Hochschule für Technik und Wirtschaft Berlin
Fachbereich 2
Ingenieurinformatik

Bachelorarbeit

Visualisierung von mit OSGi-Komponenten realisierten Softwarearchitekturen im 3-dimensionalem Raum mit Virtual Reality

Autor: **Marlene Brüggemann**
Neue Hochstraße 24
13347 Berlin
T.: 017680002375
E-Mail: hallo@mlen.de

Matrikelnummer:	546293
Gutachter:	Prof. Dr. -Ing. Jörg Schlingheider
Zweitgutachter:	Dipl.-Math. Andreas Schreiber
Abgabetermin:	05. September 2016
Bearbeitungszeit:	10 Wochen

Inhaltsverzeichnis

Abkürzungsverzeichnis

CSS	Cascading Style Sheets
D3	Data-Driven Documents
DLR	Deutsches Zentrum für Luft und Raumfahrt
GUI	Graphical User Interface
HTC	High Tech Computer Corporation
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
OMG	Object Management Group
OS	Operating System
OSGi	Open Service Gateway initiative
PNG	Portable Network Graphics
RCE	Remote Component Environment
RCP	Rich Client Platform
SDK	Software Development Kit
SVG	Scalable Vector Graphics
UI	User Interface
UML	Unified Modeling Language
VR	Virtual Reality
XML	Extensible Markup Language

1 Einleitung

Um die Architektur von Software, welche den Aufbau und Zusammenhang einzelner Komponenten innerhalb einer Software definiert, einfacher zu verstehen, soll ein Konzept erstellt und umgesetzt werden, welches diese Komponenten und deren Abhängigkeiten visualisiert. Das Hauptaugenmerk ist dabei Software, die auf dem *Open Services Gateway initiative* (kurz: *OSGi*) Konzept, ein dynamisches Komponentensystem für Java, basiert. Die Visualisierung soll einfach zu verstehen sein und im 3D-Raum umgesetzt werden. Mit Hilfe von Virtual-Reality-Brillen (z.B. Google Cardboard oder Oculus Rift) kann und soll die Visualisierung interaktiv betrachtet werden. Wie die Informationen genau dargestellt werden sollen, soll diese Arbeit zeigen.

Da der Aufbau von Software viele Informationen beinhalten kann eignet sich der 3D-Raum ganz gut für dessen Darstellung. Durch interaktives Explorieren können weitere Informationen erscheinen oder verborgen werden, die für den Betrachter im Betrachtungsmoment relevant sind. Vor allem mit Hilfe der relativ neuen VR-Brillen lassen sich neue Konzepte im Gebiet der Software-Visualisierung umsetzen. Anwender für eine solche Visualisierung können Entwickler sein, die mit der visualisierten Software arbeiten und diese besser verstehen wollen. Je nach Komplexität des Projektes können so viel einfacher wichtige Information und Abhängigkeiten gefunden werden als es ohne Visualisierung möglich wäre. Auch für Betrachter ohne technischen Hintergrund kann die Darstellung interessant sein. Sie können sich mit der Visualisierung ein Bild über die Softwarearchitektur verschaffen, zum Beispiel über ihre Größe oder ihre Bestandteile.

Eine der Hauptproblematiken ist ein geeignetes Konzept für folgende Fragen zu finden:

- Wie kann man die einzelnen Komponenten darstellen?
- Wie viele Informationen sollen zu sehen sein ohne dass es zu unübersichtlich wird?
- Wie sieht die Benutzerinteraktion zur Exploration der Softwarearchitektur aus?

Da jede VR-Brille auch andere Möglichkeiten der Interaktivität bietet, wie zum Beispiel einen Zusatzknopf als „Klick“ bis hin zur Möglichkeit, die Position des Betrachters im realen Raum zu erkennen, muss eine geeignete Wahl getroffen werden um sich trotz eventueller Einschränkungen interaktiv verhalten zu können.

Stand der Forschung

Da VR ein relativ neues Thema ist und die ersten Brillen für den Endbenutzer erst dieses Jahr (2016) auf dem Markt kommen, gibt es auf diesem Gebiet noch einige offene Felder. Vor allem das Thema Softwarevisualisierung im 3D-Raum mit VR-Brillen scheint ein noch sehr selten behandeltes Thema zu sein.

Aufbau und Vorgehensweise

Der Aufbau soll aus folgenden Kernpunkten bestehen:

- **Was ist OSGi?** Das Konzept der Softwarearchitektur soll näher untersucht und verstanden werden. Zudem wird ein Projekt gezeigt, was auf dieser Architektur basiert.
- **Grundlagen der Datenvisualisierung sowie Betrachtungen bisheriger Visualisierungsmodelle:** Was sind die Basisoptionen für eine geeignete Darstellung? Welche bisherigen Visualisierungsmodelle lassen sich gut für eine Softwarearchitektur verwenden, welche sind für eine Darstellung im 3D Raum geeignet und welche lassen eine Interaktivität zu? Neben Darstellungskonzepten werden auch Umsetzungsmöglichkeiten gezeigt.
- **Konzeptionelle Entwicklung einer eigenen Visualisierung:** Welche Informationen der Software sind besonders wichtig und welche können evtl. weggelassen werden? Wie kann man diese Informationen am besten unter Berücksichtigung aller wichtigen Themen (3D und VR) darstellen, so dass es auch Sinn ergibt? Es soll ein Konzept entwickelt werden bei dem sich der Benutzer sinnvoll in der Darstellung umsehen kann und den Anforderungen entsprechend die Zusammenhänge verstehen kann.
- **Umsetzung des Konzeptes:** Das Modell soll umgesetzt werden, wobei eine geeignete Auswahl der Technologien auch eine wichtige Rolle spielt. Das Ergebnis soll eine ausführbare Software sein die auf einem Mobiltelefon in Kombination mit einer Google Cardboard VR-Brille lauffähig ist. Auch das Programm zur Extraktion der Datensätze soll umgesetzt werden.

