeit bieten es in Zukunft anzupassen.
- Support: Es muss gut von der Community und den Entwicklern unterstützt werden.
- Erfahrung: Da nur 15 Personenarbeitstage für die Entwicklung zur Verfügung stehen, ist die Kenntnis über die Funktionsweise des Frameworks ein wichtiger Punkt.
- Sonstiges: GUI-Frameworks bringen viele Features mit, welche einzigartig für sie sind.

QT

Das QT-Framework ist ein hardwarebeschleunigtes C++ GUI-Framework von der Firma The Qt Company. QT ist sowohl kommerziell, als auch als Open Source, unter der GPL Lizenz, verfügbar. [71]

Sprachunterstützung: QT unterstützt offiziell nur C++ zur Entwicklung, allerdings sind 16 weitere Sprachen über Drittanbieter unterstützt. Darunter befinden sich Sprachen, wie Java, Ruby und Python. [72]

Plattformunabhängigkeit: QT unterstützt die großen Desktopplattformen wie Linux, Windows und OS X, aber auch mobile und embedded Plattformen, wie Android, iOS, WinRT und QNX. [73] Allerdings sind dies nur die von C++ offiziell unterstützten Plattformen. Die anderen Sprachen unterstützen meist weniger Plattformen.

Modernes Design: Es gibt verschiedene Möglichkeiten QT-Applikationen zu designen. Alle QT-Anwendungen verwenden, wenn nicht anders spezifiziert, das native Plattform Look&Feel. Es besteht aber die Möglichkeit eigene Styles und Designs zu entwickeln, welche in QML deklariert werden. QML ist eine von CSS und JavaScript inspirierte Sprache, welche die GUI beschreibt. Dies ermöglicht eine klare Trennung von GUI und Logik, sodass die GUI einfach auf dem neuesten Stand zu halten ist. [74]

Support: Da QT sehr verbreitet ist, ist der Support, gerade von der Community, sehr gut. Mit der kommerziellen Lizenz ist auch offizieller Support von der QT Company verfügbar [75]. Falls die Open Source Version verwendet wird, ist es immer noch möglich sich umfangreich zu informieren oder Hilfe von der Community zu erhalten. So hat QT eine umfangreiche Dokumentation inklusive Tutorials und Beispielen[76], ein eigenes Wiki[77] und ein gut besuchtes Forum[78].

Erfahrung: Im Rahmen des Studiums wurden kleine Anwendungen zur Bildbearbeitung entwickelt, welche Gebrauch von QT machten. Dabei wurden aber nur wenige Features des Frameworks genutzt.

Sonstiges: QT bietet einen plattformunabhängigen Editor an, mit dem man UIs per Drag&Drop erstellen kann. Dieser unterstützt auch die C++ Entwicklung. [79] Des weiteren bringt QT eine umfangreiche C++ Bibliothek mit, welche mit einer Standartbibliothek, wie die von Java oder C# vergleichbar ist. [80]

JavaFX

Das von Oracle entwickelte hardwarebeschleunigte GUI-Framework ist das neueste in der Java Welt. Es wurde entwickelt um Swing abzulösen und es ist seit Java 8 offiziell im JDK und JRE von Oracle enthalten.

Sprachunterstützung: Da JavaFX auf der Java Plattform läuft, ist es von allen Sprachen unterstützt, die auf dieser Plattform laufen. Dies beinhaltet Java, Kotlin, Scala, Jython (Python Java Implementation), Groovy und viele mehr. [81, 82]

Plattformunabhängigkeit: JavaFX ist, wie für Java natürlich, mit dem Gedanken der Plattformunabhängigkeit entwickelt worden. So werden alle Desktopplattformen, wie Windows, Mac OS X und Linux [83], aber auch mobile und embedded Plattformen unterstützt. [84, 85]

Modernes Design: Durch Nutzung verschiedener Technologien, wie CSS und FXML, können JavaFX Applikationen problemlos auf dem neusten Stand gehalten werden. Oracle liefert im JRE bereits ein modernes Design mit, welches standardmäßig verwendet wird. Dieses wird auch in regelmäßigen Abständen ausgetauscht, um den modernen Anforderungen zu entsprechen.

Es besteht aber auch die Möglichkeit eigene Designs zu spezifizieren. Da dies auf CSS und FXML beruht, kann das Design einfach ersetzt werden, ohne das die Anwendung neu compiliert werden muss. [86, 87]

Support: Oracle bietet eine umfangreiche Dokumentation an [88]. Ebenfalls werden Tutorials von Oracle bereitgestellt um einen Einstieg in die JavaFX Entwicklung zu geben [89]. Auch die Community bietet umfangreiche Unterstützung an, so gibt es von Oracle verwaltete Foren [90], aber auch das OpenJFX Project [91].

Erfahrung: Im Rahmen eines Hochschulprojektes (Mockup++) und mehrerer persönlicher Projekte wurde schon Erfahrung in der Arbeit mit JavaFX gesammelt. Dabei wurde die API nahezu vollständig verwendet.

Sonstiges: JavaFX Applikationen können mithilfe des Scene Builder Werkzeuges schnell und einfach erstellt werden. Ursprünglich wurde das Werkzeug von Oracle bereitgestellt, diese haben es allerdings Open Source zur Weiterentwicklung freigegeben. [92, 93]

JavaFX bietet auch Features zur Interoperabilität mit Swing und SWT an, was ermöglicht alle dort vorhandenen Features zu nutzten. Da JavaFX noch relativ neu ist, hilft dies die noch fehlende Funktionalitäten auszugleichen. [94]

Swing

Swing ist Oracles GUI-Framework der letzten Generation. Es ist vermutlich das meist genutzte Java GUI-Framework.

Sprachunterstützung: Swing ist ein Java Framework, welches auf der Java Plattform läuft. Dies ermöglicht die Nutzung des Frameworks aus verschiedenen Sprachen. Java und Groovy sind einige davon. [95]

Plattformunabhängigkeit: Swing ist auf Desktopplattformen, wie Windows, Linux und OS X verfügbar. Unterstützung für mobile Plattformen, wie Android und iOS, gibt es nicht.

Modernes Design: Sowohl Javas eigenes Look&Feel, als auch native Look&Feels werden unterstützt. Eigene Designs sind eher schwer zu definieren, allerdings gibt es Drittanbieterbibliotheken, welche ihre Designs anbieten. Da Oracle Swing nicht mehr weiterentwickelt, besteht die Gefahr, dass die Look&Feels nicht mehr auf den neuesten Stand gebracht werden. [96]

Support: Mit Swing kommt ein sehr guter Support. Es wurde von Sun Microsystems und Oracle seit 1998 entwickelt und ist seit Java SE 1.2 offiziell im JDK und JRE enthalten [97]. Dadurch hat Swing über die Jahre eine riesige Community aufgebaut, welche neben Oracle umfangreichen Support anbietet. Unteranderem wird Hilfe über Mailinglisten der Swing GUI Toolkit Group bereitgestellt. [98]

Swing wird allerdings nicht mehr weiterentwickelt, sondern von JavaFX abgelöst.

Erfahrung: Im Rahmen der Programmiervorlesungen und kleinerer Projekte wurde Swing eingesetzt. Es sind nahezu alle Features bekannt.

GTK+

GTK+, oder auch das GIMP Toolkit, ist ein GUI-Framework von The GTK+ Team. Es wurde in C geschrieben und unter der GNU LGP Lizenz verfügbar. [99]

Sprachunterstützung: Zur Zeit unterstützt GTK+ 22 Sprachen. Darunter sind unter anderem C++, Java und C#. Allerdings sind nicht alle unterstützten Sprachen auf dem neusten Stand. Lediglich vier der 22 Sprachen haben regelmäßige Updates. [100]

Plattformunabhängigkeit: GTK+ wird offiziell auf Windows, Linux und OS X unterstützt. Es werden aber auch wenige, nicht weit verbreitete Mobile Plattformen unterstützt. [101]

Modernes Design: Es werden sowohl native Look&Feels, als auch sogenannte Themes unterstützt. Dies garantiert immer das neuste und modernste Systemdesign zu nutzen. [101]

Support: GTK+ bietet eine große Bandbreite an Unterstützung an. Gute Dokumentation, eine große Community mit Mailinglisten und IRC Chats und ein Wiki sind verfügbar. [102, 103]

Erfahrung: Mit GTK+ wurden noch keine Erfahrungen gesammelt.

Fazit

GTK+ ist schon wegen der fehlenden Erfahrung nicht geeignet, allerdings auch die begrenzte Plattformunterstützung schließt diese Framework aus. Auch QT hat ein Defizit hinsichtlich der Erfahrung, außerdem erfordert QT für zuverlässiges und offiziell unterstütztes Entwickeln von Anwendungen C++, was die Plattformunabhängigkeit im Vergleich zu Java Anwendungen einschränkt.

JavaFX besitzt zwar noch nicht so viele Features wie Swing, scheint aber zukunftssicherer zu sein. Es bietet vor allem bessere Möglichkeiten Designs zu definieren. Mit dem JavaFX-Framework ist es auch einfacher Applikationen zu entwickeln, da es sich vieler neuerer Technologien und Patterns bedient. Zudem unterstützt es mehr Plattformen als Swing.

JavaFX wird als GUI-Framework eingesetzt. Es gibt verschiedene Sprachen, die JavaFX gleich gut unterstützen. Zwar bieten Scala und Kotlin wahrscheinlich die bessere Syntax, aber da Java die am weitest verbreitete Sprache ist und diese im DLR eingesetzt wird, fällt die Wahl auf Java.

4.4.2 Architektur

Basierend auf den Anforderungen und den ausgewählten Technologien, wurde eine Architektur für die Editoranwendung entworfen. Diese basiert auf dem Model-View-Controller Pattern und wird im Folgenden näher erläutert.

Model:

Die Models repräsentieren die Objekte, welche aus den XML-Nachrichten deserialisiert werden.