2 OSGi

2.1 Modulare Software

Besonders für komplexere Anwendungen eigenen sich modulare Softwarearchitekturen. Modular bedeutet, dass die Software aus einzelnen Komponenten besteht, die eine für sich alleine stehende funktionale Einheit des Softwareprojektes sind [2]. Auszeichnend für ein modulares Softwaresystem ist das Bestehen aus einer Menge von Modulen die im Zusammenhang zueinander stehen. Ein modulares Softwaresystem ist dann gegeben wenn folgende Kriterien erfüllt werden:

- Das Modul muss definiert sein.
- Die externen Schnittstellen eines Moduls müssen definiert sein.
- Die Abhängigkeiten eines Moduls müssen definiert sein.
- Code und Moduldefinitionen müssen voneinander separiert sein.

Neben der Verringerung der Komplexität des Codes ist das einfache Austauschen einzelner Module ein großer Vorteil. Während der Laufzeit können einzelne Module gestartet, gestoppt, installiert und deinstalliert werden. Die *OSGi Serviceplattfom* basiert auf diesen Prinzipien und ist somit die Grundlage für ein modulares Softwaresystem.

2.2 Was ist OSGi?

Bei der *OSGi Service Platform* handelt es sich um ein modulares System für Java welches von der *OSGi Alliance*, eine Vereinigung aus den Unternehmen *Erricson*, *IBM* und *Oracle* entwickelt wurde [1]. Ursprünglich wurde es für eingebettete Systeme verwendet. Heute wird es für Softwareanwendungen verschiedenster Art verwendet. Für die Umsetzung einer Anwendung in der OSGi-Architektur gibt es verschiedene Implementierungen, einer der bekanntesten ist *Equinox*. Sie stellt die Grundlage aller *Eclipse*-Anwendungen dar. Weitere Implementierungen sind *mBedded Server Professional*, die sich besonders durch wenig Speicherbedarf und Performance auszeichnet was sie besonders attraktiv für Anwendungen macht, die auf kleinere Ressourcen zurückgreifen müssen. Weitere sind *Knopflerfish* und *Apache Felix*.

2.3 Der Aufbau

Basierend auf dem *OSGi-Framework* stellt die OSGi Service Plattform Funktionen für die Erstellung von *Module* und *Services* bereit, welche in Kapitel 2.3.1 und 2.3.2 genauer beschrieben werden. „Framework" bedeutet in diesem Fall ein Programmiergerüst, welche die Struktur des auf ihm aufbauenden Programms vorgibt. Das OSGi Framework besteht aus mehreren Schichten. Die Hauptbestandteile bestehen aus den Modulen, auch Bundles genannt, deren Zuständen und die Services, die miteinander arbeiten. Abbildung 2.1 zeigt die drei Schichten die Teil des Frameworks sind sowie die *Execution Enviroment* und die Schicht die das Operating System darstellt.

Die Execution Enviroment ermöglicht die Unabhängigkeit vom OSGi-Framework zur *Java Runtime Environment* (JRE) Version. Mit der JRE lassen sich Programme ausführen, die in Java geschrieben wurden und definieren, welche Klassen, Interfaces und Methoden in der zugrunde liegende Plattform vorhanden sein müssen [1]. Die Execution Environment ist kein Teil des OSGi-Frameworks, aber es benötigt diese, um lauffähig zu sein. Sie wiederum basiert auf dem Operating System, die unterste Ebene im Schichtenmodell. Die Schichten innerhalb des OSGi Frameworks werden im folgenden erläutert. Da die Security-Schicht in der Visualisierung nicht einbezogen wird, wird diese nicht weiter erläutert.

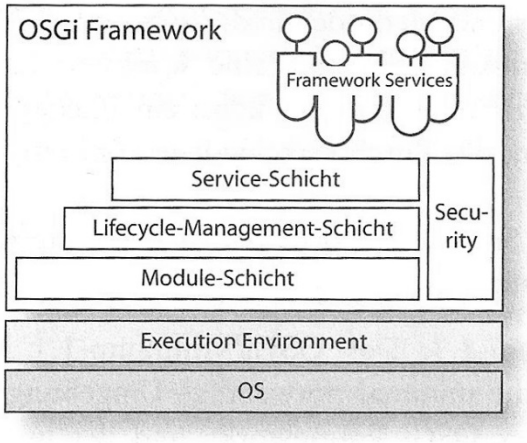

Abbildung 2.1: Das OSGi Schichtenmodell [1].

2.3.1 Bundles

Bundles, welche auch unter dem Begriff Module bekannt sind, bilden die unterste Schicht des Aufbaus. Ein Bundle ist dabei ein Element bestehend aus mehreren Unterelementen, die Packages, welche wiederum aus mehreren Klassen bestehen. Jedes einzelne Bundle kann für sich gestartet, gestoppt, installiert oder deinstalliert werden (während der Laufzeit). Die Abhängigkeiten der Bundles untereinander

und wie mit Services umgegangen wird ist fest definiert. Wenn kein Export definiert ist, können die Klassen nicht von anderen Bundles genutzt werden.

Technisch gesehen ist ein Bundle eine .jar Datei mit einer angehängten *Manifest Datei*, welche bundle-spezifische Informationen enthält. Jedes Bundle besitzt eine *Activator*-Klasse in der sich die *Start*- und *Stop*-Funktion befindet, die man mit beliebigen Operationen füllen kann. Diese Klasse muss im Bundle-Manifest definiert sein um den Aufruf des Activators zu gewährleisten. Abbildung 2.2 zeigt eine solche Manifest-Datei. Die dort definierten Parameter sind unter anderem Name, der symbolischen Name, alle importierten und exportierenden Packages und die Versionsnummer des Bundles.

```
1  Manifest-Version: 1.0
2  Bundle-RequiredExecutionEnvironment: JavaSE-1.7
3  Bundle-ManifestVersion: 2
4  Bundle-Name: RCE Cluster Component GUI
5  Bundle-SymbolicName: de.rcenvironment.components.cluster.gui;singleton:=true
6  Bundle-Version: 7.1.2.qualifier
7  Bundle-Vendor: DLR
8  Bundle-ClassPath: .
9  Import-Package: de.rcenvironment.components.cluster.common,
10  de.rcenvironment.core.component.api,
11  de.rcenvironment.core.component.executor,
12  de.rcenvironment.core.component.model.endpoint.api,
13  de.rcenvironment.core.component.model.spi,
14  de.rcenvironment.core.component.workflow.model.api,
15  de.rcenvironment.core.datamodel.api,
16  de.rcenvironment.core.gui.datamanagement.browser.spi,
17  de.rcenvironment.core.gui.utils.incubator,
18  de.rcenvironment.core.gui.workflow.editor.properties,
19  de.rcenvironment.core.gui.workflow.editor.validator,
20  de.rcenvironment.core.gui.workflow.executor.properties,
21  de.rcenvironment.core.utils.cluster,
22  de.rcenvironment.core.utils.common,
23  de.rcenvironment.core.utils.common.channel.legacy,
24  org.apache.commons.logging;version="1.1.1",
25  org.eclipse.osgi.util
26 Require-Bundle: org.eclipse.ui.views.properties.tabbed
27 Bundle-Localization: plugin
28 RCE-Component: true
```

Abbildung 2.2: Manifest-Datei eines OSGi-Bundles.

Die zweite Schicht ist der *LifeCycle* der Bundles, welche aus mehreren Zuständen besteht. Ein Bundle befindet sich immer in einem bestimmten Zustand welche sich während der Laufzeit verändern kann. Abbildung 2.3 zeigt den Ablauf der Zustände eines Bundles.

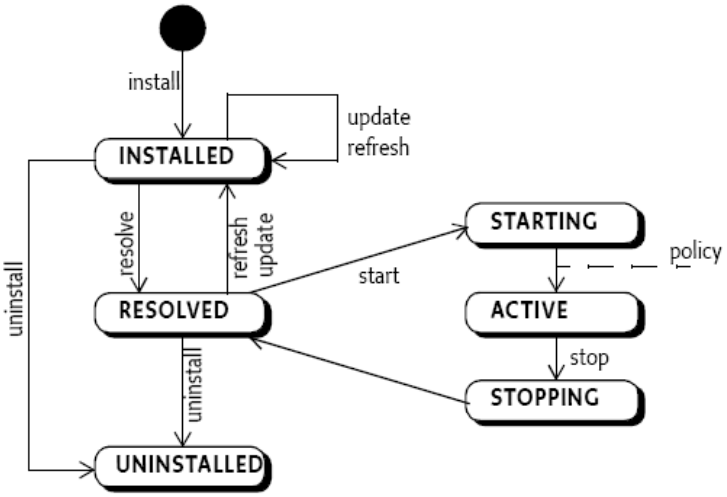

Abbildung 2.3: Die Zustände eines Bundles [112].

Der erste Zustand eines Bundles ist INSTALLED der direkt nach der Installation erreicht wird. Wurde die Installation erfolgreich abgeschlossen befindet sich das Bundle im Zustand RESOLVED. Ist dieser Zustand erreicht, werden alle nötigen Klassen geladen und das Bundle kann gestartet werden. Der Zustand STARTING kann nur eintreffen, wenn der Zustand RESOLVED vorrangig ist. Generell ist dieser Zustand solange aktiv, wie die Start()-Funktion noch nicht beendet ist. Nach erfolgreicher Beendigung der Start()-Funktion wechselt der Zustand auf ACTIVE. Wird das Bundle gestoppt wechselt der Zustand auf STOPPING und wird erst dann in dem Zustand RESOLVED treten, wenn die Stop()-Methode erfolgreich beendet wurde. Von dort aus, wie auch vom Zustand INSTALLED, kann das Bundle wieder aus dem Framework deinstalliert werden, welches dann in den zustand UNINSTALLED wechselt.

Eine spezifizierte Form des Bundles ist das *Fragment-Bundle*. Ein Fragment Bundle erweitert den Klassenpfad, der den Speicherort der vom Benutzer definierten Klassen anzeigt, eines anderen Bundles (*Host-Bundle*) und ermöglicht somit das Hinzufügen anderer Klassen und Ressourcen. So ist es möglich die Funktionen eines Bundles zu erweitern ohne Veränderungen in diesem vorzunehmen [1]. Eine weitere Besonderheit im Vergleich zum Standard-Bundle ist, dass es keine eigenen Zustände besitzt, sondern die des Host-Bundles übernimmt. Die Zugehörigkeit wird in der Manifest-Datei des Fragment-Bundles definiert. Dadurch benötigt es auch keinen Activator.

2.3.2 Services

OSGi-Services sind Dienste, die implementiert und zur Verfügung gestellt werden, auf die externe Bundles zugreifen können. Sie sind sinnvoll um die Komplexität durch Entkopplung von Bundles zu verringern, die Aufgrund der vielen Abhängigkeiten gegeben ist [100]. Bundles stellen Klassen zur Verfügung welche von anderen Bundles genutzt werden können. Services sind Instanzen von Klassen.

Ein OSGi-Service ist ein Java-Objekt in einem Bundle, welches in einer Registratur, der *Service Registry*, angemeldet wird. Sie bilden die oberste Ebene des OSGi Schichten-Modells [101]. Nach der Anmeldung in der Registry ist es anderen Bundles möglich, diese abzufragen und dann zu verwenden. Werden sie nicht mehr genutzt, müssen sie freigegeben und deregistriert werden. Benötigt das Service eine bundle-spezifische Konfiguration, so kann man dies mit einer *Service Factory* lösen, mit der es möglich ist, das Bundle zu übergeben. Da Services während der Laufzeit registriert, verwendet und deregistriert werden können, ist es sinnvoll diese Zustände zu überwachen. Das kann mittels eines *Service Trackers* geschehen. Services haben den gleichen Lebenszyklus wie das bereitstellende Bundle.