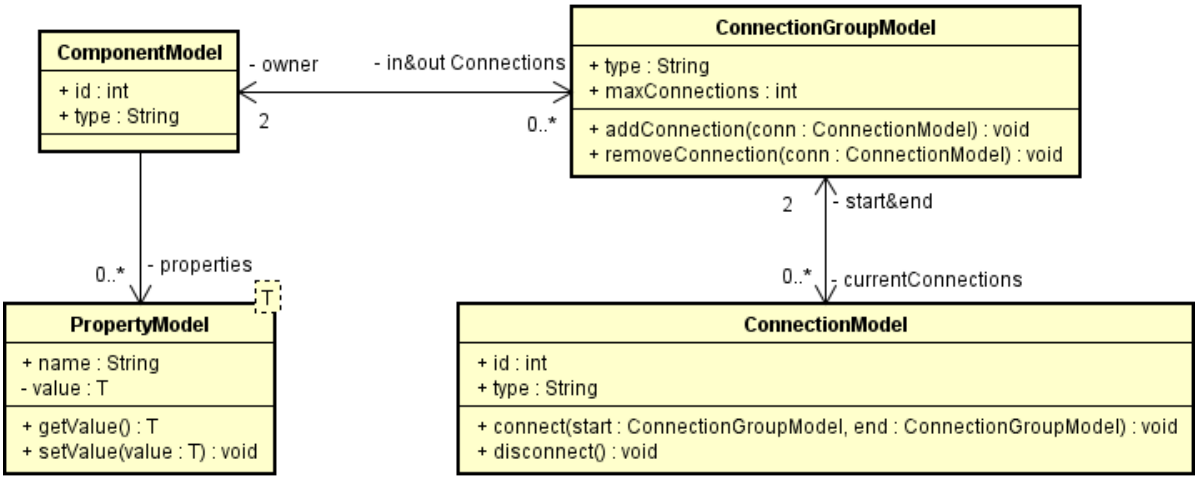

Abbildung 4.16: Die Models der Editoranwendung.

Die Model-Klassen spiegeln die Klassen der Visualisierungsanwendung wider. Es gibt ebenfalls Klassen für Components, Connections und ConnectionGroups (siehe Abbildung 4.16).

Hinzugekommen sind Properties. Diese repräsentieren die Eigenschaften der ViSTA FlowLib Komponenten. In der Visualisierungsanwendung sind sie in den ViSTA Klassen definiert und dort direkt serialisiert, während sie in der Editoranwendung eine eigene Klasse brauchen. Eventuell kann man in der Implementierung von JavaFX Properties [104] Gebrauch machen, da diese die in Abbildung 4.16 definierten Anforderungen schon erfüllen und zusätzlich noch weitere nützliche Funktionen mit sich bringen. So sind sie zum Beispiel beobachtbar und können über Property-Binding mit GUI-Elementen verknüpft werden.

Controller:

Der Controller muss mit der Netzwerkschicht verknüpft werden und Nachrichten serialisieren und deserialisieren.

Der Controller bietet, wie in Abbildung 4.17 dargestellt, Funktionalitäten zum Erstellen von Komponenten und Verbindungen an. Des weiteren wurde eine Project Klasse hinzugefügt, welche auf alle Verbindungen und Komponenten verweist. Sowohl die Project als auch die Controller Klasse haben update Methoden. Diese sollen per Observer-Pattern aufgerufen werden, wenn sich Eigenschaften oder Verbindungen ändern, sodass diese zur Visualisierungsanwendung propagiert werden.

Der Controller ist auch dafür zuständig neue Netzwerkverbindungswünsche an den Connection-Handler weiterzuleiten. Der ConnectionHandler ist in Abbildung 4.15 bereits vorgestellt worden. Nachrichten werden vom MessageParser erzeugt und ausgewertet. Der ConnectionHandler empfängt und verschickt die Nachrichten.

Der MessageParser definiert sowohl Methoden zur Nachrichtenauswertung(updateProjectFromMessage), als auch zur Nachrichtengenerierung (get*Msg) (siehe Abbildung 4.18). Diese Methoden werden dann

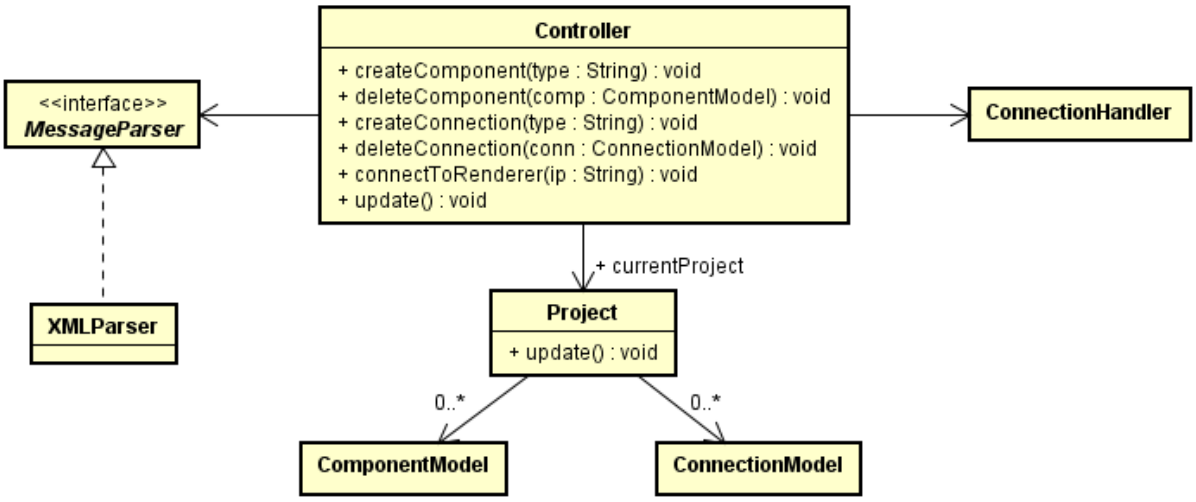

Abbildung 4.17: Der Controller der Editoranwendung.

in einer konkreten Klasse, wie hier der XMLParser, implementiert. Dies erhöht den Abstraktionsgrad und vereinfacht das ersetzten des Nachrichtenformates.

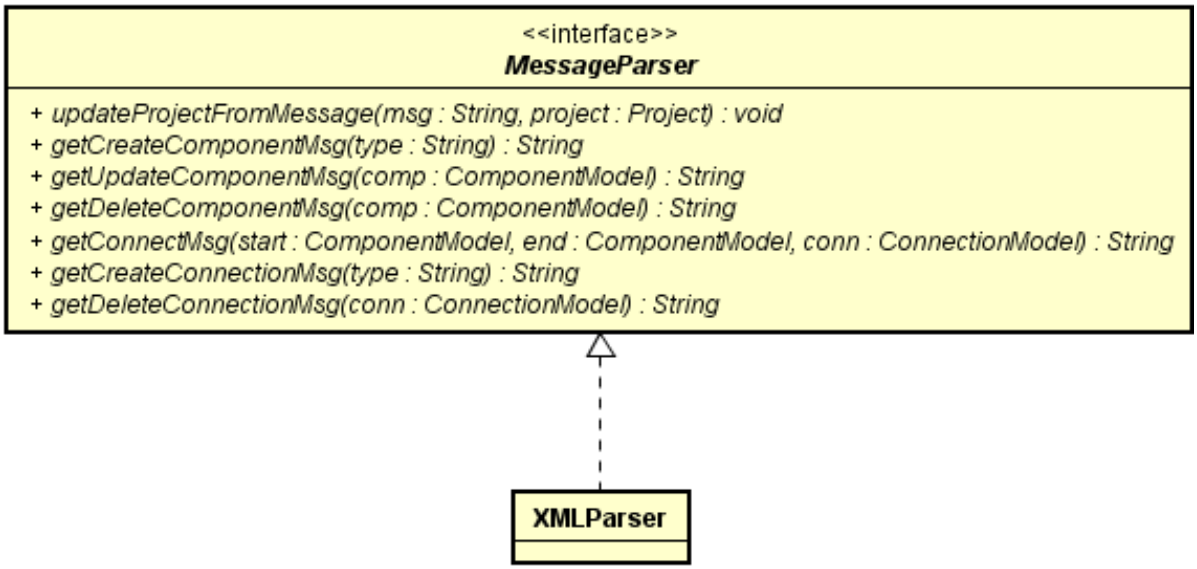

Abbildung 4.18: Der MessageParser der Editoranwendung.

View:

Der View implementiert sowohl grafische Repräsentationen der Models, als auch die Logik zur Interaktion mit dem Controller. Zuerst wurde ein simples Mockup gezeichnet, welches das Layout festlegen soll. Daraus wurden dann Klassen abgeleitet.

Aus Abbildung 4.19 lässt sich ableiten, dass drei Klassen notwendig sind um dieses Layout darzustellen. Diese sind zusammen mit den View-Klassen der Models in Abbildung 4.20 dargestellt.
Die View-Klassen sind in drei Bereiche einzuteilen:

1. Editoren: Dies sind die Klassen, welche in Abbildung 4.19 dargestellt sind.
   - ComponentLibrary: Von hieraus können neue Komponenten erstellt werden. Dies geschieht in Zusammenarbeit mit dem EditorPanel über Drag&Drop.
   - EditorPanel: Hier werden alle Komponenten und Verbindungen angezeigt und bearbeitet.
   - PropertyEditor: Hier können alle Eigenschaften der ausgewählten Komponente verändert werden.
2. Editor-Klassen: Repräsentieren Models im EditorPanel.
   - EditorConnection: Bestehend aus einer Linie zwischen zwei Komponenten.
   - EditorComponent: Stellt eine Komponente im EditorPanel dar.
   - EditorConnectionGroup: Stellt die einzelnen Verbindungsgruppen einer Komponente dar.
3. UI-Klassen:
   - UIComponent: Verwaltet gemeinsame Referenzen für das EditorPanel, die EditorComponents und den PropertyEditor.
   - UIProperty: Eine grafische Darstellung der jeweiligen Komponenteneigenschaft für den PropertyEditor.