2.3.3 Abhängigkeiten

Besonders wichtig sind die Abhängigkeiten der einzelnen Komponenten. Bundles stehen erstmal für sich allein, das heißt, die Packages eines Bundles sind nicht sichtbar für andere Bundles außer dem, in dem sich das Package befindet. Erst wenn das Package, das von einem anderen Bundle benutzt werden soll, exportiert wird und in dem entsprechendem Bundle, welches dieses Package nutzen möchte, importiert wird, kann auch das Package vom importierenden Bundle aus genutzt werden. Somit liegt eine *Package-Anhängigkeit* vor. Die zugehörige Manifest-Datei eines jeden Bundles beinhaltet diese Informationen. Weitere Einstellungen dieser Datei diesbezüglich sind möglich, wie das Exkludieren und Inkludieren einzelner Klassen. Alle Packages, die von einem Bundle importiert werden, werden während der Installation des zu importierenden Bundles zur Verfügung gestellt. Sie müssen vorhanden sein um den Zustand des Bundles von INSTALLED auf STARTED setzen zu können. Wenn die Abhängigkeit eines oder mehrerer Bundles optional ist, lässt sich dies mit einem zusätzlichen Eintrag in der Manifest-Datei definieren. Zudem gibt es noch die Option eines dynamischen Imports, welche nicht das Package während der Installation lädt, sondern erst wenn es gestartet wurde und benötigt wird.

Da jedes Bundle bzw. Package eine Versionsnummer trägt, lassen sie sich auch in den Abhängigkeiten definieren. Das macht vor allem Sinn wenn das gleiche Bund-

le in mehreren Versionen verfügbar ist. Neben einer einzelnen Version lassen sich in den Imports auch Versionsbereiche definieren. Sollte dieser Bereich mehrere Versionen eines Bundles auflisten die importiert werden, wird das Bundle mit der höchsten Versionsnummer gewählt.

Neben der weiter oben beschriebenen *Package-Abhängigkeit* gibt es auch noch die *Service-Abhängigkeit.* Hier sind Bundles nicht voneinander abhängig sind weil sie Packages anderer Bundles nutzen, sondern weil sie einen Service des importierten Bundles nutzen. In beiden Fällen ist ein Import des zur Verfügung stellenden Bundles nötig. Der Vorteil durch den Einsatz von Services liegt in der Flexibilität. Auch wenn das Softwareprojekt anfangs ohne Services strukturierter erscheint, ist der Einsatz von Services sinnvoll, da nicht nur Bundles direkt referenziert werden. So vereinfacht es das „Weglassen" eines Bundles. Abbildung 2.4 zeigt die Package-Abhängigkeit, also die Nutzung der Klassen anderer Bundles sowie die Service-Abhängigkeit.

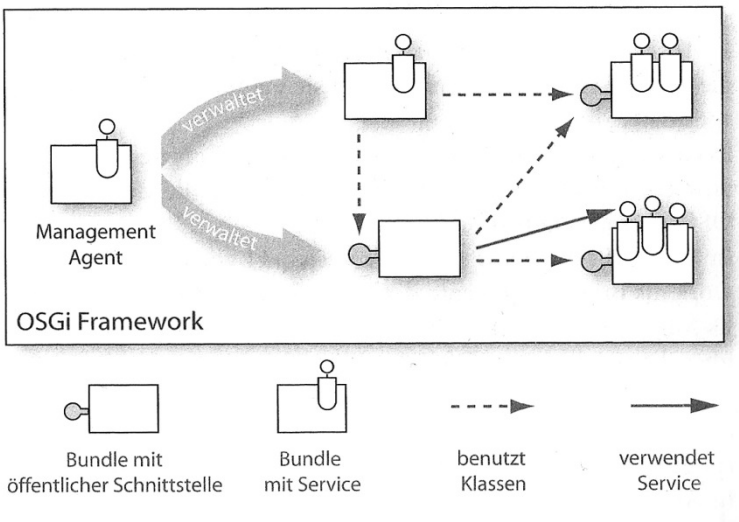

Abbildung 2.4: Bundles und dessen Abhängigkeiten [1].

2.4 Beispiel

Im folgenden Beispiel wird gezeigt wie eine OSGi basierte Anwendung in der Implementierung aussehen kann. Die Software *Remote Component Environment* (RCE) ist eine vom DLR entwickelte Open-Source-Software die es Wissenschaftlern und Ingenieuren ermöglicht, komplexe Systeme und Abläufe darzustellen und zu analysieren. So können verschiedene Tools, die in unterschiedlichen Programmiersprachen geschrieben sind, zusammengeführt werden. Die Zusammenstellung einzelner Tools, die den *Workflow* ergeben, (wie zum Beispiel ein Simulationsprogramm,) werden mit der grafischen Oberfläche verbunden und dessen Ausführung

definiert. Vor allem bei komplexeren Projekten, an dem Ingenieure aus verschiedensten Disziplinen arbeiten, bietet diese Software einen großen Vorteil [102]. Abbildung 2.5 zeigt die Oberfläche des Programmes mit einem Workflow.

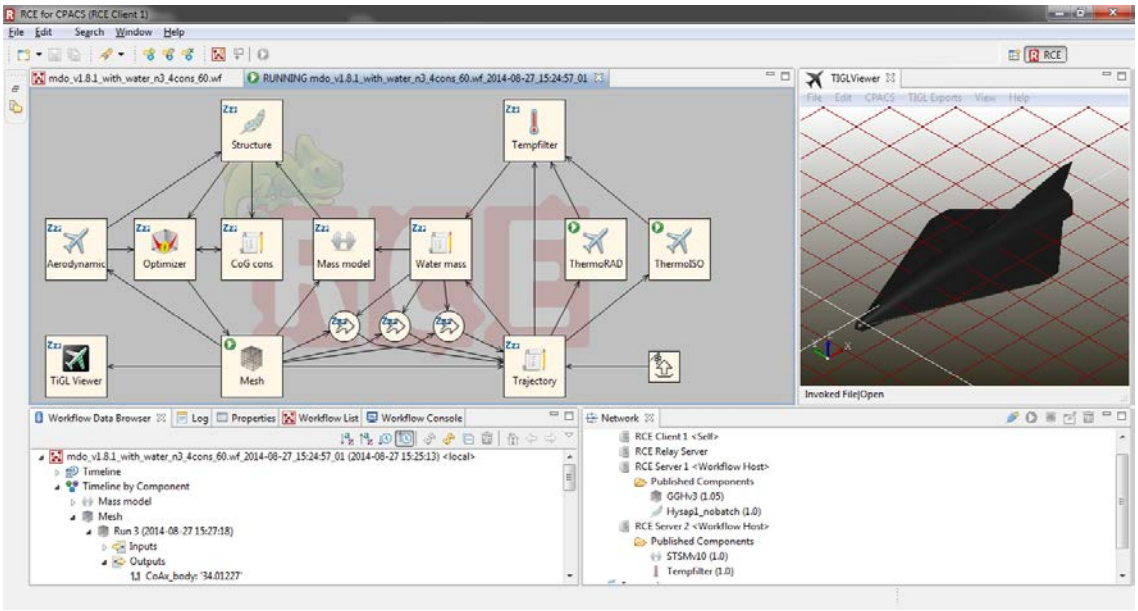

Abbildung 2.5: Die Oberfläche der RCE Software [102].

Die Grundlage der in Java umgesetzten Software ist die *Rich Client Plattform* (RCP) [14]. Sie ist die Basis der *Eclipse* IDE, kann aber auch für sich allein genutzt werden. Sie bietet Grundstrukturen für den Aufbau von GUI-Anwendungen und ist auf Modularität ausgelegt. Die Basis aller auf RCP aufbauenden Softwareprojekte ist OSGi.

3 Datenvisualisierung

3.1 Einführung in die Datenvisualisierung

Unter Datenvisualisierung versteht man die Repräsentation von Daten auf grafischer Ebene. Ziel ist es, komplexe Daten so darzustellen, dass sie für den Betrachter einfacher zu verstehen sind. Datenvisualisierung ist ein sehr weites Feld mit vielen weiteren Unterarten wie zum Beispiel wissenschaftliche Visualisierung (computergestützte Darstellung von wissenschaftlichen Daten), Informationsvisualisierung und Softwarevisualisierung. Beispiele für Datenvisualisierungen sind unter anderem Landkarten, Wetterkarten und Diagramme jeglicher Art.

Jedes grafische Symbol hat eine bestimmte Semiologie. Daher ist es besonders wichtig, dass die grafischen Elemente so dargestellt werden, dass ihre Bedeutung eindeutig interpretiert werden kann. Dabei gibt es zwei Kernpunkte zu beachten. Einmal die Darstellung der einzelnen Objekte an sich. Zum Beispiel wird ein Pfeil als Richtungsweiser interpretiert, ohne dieses vorher definieren zu müssen. Und des Weiteren die Darstellung der Objekte im Zusammenhang. Zum Beispiel sagt ein Balken in einem Balkendiagramm allein nicht sehr viel aus, doch wenn mehrere Balken zu sehen sind, lässt sich eine Aussage durch die Verhältnisse untereinander treffen [7].

Die Darstellung von grafischen Elementen kann man mit visuellen Variablen versehen werden, die das Verständnis vereinfachen sollen, welche im Folgenden definiert werden:

- **Position**: Die Position im zwei- oder dreidimensionalen Raum ist ein besonders auffälliges Attribut, mit der sich mehrere Informationen auf einmal darstellen lassen. So lassen sich in einem dreidimensionalen Raum gleich drei Variablen darstellen.
- **Form**: Das geometrische Erscheinungsbild ist besonders wichtig im Zusammenhang mehrerer Symbole. So lassen sich verschiedene Objekte auch unterschiedlich darstellen.
- **Größe**: Die Größe eines Objekts kann Informationen über die Wichtigkeit oder eben die Größe der Komponente geben. Vor allem im Vergleich sind verschieden große Objekte schnell zu analysieren.
- **Helligkeit**: Auch die Helligkeit kann Aufschluss über gewisse Informationen geben, wie beispielsweise unterschiedliche Prioritäten.

- **Farbe**: mit Farbe lassen sich viele Informationen darstellen, ein Beispiel hierfür sind Temperaturdarstellungen auf Wetterkarten, die sich durch ihre Farbkennzeichnung sehr schnell erfassen lassen.
- **Orientierung**: Vor allem mit länglichen Formen lassen sich Richtungen darstellen.
- **Textur**: die Textur eines Elementes kann viele Informationen beherbergen. Es lassen sich hiermit sehr viele verschiedene Objekte der gleichen Form darstellen, die trotzdem eine unterschiedliche Aussage besitzen können.