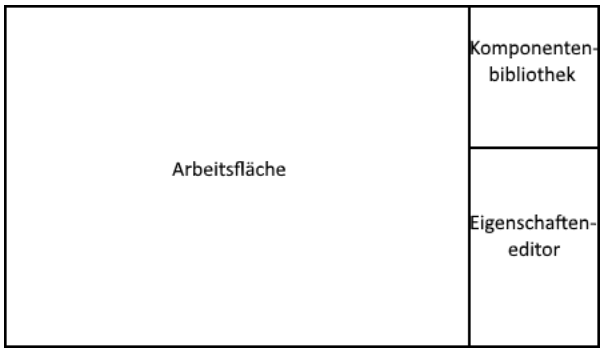

Abbildung 4.19: Das Layout der Editoranwendung.

Die Architektur baut größtenteils auf dem Observer-Pattern auf. Der Controller beobachtet alle Models und diese werden direkt von den View-Klassen aktuell gehalten. Somit muss lediglich das Erstellen und Löschen von Models über aktives aufrufen des Controllers geschehen.

Abbildung 4.21 zeigt die Aufrufe, die zum Erstellen einer Komponente getätigt werden. Die Aufrufsequenz wird durch das Erstellen einer Komponente im EditorPanel vom Benutzer initiiert, diese Anfrage wird vom EditorPanel an den Controller weitergeleitet. Der Controller lässt sich eine Nachricht vom MessageParser generieren und schickt diese über den ConnectionHandler an die Visualisierungsanwendung. Als nächstes wird die erhaltene Nachricht wieder vom MessageParser gelesen und

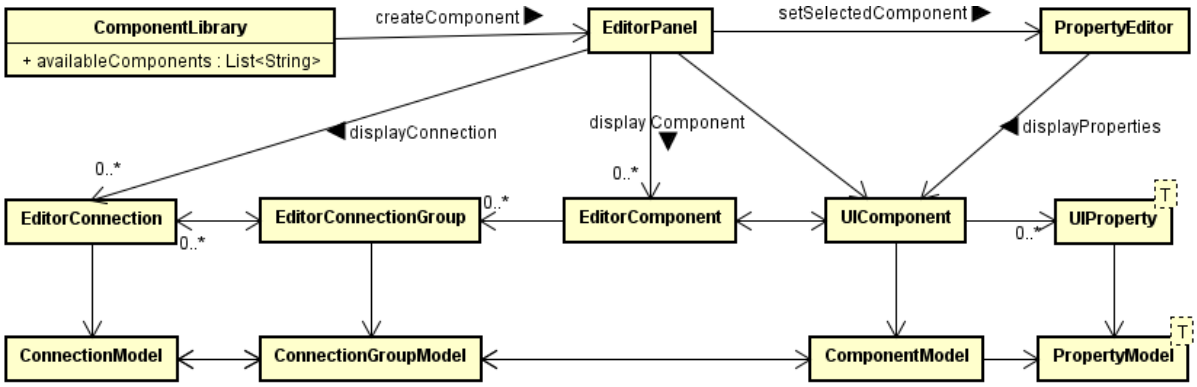

Abbildung 4.20: Die View-Architektur der Editoranwendung.

das Projekt aktualisiert. Zuletzt aktualisiert sich das EditorPanel und zeigt die neu hinzugekommene Komponente dem Benutzer.

Das Sequenzdiagramm in Abbildung 4.22 zeigt die Aufrufe, die nötig sind um eine Eigenschaft zu aktualisieren. Beginnend mit dem Ändern einer Eigenschaft vom Benutzer wird das Model aktualisiert. Danach führen mehrere Aufrufe des Observer-Patterns dazu, dass der Controller eine Notifikation bekommt, sodass dieser eine Update-Nachricht an die Visualisierungsanwendung schickt. Zum Schluss wird die Antwort der Visualisierungsanwendung ausgewertet und überprüft, ob die Eigenschaft auch korrekt aktualisiert wurde.

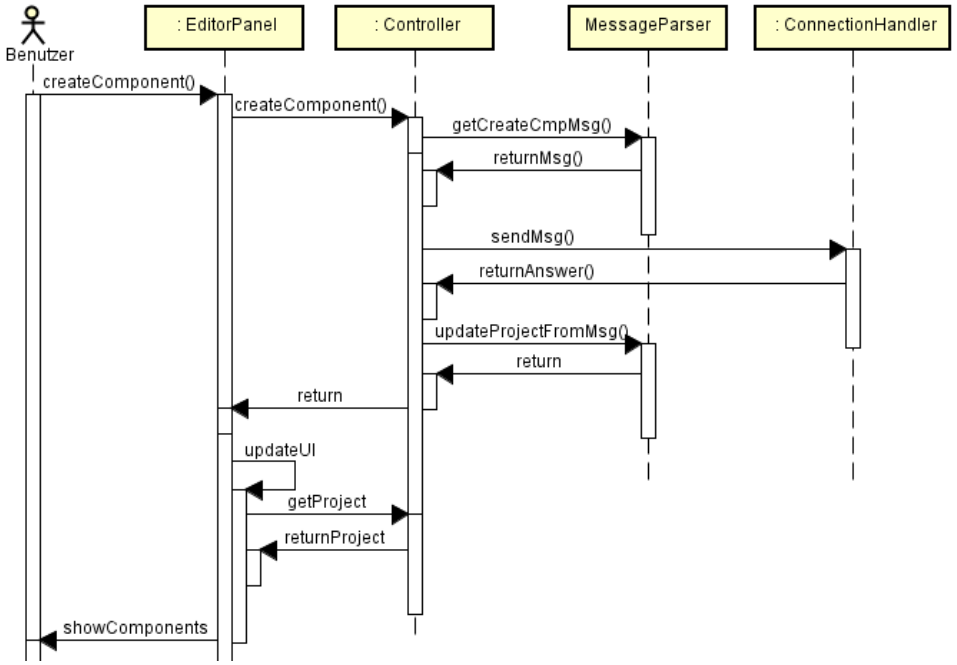

Abbildung 4.21: Sequenzdiagramm, welches das erstellen einer Komponente beschreibt.

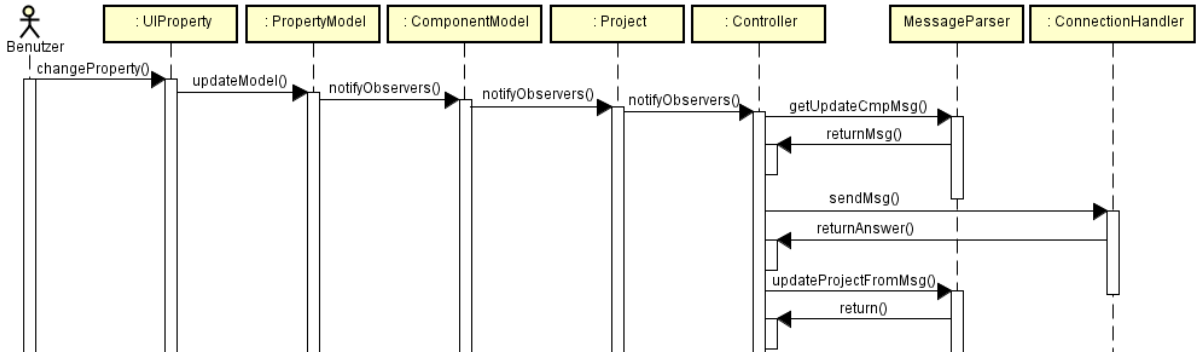

Abbildung 4.22: Sequenzdiagramm, welches das aktualisieren einer Eigenschaft beschreibt.

4.5 Unit Tests

Um die Anforderungen an die Softwarequalität zu erfüllen, muss jeweils ein Test-Framework für Unit Tests gefunden werden. Unter Berücksichtigung der eingesetzten Technologien muss ein C++ Framework für die Visualisierungsanwendung und ein Java Framework für die Editoranwendung gewählt werden. Das DLR nutzt bereits ein C++ und ein Java Framework. Diese sollen auch für dieses Projekt verwendet werden.

4.5.1 Google C++ Testing Framework

Google Test ist ein Plattformunabhängiges C++ Framework zum Schreiben von Unit Tests. Es basiert auf der beliebten xUnit Test-Architektur und ist somit leicht und verständlich zu bedienen [105]. Es ist in vielen Entwicklungsumgebungen, wie CLion[106] und QT Creator[107] unterstützt, wodurch das Erstellen und Ausführen von Tests einfach und übersichtlich gestaltet.

4.5.2 JUnit

JUnit ist ein einfaches Unit Testing Framework für die Java Plattform. Es basiert auf der xUnit [108] Test-Architektur, was es ähnlich zu Googles C++ Testing Framework macht. Es ist das meist verbreitetste Unit Testing Framework für Java, was eine gute IDE-Integration mit sich bringt. So ist JUnit in nahezu allen modernen IDEs, wie Eclipse[109], Netbeans[110] und IntelliJ[111], standardmäßig unterstützt. Dies erlaubt ein einfaches Entwickeln, Ausführen und Auswerten von Tests.

5 Implementierung

In diesem Kapitel wird der Implementierungsprozess beschrieben. Zuerst werden kurz die verwendeten Entwicklungswerkzeuge beschrieben, danach wird die programmiertechnische Implementierung behandelt. Dabei werden Abweichungen von dem Entwurf erläutert. Des Weiteren werden Probleme und deren Lösungen erklärt.

5.1 Allgemein

Der Entwicklungsprozess hat sich an ein agiles Modell angelehnt. Obwohl der Entwurf schon eine umfangreiche Architektur bereitgestellt hatte, mussten noch viele Probleme gelöst werden. Deswegen wurden die Bausteine des Entwurfes inkrementell entwickelt, sodass stets ein Problem nach dem anderen gelöst werden konnte.

Es wurden regelmäßig Gespräche über den Entwicklungsstand mit DLR Mitarbeitern geführt und kleine Demonstrationen der schon funktionierenden Softwarekomponenten gezeigt. Dies hat garantiert, dass die Software den Anforderungen des DLR entspricht.