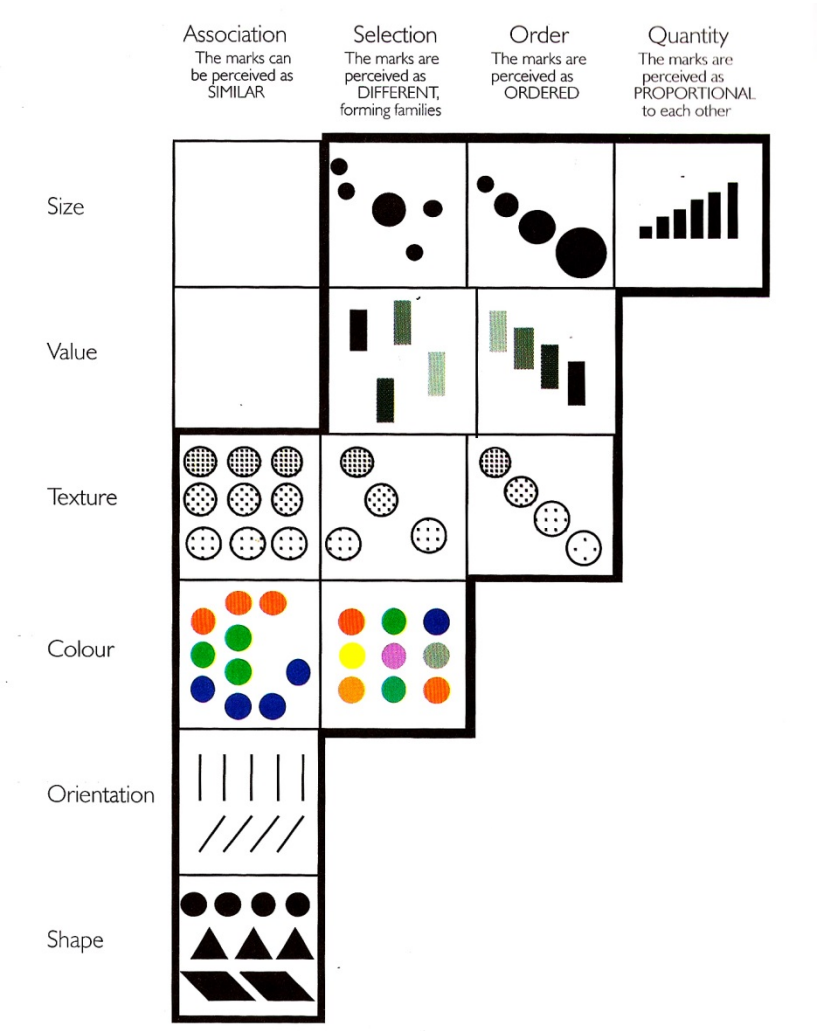

Abbildung 3.1: Visuelle Variablen [7].

Je nach Art der vorhandenen Daten lassen sich Datenvisualisierungen in verschiedenen Techniken umsetzen. Abbildung 3.1 zeigt die visuellen Variablen zusätzlich in vier fundamentalen Darstellungsarten. Die erste Spalte gibt Auskunft darüber, wie einfach es ist die visuelle Variable im einzelnem wahrzunehmen. Spalte 2 zeigt die Wahrnehmung durch Hervorhebung eines Elements. In Spalte 3 wird dargestellt, wie die visuelle Variable in einer Ordnung erfasst wird und Spalte 4 stellt dar wie Elemente in Abhängigkeit untereinander wahrgenommen werden.

Durch den Einsatz und der Kombination verschiedener solcher Variablen lassen sich einige Grundkonzepte umsetzen.

3.1.1 Datenvisualisierung durch Netze

Eine Netzdarstellung ist der Zusammenhang aus zwei unverzichtbaren Komponenten: Die Graphen und die Knotenpunkte (auch *Nodes* genannt). Daten lassen sich als Knotenpunkte darstellen und deren Abhängigkeit zu anderen Knotenpunkten durch Graphen (Abbildung 3.2). Dabei kann es sich um eine zweiseitige Abhängigkeit handeln aber auch um eine richtungsbezogene Abhängigkeit. Zum einem lassen sich mit Graphen und Knotenpunkten komplexe Systeme darstellen, wie etwa Moleküle oder Transportnetzwerke, zum anderen Abhängigkeiten. Beispiele hierfür sind die Visualisierung von sozialen Netzwerken oder von Datenbanksystemen [3].

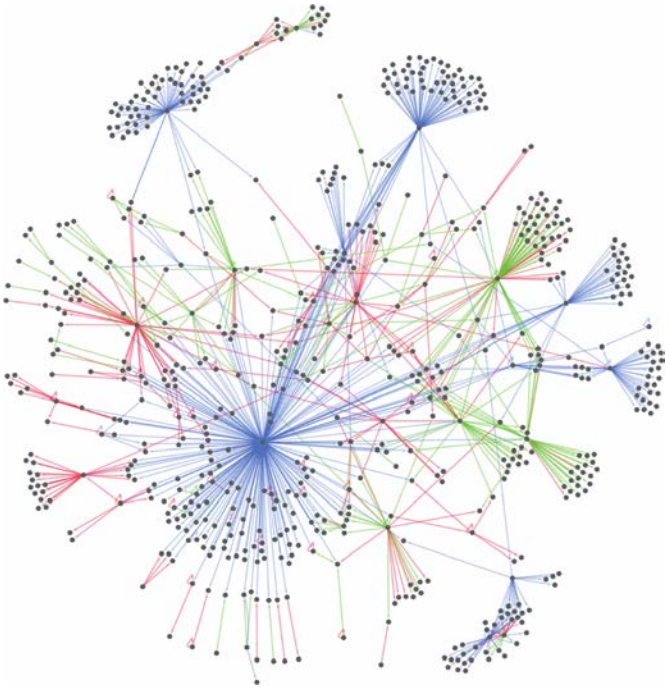

Abbildung 3.2: Netzdarstellung von Daten [20].

3.1.2 Visualisierung von hierarchischen Daten

Die meisten Darstellungsarten eignen sich eher für die Darstellung von Werten und deren Attribute. Wenn dargestellt werden soll wie Daten in Relation zueinander stehen, bedarf es anderer Konzepte. Solche Abhängigkeiten können hierarchische Abhängigkeiten wie *Parent*/*Child*-Beziehungen sein oder die Relation von Objekten untereinander [3].