Entwickelt wurde auf der SUSE Enterprise Plattform.

5.2 Visualisierungsanwendung

5.2.1 Werkzeuge zur Entwicklung

C++ Standard

Es wird mit C++ Standard 2014 entwickelt. Dies erlaubt vor allem die Verwendung von Lambdas und dem auto Keyword. Auf Smart Pointer werden verzichtet, da ViSTA und ViSTA FlowLib diese nicht unterstützen.

CMake

Um ein einheitliches Bauen der Anwendung auf allen Systemen zu garantieren, nutzt das DLR CMake. In CMake werden die Quellcodedateien und Bibliotheksabhängigkeiten definiert und daraus können Makefiles oder Visual Studio Solutions generiert werden. Somit müssen die Projekte nicht für jede Plattform neu aufgesetzt werden. [112]

CLion

CLion ist eine plattformunabhängige C++ Entwicklungsumgebung der Firma JetBrains. Sie eignet sich vor allem zur Entwicklung in diesem Projekt, da sie CMake und Google Test Unterstützung mit bringt. [106, 113]

5.2.2 Entwicklungsverlauf

Da agilen Prinzipien gefolgt wurden, war das erste Ziel eine funktionsfähige Anwendung zu entwickeln. Diese sollte mindestens eine Komponente in den ViSTA Szenegraphen einfügen können. Somit wurde als erstes ein Standard FlowLib-Projekt erstellt. FlowLib-Projekte sind immer ähnlich aufgebaut:

1. ViSTA initialisieren
2. FlowLib initialisieren und mit ViSTA verknüpfen
3. ViSTA-Render-Schleife starten.

Aus zahlreichen Beispielen konnte eine solche Anwendung schnell geschrieben werden. Im nächsten Schritt musste ein System entwickelt werden, was erlaubt nach dem Starten der ViSTA-Render-Schleife Änderungen am Szenegraphen vorzunehmen. Hier trat das erste Problem auf, da keine Dokumentation vorhanden war, die eine solche Funktionalität beschrieb. UVDV hatte dieses Problem nicht, da dort der Szenegraph nur vor dem Starten der Render-Schleife verändert wurde.

Nach einem Gespräch mit einem DLR Mitarbeiter kam heraus, dass eine solche Funktionalität selbst implementiert werden muss.

Hierzu wurde ein VistaObserver an den VistaEventManager übergeben, der einmal pro Schleifendurchlauf aufgerufen wird. Dieser ist in Abbildung 5.1 dargestellt. Er besitzt eine Queue, die auszuführende Funktionen speichert, einen Lock, der ein Threadsicheres einfügen und herausnehmen aus der Queue erlaubt und eine notify Methode, die von ViSTA einmal pro Schleifendurchlauf aufgerufen wird. Nach außen hin ist die runLater Methode definiert, hier können Funktionen an die Queue übergeben werden, die beim nächsten Aufruf ausgeführt werden.

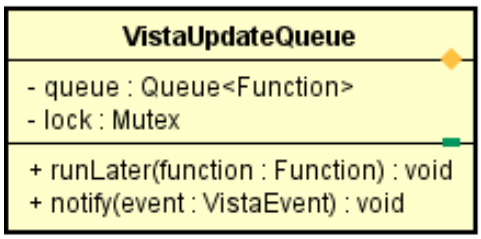

Abbildung 5.1: Die VistaUpdateQueue zum verändern des ViSTA Szenegraphen zur Laufzeit.

Eine einfache Implementation sieht wie folgt aus:

```
void VistaUpdateManager::runLater(std::function<void (void)> function) {
        lock.lock();
        queue.push_back(function);
        lock.unlock();
}
```

```
void VistaUpdateQueue::Notify(VistaEvent *event) {
	lock.lock();
	while (!queue.empty()) {
		queue.front()(); //Execute first function
		queue.pop_front();
	}
	lock.unlock();
}
```

Mit dieser Implementierung ist es nun möglich den Szenegraphen jederzeit zu verändern.

Der Rest der im Entwurf spezifizierten Architektur war schnell und Problemlos entwickelt worden. Die Entwicklung der Visualisierungsschicht, war schon nach 13 der geplanten 20 Personenarbeitstagen beendet.

5.3 Netzwerk Kommunikation

Die Entwicklung der Netzwerkschicht stellte sich Aufgrund der guten Unterstützung von XML, als auch ZeroMQ, sowohl in Java, als auch in C++ mit ViSTAs TinyXML, als sehr schnell und einfach heraus. Diese war nach lediglich fünf der geplanten zehn Personenarbeitstagen beendet. Es konnten problemlos Komponenten über das Netzwerk mit XML Nachrichten erstellt, gelöscht, verknüpft und aktualisiert werden.

Dateipfade: Da VTK-Datensätze mehrere hundert Gigabyte an Größe aufweisen können, werden lediglich Dateipfade ausgetauscht, welche von überall aus dem Netzwerk erreichbar sind. Dies führt vor allem zu Kompatibilitätsproblemen zwischen Linux und Windows, da Dateipfade eine andere Syntax haben, insbesondere Netzlaufwerknamen können nur schwierig aufgelöst werden. Für dieses Problem wurde keine Lösung gefunden, wodurch nur Dateipfade von gleichen Systemen ausgetauscht werden können.

5.4 Editoranwendung

5.4.1 Werkzeuge zur Entwicklung

Java Standard

Zur Entwicklung wurde der neuste Java 8 Standard verwendet. Dieser erlaubt vor allem die Verwendung des JavaFX Frameworks, aber auch Lambdas und Streams sind nützliche Features.

Gradle

Um das Bauen der Java Anwendung einheitlich und einfach zu gestalten wurde das Gradle Build System verwendet. Dies erlaubt Abhängigkeiten zu externen Bibliotheken zu spezifizieren, welche in Online Repositories liegen. Gradle lädt diese automatisch herunter und bindet sie in das Projekt mit ein. [114]

IntelliJ

Als Entwicklungsumgebung wurde IntelliJ verwendet. Es wird von JetBrains entwickelt und bietet einen großen Umfang an Features. Es nutzt die gleiche Benutzeroberfläche wie CLion, weshalb auch ein abwechselndes Arbeiten keine Umgewöhnung benötigt. JUnit und Gradle werden auch standardmäßig unterstützt. [115, 111]

5.4.2 Entwicklungsverlauf

Models und Observer: Die Entwicklung der Editoranwendung begann mit den Models. Hierzu wurde sichergestellt, dass diese aus XML-Nachrichten generiert werden konnten und Aktualisierungen der Komponenten über das Observer-Pattern automatisch Updatenachrichten an die Visualisierungsanwendung schicken.

Für das Observer-Pattern konnte vor allem Gebrauch von JavaFX Properties[104] gemacht werden. Die Definition der PropertyModels sieht wie folgt aus:

```
//in PropertyModel.java
public class PropertyModel<T> extends SimpleObjectProperty<T> {
       public PropertyModel(T initialValue, String name) {
              super(null, name, initialValue);
       }
}
```

Dadurch dass Properties schon Observable sind, muss kein weiteres Pattern implementiert werden. Somit kann die ComponentModel Klasse einfach alle ihre Eigenschaften beobachten und Änderungen zum Controller weiterleiten:

```
//in ComponentModel.java
for (final PropertyModel property : properties.values()) {
        property.addListener((observable, oldValue, newValue) -> {
              setChanged();
              notifyObservers("property change");
       });
}
```

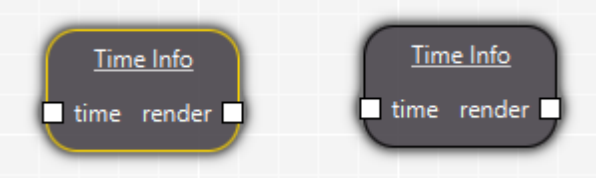

Abbildung 5.2: Das Komponentendesign. Links: Eine ausgewählte Komponente. Rechts: Eine nicht ausgewählte Komponente.

Design: Für die Benutzeroberfläche wurde das von Oracle mitgelieferte Look&Feel genutzt, allerdings mussten neue GUI-Klassen ein eigenes Design bekommen, die die Komponenten darstellen. Dies

ist mit JavaFX simpel. Das Design wurde mit CSS definiert, sodass es jederzeit ausgetauscht werden kann. Wie in den Abbildungen 5.2 und 5.3 dargestellt ist, kann ein modernes und attraktives Design in nur sieben Zeilen Code definiert werden. Es wurde ein gräulicher Hintergrund mit einer schwarzen Kante gewählt. Bei ausgewählten Komponenten ist die Kante goldfarben. Die Ecken sind abgerundet und ein Schatten ist hinter die Komponenten gelegt worden, sodass ein 3D-Effekt zustande kommt.

```
.editor-component-selected {
    -fx-padding: 5 -3 10 -3;

    -fx-background-color: rgba(78, 72, 81, 0.74);
    -fx-background-radius: 15;
    -fx-border-radius: 15;
    -fx-border-color: gold;
    -fx-border-style: solid;

    -fx-effect: dropshadow(gaussian, rgba(0, 0, 0, 0.5), 10, 0.7, 0, 0);
}
```

Abbildung 5.3: Ein Stylesheet, welches das Design für die Komponenten definiert.

PropertyEditor: Für den PropertyEditor wurde auf eine Bibliothek zurückgegriffen, die einen solchen schon implementiert hat. Das PropertySheet von ControlsFX ist ein generischer PropertyEditor, welcher alle Anforderungen an den PropertyEditor erfüllt. Properties müssen lediglich ein Interface implementieren, damit sie im PropertySheet dargestellt werden können. [116] Dadurch war die Implementierung des PropertyEditors trivial.

Komponenten verbinden: Der zeitaufwändigste Teil der Editorschicht war die Implementierung der Verbindungslogik. Es musste sichergestellt werden, dass keine falschen Verbindungen erstellt werden können. Außerdem dürfen Komponenten nicht mehr Verbindungen annehmen als für sie vorgesehen sind. In Abbildung 5.4 ist dargestellt, wie sich Verbindungen verhalten.

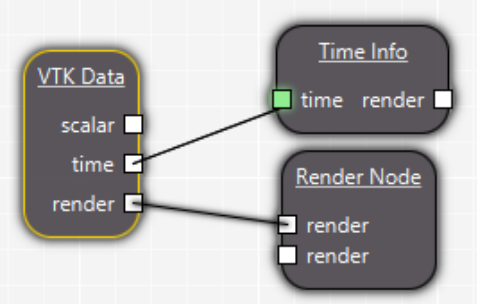

Abbildung 5.4: Verbindungen zwischen Komponenten.

Beim Erstellen von Verbindungen zeigt die Farbe an, ob eine Verbindung möglich ist. Wenn man eine Verbindung mit der Maus zu einem Kästchen zieht, zeigt ein grünes Kästchen, dass hier eine Verbindung entstehen kann, während ein rotes Kästchen anzeigt, dass dies nicht funktioniert. Außerdem

ist in der Abbildung dargestellt, wie die VTK Data Komponente nur eine Verbindung, aber die Render Node Komponente mindestens zwei Verbindungen vom Typ render akzeptiert. Wenn Komponenten mehr als eine Verbindung akzeptieren, generieren sie solange neue Kästchen bis das Maximum erreicht ist. Das heißt, wenn in dieser Abbildung noch eine Verbindung zum Render Node erstellt werden würde, dann würde ein drittes Kästchen generiert werden.

Heartbeats: Während der Entwicklung der Editorschicht ist aufgefallen, dass ZeroMQ nicht die Möglichkeit anbietet zu erkennen, ob die Verbindungen unterbrochen worden sind. Somit musste nachträglich noch ein sogenannter Heartbeat implementiert werden. Dieser versendet in regelmäßigen Abständen Nachrichten zwischen den Anwendungen. Dadurch ist es möglich Verbindungsabbrüche zu erkennen, sobald der Heartbeat nicht mehr empfangen wird. Diese Nachrichten werden auf einem extra Kommunikationskanal versendet, sodass sie nicht von der restlichen Applikationslogik unterbrochen werden können.

Implementationszeit: Die restliche Implementierungszeit von 33 Personenarbeitstagen wurde vollständig genutzt, um die Editorschicht zu entwickeln.

5.5 Tests

Entsprechend der Anforderungen wurden für alle Klassen und Methoden Testfälle geschrieben. Jeder mögliche Logikpfad wurde von mindestens einem Test durchlaufen. Lediglich die Überprüfung der Netzwerkkommunikation stellte sich als schwierig heraus, da hier eine Zeitabhängigkeit für Timeouts vorliegt, es musste eine Latenz simuliert werden, die solche Timeouts auslöst. Dies führte dazu, dass Tests mehrere Minuten liefen, um alle Timeouts zu überprüfen. Bei einer noch komplexeren Anwendung könnte es schwierig werden die Anforderungen an die Tests zu erfüllen.

6 Ergebnisse

In diesem Kapitel werden die Ergebnisse der Implementierung präsentiert. Zuerst werden die Anforderungen auf ihre Erfüllung überprüft, danach wird ein Anwendungsbeispiel vorgeführt.

6.1 Erfüllung der Anforderungen

Hier werden die Anforderungen aus Kapitel 3.5 auf ihre Erfüllung überprüft.

Funktionale Anforderungen

F1: Diese Anforderung wurde vollständig erfüllt.

F2: Diese Anforderung wurde vollständig erfüllt.

F3: Diese Anforderung wurde vollständig erfüllt.

F4: Diese Anforderung wurde vollständig erfüllt.

F5: Diese Anforderung wurde teilweise erfüllt.
Es existiert ein Fehler in ViSTA FlowLib der verhindert, dass Time Info Komponenten nicht mehr gezeichnet werden, selbst wenn sie mit keinem Render Node mehr verbunden sind. Dieser Fehler kann nur von den FlowLib Entwicklern behoben werden.

F6: Diese Anforderung wurde vollständig erfüllt.

F7: Diese Anforderung wurde vollständig erfüllt.

F8: Diese Anforderung wurde teilweise erfüllt.
Dies funktioniert nur, wenn sich mit allen Visualisierungsanwendungen verbunden wurde, bevor eine erste Komponente erstellt wurde. Eine umfangreiche Synchronisationstrategie war zeitlich nicht implementierbar.

Nichtfunktionale Anforderungen

NF1: Diese Anforderung wurde vollständig erfüllt.

NF2: Diese Anforderung wurde vollständig erfüllt.

NF3: Diese Anforderung wurde vollständig erfüllt.

NF4: Diese Anforderung wurde vollständig erfüllt.

Technische Anforderungen

T1: Diese Anforderung wurde vollständig erfüllt.

T2: Diese Anforderung wurde vollständig erfüllt.

T3: Diese Anforderung wurde vollständig erfüllt.

T4: Diese Anforderung wurde vollständig erfüllt.

T5: Diese Anforderung wurde teilweise erfüllt.
Die Implementierungszeit hat nicht ausgereicht, um den Flystick mit der Powerwall kompatibel zu implementieren.

6.2 Anwendungsbeispiel

Für das Anwendungsbeispiel steht wieder der Zyklon zur Verfügung:

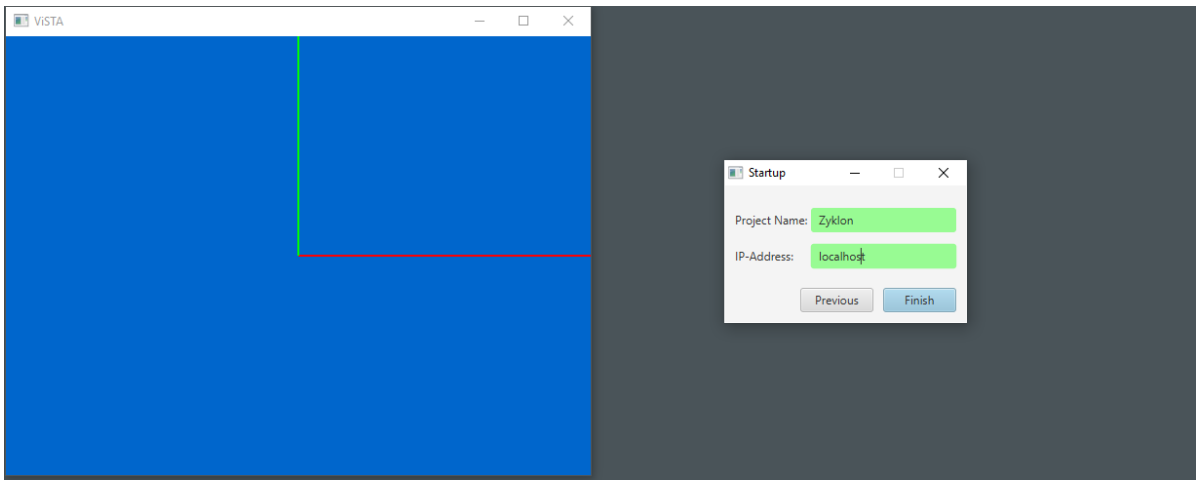

Abbildung 6.1: Die Visualisierungsanwendung (links) und die Editoranwendung (rechts) warten auf die Eingabe der IP-Adresse. Da beide auf dem selben Rechner laufen, kann localhost verwendet werden.

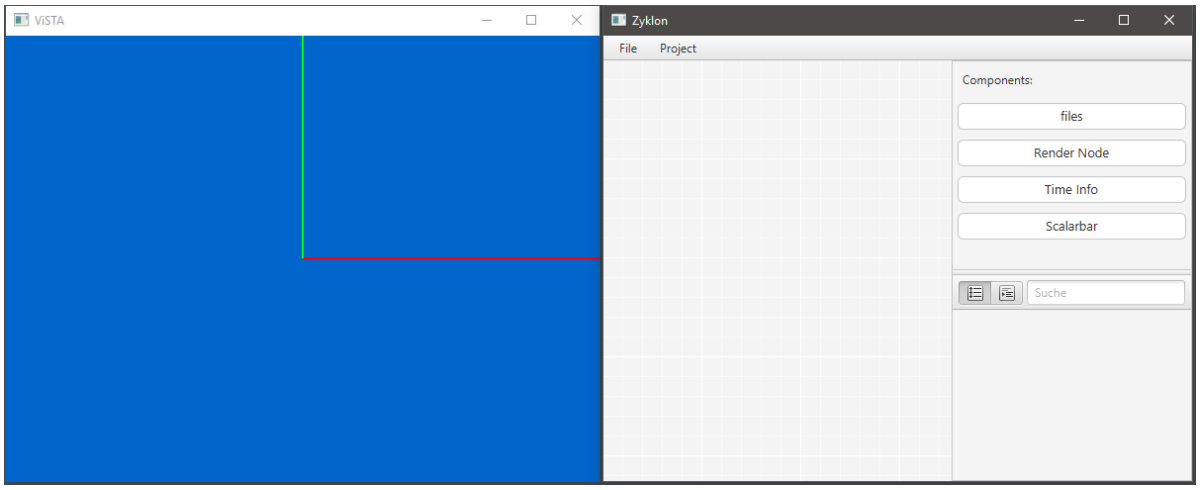

Abbildung 6.2: Nach dem Verbinden ist die Editoranwendung im Bearbeitungsmodus. Es können nun Komponenten aus der oberen rechten Ecke, oder VTK Dateien aus dem Explorer, in die Arbeitsfläche gezogen werden.