Eine der üblichsten visuellen Darstellungsarten von Hierarchien ist die Baum-Darstellung durch das *node-link-tree-diagram.* Der Aufbau ist dabei ähnlich dem eines Baumes. Es gibt Elemente, Ebenen und Verzweigungen. Das erste *Parent*-Element befindet sich dabei auf erster Ebene, dessen *Children* auf zweiter Ebene, verbunden mit dem Parent-Element. Dieses Prinzip wird mit den Children der Children usw. fortgeführt. Elemente die keine weiteren Children haben, werden *Leaves* genannt. Gewisse Konventionen, wie das Einhalten der Ebenen, Vermeidung von Überschneidungen der Linien, Kleinhalten der Darstellungsfläche und das Halten einer eines symmetrischen Aufbaus sind dabei zu beachten um das Verständnis der Grafik zu vereinfachen. Abbildung 3.3 zeigt ein Beispiel eines solchen Diagramms. Ein weiteres gängiges Beispiel einer solchen Darstellung ist auch der Stammbaum.

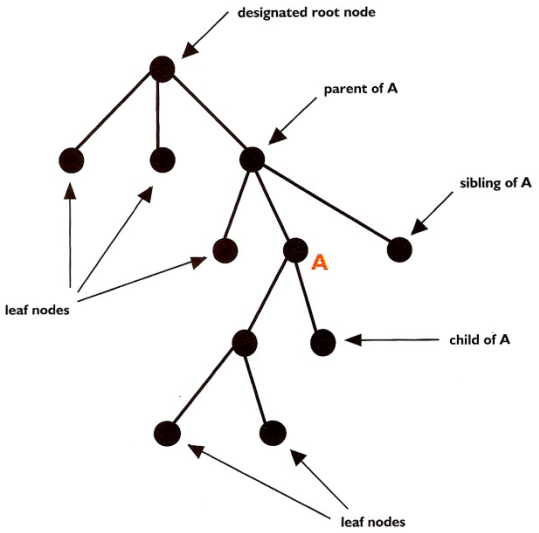

Abbildung 3.3: *Tree-Diagram* [7].

Da diese Grafiken bei höherer Komplexität schnell unübersichtlich werden können eignet sich in diesem Fall die Darstellung durch *Cone-Trees* im dreidimensionalen Raum (Abbildung 3.4). Bei dieser Darstellungsart sind die Children eines Elementes radial abgehend arrangiert. Dabei ist zu beachten einen geeigneten Radius und Abstand zum Parent-Element zu wählen um Überschneidungen zu vermeiden. Hierbei ist es sinnvoll die Elemente nur auf der Cone-Oberfläche zu platzieren, um das Verständnis der Grafik einfach zu halten [9]. Da die Darstellungen durch das Node-Link-Tree-Diagramm und des Cone-Trees viel Platz in Anspruch nehmen können, werden diese auch als *Non-Space-Filling* Methoden beschrieben [4].

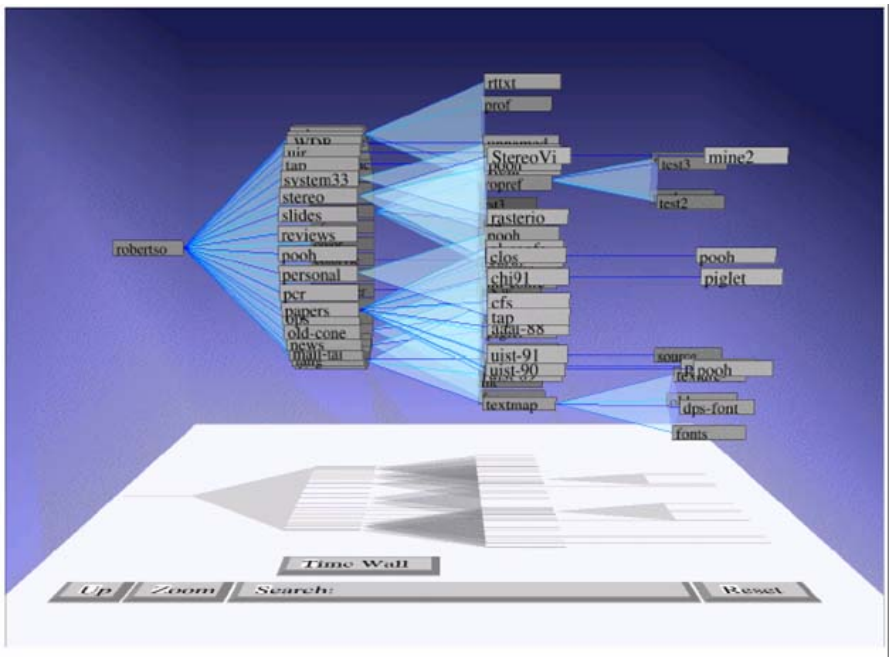

Abbildung 3.4: *Cone-Tree-Diagram* [4].

Space-Filling Methoden sind Darstellungsarten welche den Raum füllend nutzen [4]. Das *Treemap*-diagramm (Abbildung 3.5) ist dabei besonders geeignet, da die Elemente ineinander verschachtelt sind. Jede Form, dies kann ein Rechteck sein, bildet dabei ein Ast welcher die Unteräste in Form von kleineren verschachtelten Rechtecken enthält. Gruppierungen können farblich markiert werden [5].

Abbildung 3.5: Treemap-Diagramm mit rechteckigen Formen [113].

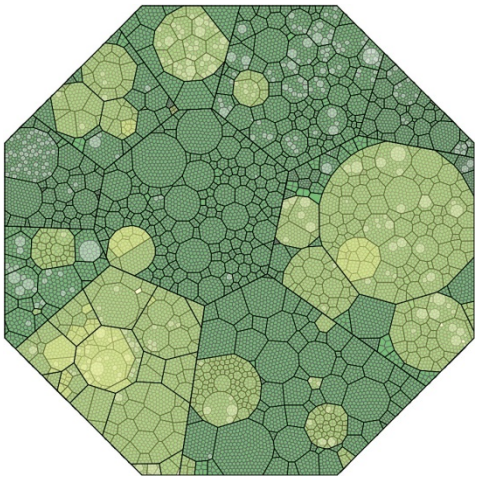

Abbildung 3.6: Treemap-Darstellung mit Voronoi-Muster [21].