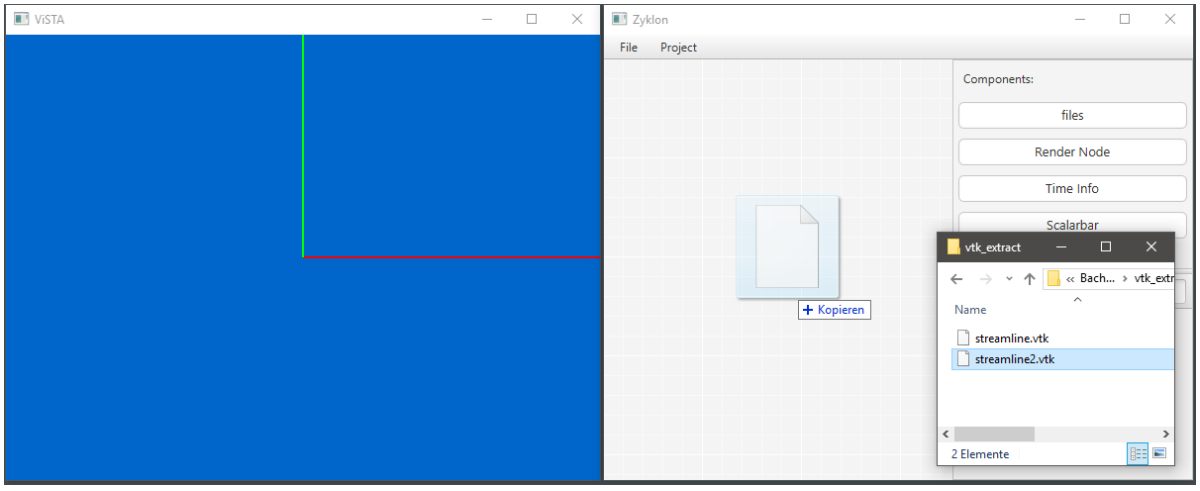

Abbildung 6.3: Hier wird gerade eine VTK Datei in die Arbeitsfläche gezogen.

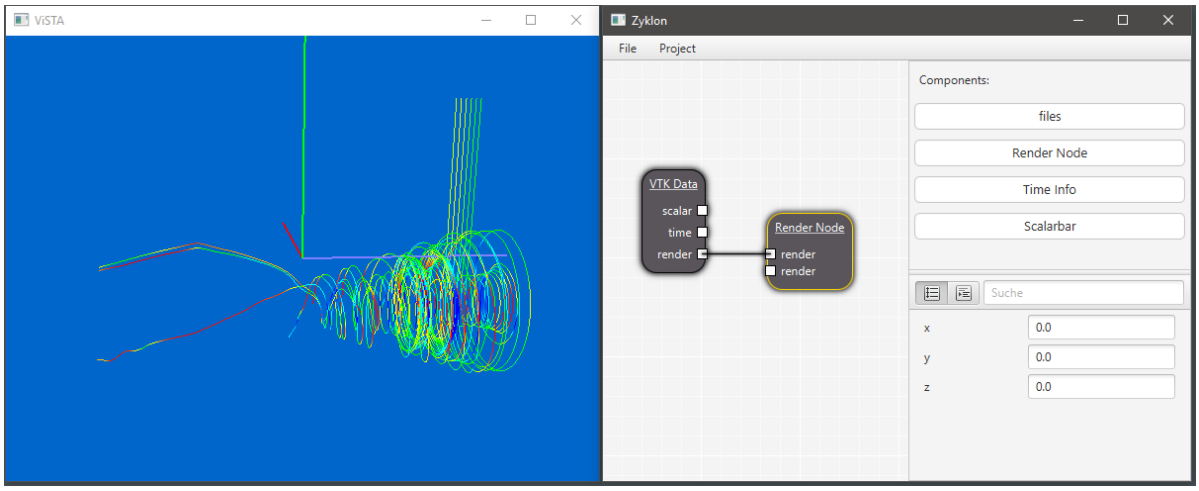

Abbildung 6.4: Nach dem Loslassen der linken Maustaste sind zwei Komponenten in der Editoranwendung erschienen. Eine VTK Data und eine Render Node Komponente. Beide sind über eine Verbindung vom Typ render verknüpft. Dies bewirkt, dass der VTK Datensatz in der Visualisierungsanwendung gerendert wird. Wie zu erkennen ist, handelt sich hier um Stromlinien.

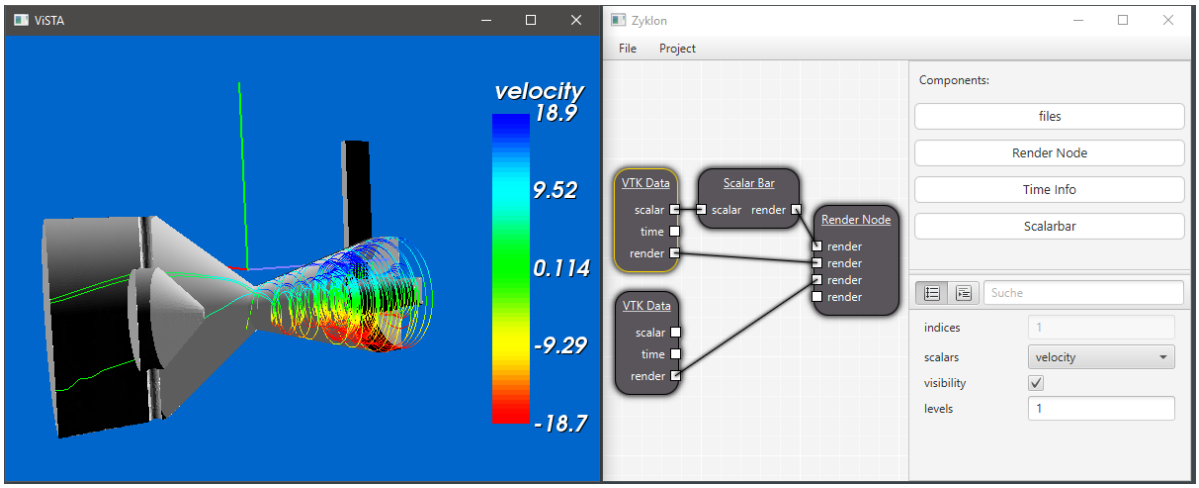

Abbildung 6.5: Nachdem noch ein weiterer VTK Datensatz und eine Scalarbar hinzugefügt und die Verbindungen erstellt wurden, kann man die Funktionsweise des Zyklon erkennen. Im Eigenschafteneditor wurde der velocity Skalar ausgewählt. So ist sichtbar, dass sich die Luft im Zyklon oben mit +18,9 m/s und unten mit -18,7 m/s bewegt.

7 Fazit und Ausblick

7.1 Fazit

Es ist gelungen innerhalb der vorgegebenen Zeit eine Plattform für das DLR zu schaffen, mit der Strömungssimulationen, basierend auf dem VTK-Dateiformat, visualisiert werden können. Sowohl die Editoranwendung als auch die Visualisierungsanwendung sind erweiterbar, sodass jederzeit neue Komponenten und neue Beziehungen hinzugefügt werden können.

Die Editoranwendung ist intuitiv zu benutzen und hat ein anschauliches Design. Der Aufwand, den UVDV mit sich brachte, ist auf ein Minimum begrenzt.

Die fehlende Dokumentation von ViSTA und ViSTA FlowLib hat sehr viel Zeit in Anspruch genommen. Auch wenn sie großartige Frameworks sind, ist es schade, dass sie nicht zu ihrem vollen Potential ausgeschöpft werden können, da viele Funktionen unentdeckt und unerklärt bleiben.

7.2 Ausblick

Um die Anwendung auf den Stand von UVDV zu bringen muss noch viel geschehen. Die erweiterbare Architektur bietet aber einen schnellen und einfachen Weg neue Komponenten zu definieren.

Als nächstes sollten folgende Features angegangen werden:

- Automatische Synchronisation zwischen mehreren Visualisierungsanwendungen
- Implementierung von VTK Filtern, um Datensätze besser auswerten zu können
- Partikel Generierung mithilfe des Flysticks, welche den Pfaden im Vektorfeld folgen
- Speichern und Laden von Projekten

Ansonsten ist es möglich das Projekt in beliebig viele Richtungen weiterzuentwickeln.

A Anhang

A.1 XML-Reply-Nachricht

```
<response>
    <componentTypes>
        <componentType type="files"/>
        <componentType type="Render Node"/>
    </componentTypes>
    <components>
        <component id="0" type="VTK Data">
            <outConnections>
                <connectionGroup connectionType="render" maxConnections="1"/>
            </outConnections>
            <properties>
                <property name="visibility" modifiable="true">
                    <value type="boolean" value="true"/>
                </property>
                <property name="scalars" modifiable="true">
                    <value type="enum" value="Normals"/>
                    <enumElement value="pressure"/>
                    <enumElement value="turb_omega"/>
                </property>
            </properties>
        </component>
        <component id="1" type="Render Node">
            <inConnections>
                <connectionGroup connectionType="render" maxConnections="10"/>
            </inConnections>
            <properties>
                <property name="x" modifiable="true">
                    <value type="float" value="0.000000"/>
                </property>
            </properties>
        </component>
    </components>
    <connections>
        <connection id="0" type="render" start="0" end="1"/>
    </connections>
</response>
```

Literaturverzeichnis

[1] VTK – The Visualization Toolkit. www.vtk.org, abgerufen am 17. 11. 2016.

[2] RWTH Aachen: *ViSTA Virtual Reality Toolkit*, 2015. http://www.itc.rwth-aachen.de/cms/IT-Center/Forschung-Projekte/Virtuelle-Realitaet/Infrastruktur/~fgmo/ViSTA-Virtual-Reality-Toolkit/, abgerufen am 17. 11. 2016.

[3] Deutscher Wetterdienst: *Forschung und Entwicklung für die Wettervorhersage*. http://www.dwd.de/DE/forschung/wettervorhersage/wettervorhersage_node.html, abgerufen am 18. 11. 2016.

[4] Dr. Alexander Sonnenburg: *Strömungssimulation in der Umwelttechnologie – Effiziente Versuchsplanung mit CFD*. HA Hessen Agentur GmbH, Wiesbaden, Deutschland, 1. Aufl., 2009. https://www.hessen-umwelttech.de/mm/cfd_09_screen.pdf, abgerufen am 19. 11. 2016.

[5] Wito Engelke: *Exploration und Analyse von Strömungsdaten mit Hilfe von Tabletop Displays*. Diplomarbeit, Otto von Guericke Universität Magdeburg, 2011.

[6] Robert S. Laramee, Helwig Hauser, Helmut Doleisch, Benjamin Vrolijk, Frits H. Post, Daniel Weiskopf: *The State of the Art in Flow Visualization: Dense and Texture-Based Techniques*. Computer Graphics Forum, 23(2):203–208, 2004.

[7] Daniel Weiskopf: *GPU-Based Interactive Visualization Techniques*. Springer Science & Business Media, Berlin Heidelberg, Deutschland, 1. Aufl., 2006.

[8] The MathWorks: *Using MATLAB to Visualize Scientific Data*. https://www.bu.edu/tech/support/research/training-consulting/online-tutorials/visualization-with-matlab/, abgerufen am 14. 11. 2016.

[9] David Kinnear, Mark Atherton, Michael Collins, Jason Dokhan und Tassos Karayiannis: *Colour in Visualisation for Computational Fluid Dynamics*. Optics & Laser Technology, 38:286—-291, 2006. http://bura.brunel.ac.uk/bitstream/2438/2322/1/Colour%20in%20visualisation%20for%20computational%20fluid%20dynamics.pdf, abgerufen am 19. 11. 2016.

[10] JAXA – Japan Aerospace Exploration Agency: *Research and Development of Vortex Generators for Higher-Performance, Safer Passenger Aircraft*. Aviation Program News, (25):7, 2012. http://www.aero.jaxa.jp/eng/publication/magazine/pdf/en_apgnews25.pdf, abgerufen am 19. 11. 2016.

[11] Ruud Börger: *How to Get the Most out of Arrow Plots in COMSOL Multiphysics*, September 2014. https://www.comsol.com/blogs/get-arrow-plots-comsol-multiphysics/, abgerufen am 24. 11. 2016.

[12] Robert S. Laramee, Gordon Erlebacher, Christoph Garth, Tobias Schafhitzel, Holger Theisel, Xavier Tricoche, Tino Weinkauf, Daniel Weiskopf: *Applications of Texture-Based Flow Visualization*. Engineering Applications of Computational Fluid Mechanics (EACFM), 2(3):270, 9 2008.

[13] Robert S. Laramee, Helwig Hauser, Helmut Doleisch, Benjamin Vrolijk, Frits H. Post: *Feature Extraction and Visualisation of Flow Fields*. Eurographics 2002 State-of-the-Art Reports, S. 10–12, 2002.

[14] Doug McLean – Boing Commercial Airplanes: *Understanding Aerodynamics – Arguing from the Real Physics*. Wiley – John Wiley & Sons Ltd., Hoboken, New Jersey, Vereinigte Staaten, 1. Aufl., 2013.

[15] www.nas.nasa.gov – NASA Advanced Supercomputing Division: *Visualization Techniques in the Virtual Windtunnel*. https://www.nas.nasa.gov/Software/VWT/visualization.html, abgerufen am 24. 11. 2016.

[16] Robert S. Laramee, Helwig Hauser, Helmut Doleisch, Benjamin Vrolijk, Frits H. Post: *The State of the Art in Flow Visualisation: Feature Extraction and Tracking*. Computer Graphics Forum, 22(4):1–4, 2003.

[17] Oxford Dictionary: *Definition of virtual reality in English*, 2016. https://en.oxforddictionaries.com/definition/virtual_reality, abgefufen am 20. 11. 2016.

[18] spektrum.de – Lexikon der Psychologie: *Kinetose*. Spektrum Akademischer Verlag, Heidelberg, 2000. http://www.spektrum.de/lexikon/psychologie/kinetose/7776, abgefufen am 20. 11. 2016.

[19] www.oculus.com – Oculus VR: *Oculus Rift*, 2016. https://www3.oculus.com/en-us/rift/, abgefufen am 20. 11. 2016.

[20] www.vive.com – HTC Corporation, Steam VR: *HTC Vive*, 2016. https://www.vive.com/uk/, abgefufen am 20. 11. 2016.

[21] www.oculus.com – Oculus VR: *Gear VR*, 2016. https://www3.oculus.com/en-us/gear-vr/, abgefufen am 20. 11. 2016.

[22] Google VR: *Google Cardboard*. https://vr.google.com/cardboard/, abgefufen am 20. 11. 2016.

[23] NVIDIA: *NVIDIA 3D Vision Technologie*, 2016. http://www.nvidia.de/object/3d-vision-technology-de.html, abgefufen am 20. 11. 2016.

[24] http://psi.physik.kit.edu/ – Karlsruher Institut für Technologie: *3D durch Polarisationstechnik*, 2013. http://psi.physik.kit.edu/332.php, abgefufen am 20. 11. 2016.

[25] manus-vr.com – Manus VR, 2015. https://manus-vr.com/, abgefufen am 20. 11. 2016.

[26] ar-tracking.com – Advanced Realtime Tracking: *Flystick2*. http://www.ar-tracking.com/products/interaction/flystick2/, abgefufen am 20. 11. 2016.

[27] leapmotion.com – Leap Motion: *Orion Beta*, 2016. https://developer.leapmotion.com/orion, abgefufen am 20. 11. 2016.

[28] www.techviz.net – 3D Visualization Software: *CAVE Virtual Reality System: a visualization solution to create a complete immersive experience*. http://www.techviz.net/cave#.WDUgCOYrK71, abgefufen am 23. 11. 2016.

[29] docs.unrealengine.com – Epic Games: *Basic Scripting – Flow Control*. https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/FlowControl/index.html, abgefufen am 25. 11. 2016.

[30] Prof. Dr. R. Bruns: *Software Engineering 1 – Anforderungsanalyse*. Hochschule Hannover, 2015.

[31] vtk.org – The Visualization Toolkit: *VTK – About*. http://www.vtk.org/overview/, abgefufen am 22. 11. 2016.

[32] vtk.org – The Visualization Toolkit: *VTK – Features: Data Model*. http://www.vtk.org/data-model/, abgefufen am 22. 11. 2016.

[33] vtk.org – The Visualization Toolkit: *VTK – Features: 3D Graphics*. http://www.vtk.org/features-3d-graphics/, abgefufen am 22. 11. 2016.

[34] unity3d.com – Unity: *Understanding Automatic Memory Management*, 2016. https://docs.unity3d.com/Manual/UnderstandingAutomaticMemoryManagement.html, abgefufen am 22. 11. 2016.

[35] John J. Scott: *Native C++ and C# - Which is the Fastest?*, Januar 2015. http://www.codeproject.com/Tips/860631/Native-Cplusplus-and-Csharp-Which-is-the-Fastest-P, abgefufen am 22. 11. 2016.

[36] unity3d.com – Unity: *VR overview*, 2016. https://docs.unity3d.com/Manual/VROverview.html, abgefufen am 22. 11. 2016.

[37] www.mechdyne.com – Mechdyne Enabling Discovery: *getReal3D for Unity: Bring Unity Simulations into Virtual Reality Environments*, 2016. https://www.mechdyne.com/software.aspx?name=getReal3D+for+Unity, abgefufen am 22. 11. 2016.

[38] http://www.middlevr.com/ – .middleVR Improve Reality: *.middleVR for UNITY*. http://www.middlevr.com/middlevr-for-unity/, abgefufen am 22. 11. 2016.

[39] www.vtk.org – The Visualization Toolkit: *VTK/CSharp/ActiViz.NET*, 2012. http://www.vtk.org/Wiki/VTK/CSharp/ActiViz.NET, abgefufen am 24. 11. 2016.

[40] VtkUnity, 2015. https://github.com/ufz-vislab/VtkUnity, abgefufen am 24. 11. 2016.

[41] Matthias Berger, Verina Cristie: *CFD post-prossing in Unity3D*. Procedia Computer Science, 51:6, Dezember 2015.

[42] www.west-racing.com – Chris West. http://www.west-racing.com/mf/?page_id=5892, abgefufen am 22. 11. 2016.

[43] www.khronos.org – Khronos Group, 2016. https://www.khronos.org/vulkan/, abgefufen am 22. 11. 2016.

[44] docs.unrealengine.com – Epic Games: *Programming Guide*. https://docs.unrealengine.com/latest/INT/Programming/index.html, abgefufen am 22. 11. 2016.

[45] www.unrealengine.com – Epic Games: *Virtual Reality*. https://www.unrealengine.com/vr, abgefufen am 22. 11. 2016.

[46] vrcluster.io – Pixela Labs: *VR Cluster — "Virtual Reality for multi screen and clustered solutions"*. http://vrcluster.io/, abgefufen am 22. 11. 2016.

[47] EnSight: *Products*. https://www.ensight.com/which-ensight/, abgefufen am 28. 11. 2016.

[48] EnSight: *Data Formats*. https://www.ensight.com/ensight-data-interfaces/#tab-id-1, abgefufen am 28. 11. 2016.

[49] EnSight: *EnSight VR*. https://www.ensight.com/ensight-vr/, abgefufen am 28. 11. 2016.

[50] www.paraview.org – ParaView: *ParaView/Data formats*, 2016. http://www.paraview.org/Wiki/ParaView/Data_formats#VTK.28Visualization_ToolKit.29_files, abgefufen am 28. 11. 2016.

[51] www.paraview.org – ParaView: *Download*. http://www.paraview.org/download/, abgefufen am 28. 11. 2016.

[52] www.paraview.org – ParaView: *Setting up a ParaView Server*, 2016. http://www.paraview.org/Wiki/Setting_up_a_ParaView_Server, abgefufen am 28. 11. 2016.

[53] www.paraview.org – ParaView: *ParaView/Plugin HowTo*, 2016. http://www.paraview.org/Wiki/ParaView/Plugin_HowTo, abgefufen am 28. 11. 2016.

[54] zeromq.org – ØMQ. http://zeromq.org/, abgefufen am 22. 11. 2016.

[55] zeromq.org – ØMQ: *ØMQ Language Bindings*. http://zeromq.org/bindings:_start, abgefufen am 22. 11. 2016.

[56] Pieter Hintjens: *ZeroMQ: Messaging for Many Applications*. O'Reilly Media, Sebastopol, Kalifornien, Vereinigte Staaten, 1. Aufl., März 2013. http://zguide.zeromq.org/page:all, abgefufen am 22. 11. 2016.

[57] rabbitmq.com – RabbitMQ: *Client & Developer Tools*. https://www.rabbitmq.com/devtools.html, abgefufen am 22. 11. 2016.

[58] rabbitmq.com – RabbitMQ: *Supported Platforms*. https://www.rabbitmq.com/platforms.html, abgefufen am 22. 11. 2016.

[59] rabbitmq.com – RabbitMQ: *Documentation*. https://www.rabbitmq.com/documentation.html, abgefufen am 22. 11. 2016.

[60] rabbitmq.com – RabbitMQ: *RabbitMQ Tutorials*. https://www.rabbitmq.com/getstarted.html, abgefufen am 22. 11. 2016.

[61] nanomsg.org – nanomsg. http://nanomsg.org/index.html, abgefufen am 22. 11. 2016.

[62] nanomsg.org – nanomsg: *Documentation*. http://nanomsg.org/documentation.html, abgefufen am 22. 11. 2016.

[63] nanomsg.org – nanomsg: *Community*. http://nanomsg.org/community.html, abgefufen am 23. 11. 2016.

[64] ecma INTERNATIONAL: *The JSON Data Interchange Format*, 1. Aufl., Oktober 2013. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, abgefufen am 23. 11. 2016.

[65] www.json.org – JavaScript Object Notation: *Introducing JSON*. http://www.json.org/json-en.html, abgefufen am 23. 11. 2016.

[66] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau: *Extensible Markup Language (XML) 1.0*, 5. Aufl., November 2008. https://www.w3.org/TR/2008/REC-xml-20081126/, abgefufen am 23. 11. 2016.

[67] www.grinninglizard.com – Lee Thomason: *TinyXml Documentation*, März 2010. http://www.grinninglizard.com/tinyxmldocs/index.html, abgefufen am 23. 11. 2016.

[68] www.w3schools.com: *XML Tutorial*. http://www.w3schools.com/xml/, abgefufen am 23. 11. 2016.

[69] developers.google.com – Google Developers: *Protocol Buffers – API Reference*, Oktober 2016. https://developers.google.com/protocol-buffers/docs/reference/overview, abgefufen am 23. 11. 2016.

[70] developers.google.com – Google Developers: *Protocol Buffers – Developer Guide*, August 2016. https://developers.google.com/protocol-buffers/docs/overview, abgefufen am 23. 11. 2016.

[71] www.qt.io – The QT Company: *Get Started with Qt*, 2016. https://www.qt.io/download/, abgefufen am 27. 11. 2016.

[72] www.qt.io – The QT Company: *Language Bindings*, 2016. https://wiki.qt.io/Language_Bindings, abgefufen am 27. 11. 2016.

[73] www.qt.io – The QT Company: *Supported Platforms*, 2016. http://doc.qt.io/qt-5/supported-platforms.html, abgefufen am 27. 11. 2016.

[74] www.qt.io – The QT Company: *Elegant UI Design*, 2016. https://www.qt.io/ui/, abgefufen am 27. 11. 2016.

[75] www.qt.io – The QT Company: *Qt Support*, 2016. https://www.qt.io/support/, abgefufen am 27. 11. 2016.

[76] www.qt.io – The QT Company: *Qt Documentation*, 2016. http://doc.qt.io/, abgefufen am 27. 11. 2016.

[77] www.qt.io – The QT Company: *Wiki*, 2016. http://wiki.qt.io/Main, abgefufen am 27. 11. 2016.

[78] www.qt.io – The QT Company: *QT Forum*, 2016. https://forum.qt.io/, abgefufen am 27. 11. 2016.

[79] www.qt.io – The QT Company: *The IDE*, 2016. https://www.qt.io/ide/, abgefufen am 27. 11. 2016.

[80] www.qt.io – The QT Company: *Qt Documentation – All Modules*, 2016. http://doc.qt.io/qt-5/qtmodules.html, abgefufen am 27. 11. 2016.

[81] Josh Juneau: *JavaFX with Alternative Languages*, Juli 2014. http://www.oracle.com/technetwork/articles/java/richclient-2266285.html, abgefufen am 27. 11. 2016.

[82] Michael Thomas: *TornadoFX: JavaFX-Anwendungsframework für Kotlin veröffentlicht*, Januar 2016. https://jaxenter.de/tornadofx-javafx-anwendungsframework-kotlin-33987, abgefufen am 27. 11. 2016.

[83] Oracle: *JavaFX: Getting Started with JavaFX – JavaFX Overview*, 2014. http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#A1095238, abgefufen am 27. 11. 2016.

[84] Gluon: *JavaFXPorts*, 2016. http://gluonhq.com/labs/javafxports/, abgefufen am 27. 11. 2016.

[85] OpenJDK: *Mobile Project*, 2016. http://openjdk.java.net/projects/mobile/, abgefufen am 27. 11. 2016.

[86] Oracle: *JavaFX: Working with JavaFX UI Components – Styling UI Controls with CSS*, 2014. http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/apply-css.htm#CHDGHCDG, abgefufen am 27. 11. 2016.

[87] Oracle: *JavaFX: Mastering FXML – Why Use FXML*, 2014. http://docs.oracle.com/javase/8/javafx/fxml-tutorial/why_use_fxml.htm#JFXMG137, abgefufen am 27. 11. 2016.

[88] Oracle: *JavaFX 8 API*, 2014. http://docs.oracle.com/javase/8/javafx/api/toc.htm, abgefufen am 27. 11. 2016.

[89] Oracle: *Getting Started with JavaFX – About This Tutorial*, 2013. http://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm, abgefufen am 27. 11. 2016.

[90] Oracle: *JAVAFX 2.0 AND LATER*. https://community.oracle.com/community/java/java_desktop/javafx_2.0_and_later, abgefufen am 27. 11. 2016.

[91] OpenJDK Wiki: *OpenJFX*, 2016. https://wiki.openjdk.java.net/display/OpenJFX/Main, abgefufen am 27. 11. 2016.

[92] Oracle: *JavaFX Scene Builder*. http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html, abgefufen am 27. 11. 2016.

[93] Gluon: *Scene Builder*, 2016. http://gluonhq.com/labs/scene-builder/, abgefufen am 27. 11. 2016.

[94] Oracle: *JavaFX: Interoperability*, 2014. https://docs.oracle.com/javase/8/javafx/interoperability-tutorial/index.html, abgefufen am 27. 11. 2016.

[95] Groovy: *Creating Swing UIs*, 2016. http://groovy-lang.org/swing.html, abgefufen am 27. 11. 2016.

[96] Oracle: *Lesson: Modifying the Look and Feel*, 2015. http://docs.oracle.com/javase/tutorial/uiswing/lookandfeel/index.html, abgefufen am 27. 11. 2016.

[97] Java Wiki: *Swing*. http://java.wikia.com/wiki/Swing, abgefufen am 27. 11. 2016.

[98] OpenJDK: *Swing GUI Toolkit Group*, 2016. http://openjdk.java.net/groups/swing/, abgefufen am 27. 11. 2016.

[99] www.gtk.org – The GTK+ Project: *What is GTK+, and how can I use it?*, 2016. https://www.gtk.org/, abgefufen am 27. 11. 2016.

[100] www.gtk.org – The GTK+ Project: *Language Bindings*, 2016. https://www.gtk.org/language-bindings.php, abgefufen am 27. 11. 2016.

[101] www.gtk.org – The GTK+ Project: *Features*, 2016. https://www.gtk.org/features.php, abgefufen am 27. 11. 2016.

[102] www.gtk.org – The GTK+ Project: *Development*, 2016. https://www.gtk.org/development.php, abgefufen am 27. 11. 2016.

[103] www.gtk.org – The GTK+ Project: *Get Assistance*, 2016. https://www.gtk.org/support.php, abgefufen am 27. 11. 2016.

[104] Oracle: *Class ObjectProperty<T>*, 2015. https://docs.oracle.com/javase/8/javafx/api/javafx/beans/property/ObjectProperty.html, abgefufen am 27. 11. 2016.

[105] Google: *Introduction: Why Google C++ Testing Framework?*, 2016. https://github.com/google/googletest/blob/master/googletest/docs/Primer.md, abgefufen am 27. 11. 2016.

[106] JetBrains: *Unit testing*, 2016. https://www.jetbrains.com/clion/features/unit-testing.html, abgefufen am 27. 11. 2016.

[107] The Qt Company: *Running Autotests*, 2016. http://doc.qt.io/qtcreator/creator-autotest.html, abgefufen am 27. 11. 2016.

[108] JUnit: *Frequently Asked Questions*, 2016. http://junit.org/junit4/faq.html#overview_1, abgefufen am 27. 11. 2016.

[109] The Eclipse Foundation: *Writing and running JUnit tests*, 2016. http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2FgettingStarted%2Fqs-junit.htm, abgefufen am 27. 11. 2016.

[110] Oracle: *Writing JUnit Tests in NetBeans IDE*, 2016. https://netbeans.org/kb/docs/java/junit-intro.html, abgefufen am 27. 11. 2016.

[111] JetBrains: *Testing Frameworks*, 2016. https://www.jetbrains.com/help/idea/2016.2/testing-frameworks.html, abgefufen am 27. 11. 2016.

[112] cmake.org – CMake. https://cmake.org/, abgefufen am 27. 11. 2016.

[113] JetBrains: *CMake support*, 2016. https://www.jetbrains.com/clion/features/cmake-support.html, abgefufen am 27. 11. 2016.

[114] Gradle Inc.: *Gradle*, 2016. https://gradle.org/, abgefufen am 27. 11. 2016.

[115] JetBrains: *Gradle*, 2016. https://www.jetbrains.com/help/idea/2016.2/gradle.html, abgefufen am 27. 11. 2016.

[116] ControlsFX: *PropertySheet*, 2016. http://controlsfx.bitbucket.org/org/controlsfx/control/PropertySheet.html, abgefufen am 27. 11. 2016.