Eine weitere Möglichkeit hierarchische Daten platzsparend darzustellen bildet die radiale Methode (Abbildung 3.7) bei der sich das erste Knotenelement im Zentrum des Kreises befindet und jede weitere Schicht nach außen hin eine neue Child-Ebene darstellt. Dabei können die Kreissegmente unterteilt werden um mehrere Elemente als Geschwister zu zeigen.

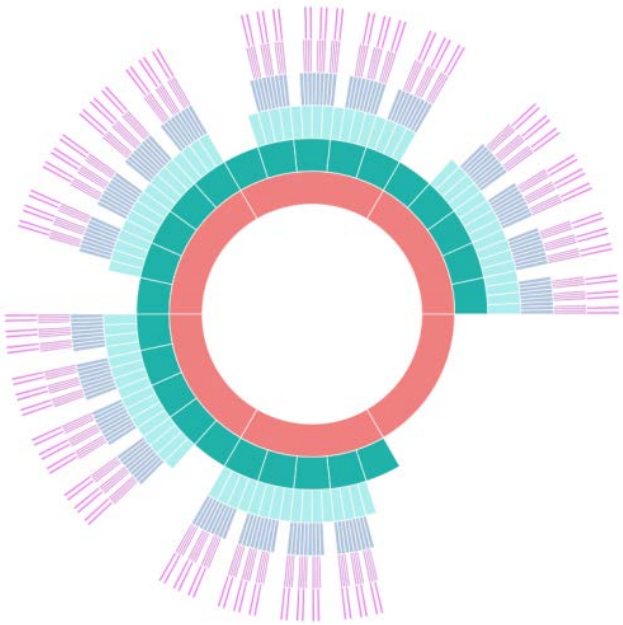

Abbildung 3.7: Radiale Darstellung von hierarchischen Daten [114].

3.2 Interaktive Datenvisualisierung

Neben der Repräsentation von Daten in der Datenvisualisierung spielt auch die Interaktion eine wichtige Rolle. Besonders komplexere Daten können so vom Betrachter einfacher verstanden werden. Zum Beispiel durch Exploration lässt sich vorerst eine geringere Menge an Informationen darstellen und erst nachdem eine Interaktion erfolgt ist wird eine bestimmte Information erweitert. Für die Interaktion zwischen Benutzer und Daten gibt es eine Reihe an Grundkonzepten die auch kombiniert werden können [11]:

- *Selektion*: Hierdurch lassen sich bestimmte Teile markieren, etwa durch Hervorheben, Verkleinern oder anderweitiges Bearbeiten. Somit verliert der Benutzer eine Information nicht aus den Augen, auch wenn schon eine weitere betrachtet wird.
- *Exploration*: Ermöglicht die Untersuchung von Subinformationen eines Datensatzes, die angezeigt (oder auch ausgeblendet) werden können, nach

dem die Interaktion vom Benutzer ausgeführt wurde (wie zum Beispiel ein Klick oder das näher Herankommen an ein Objekt).

- *Filtern*: Vom Benutzer definierte Informationen können hiermit hervorgehoben oder ausgeblendet werden. Zum Beispiel durch die Eingabe eines Suchwortes werden alle Objekte im Datensatz, welche das eingegebene Wort enthalten, in anderer Farbe dargestellt.
- *Rekonfigurierung*: Dem Benutzer wird die Ansicht mit Hilfe einer Neuanordnung, bzw. durch eine andere Darstellung der Objekte im Raum, aus einer anderen Perspektive gezeigt. So kann zum Beispiel ein selektiertes Objekt in den Vordergrund oder die Mitte treten, während weniger relevante Objekte in den Hintergrund treten.
- *Dekodierung*: Ermöglicht eine andere Art der Repräsentation der Daten.
- *Verbinden*: Ermöglicht das Aufzeigen von Abhängigkeiten, zum Beispiel durch Graphen oder die Hervorhebung voneinander abhängiger Objekte.
- *Abstrahieren*: Ermöglicht dem Benutzer den Grad der Detaillierung zu modifizieren.
- *Navigieren*: Der Benutzer hat hiermit die Möglichkeit zu zoomen, die Position der Kamera zu ändern oder die dargestellten Objekte zu rotieren.

3D und VR in Datenvisualisierungen

Der 3D-Raum bietet zum Vergleich des 2D-Raums einige Vorteile. Mit der dazugewonnenen Dimension lassen sich weitaus mehr Informationen darstellen und aus der zweiten Dimension heraus erweitern. Zum einen bietet es Möglichkeiten die Repräsentation aus verschiedenen Perspektiven zu zeigen und dadurch bestimmte Informationen sichtbar zu machen. Zum anderen bietet es die Möglichkeit eine zusätzliche Achse zu nutzen, um zum Beispiel Zeit oder andere Werte zu verdeutlichen.

Innerhalb einer *Virtual Reality* (VR) Umgebung kann interagiert werden ohne die Umgebung der Repräsentation zu verlassen. Dadurch wird es dem Betrachter ermöglicht, sich noch tiefgehender mit der Visualisierung zu beschäftigen und intuitiver zu interagieren [12].

3.3 Softwarevisualisierung

Die Visualisierung von Software ist ein Feld der Datenvisualisierung, die es ermöglicht Software auf verschiedenen Ebenen dazustellen. Sie lässt sich in drei Grundebenen aufteilen: die Code- und Klassenebene, die Algorithmus-Ebene und die

Architekturebene, welche im Folgenden näher beschrieben werden [4]. Da sich diese Arbeit auf die statische Softwarevisualisierung bezieht, wird auf die Algorithmus-Ebene nicht tiefer eingegangen, da sie ein Teil der dynamischen Softwarevisualisierung ist.

3.3.1 Repräsentation auf Code-Ebene

Eine Variante für die Visualisierung von Code ist die *Line-Repräsentation* [10]. Wie in der Abbildung 3.8 zu sehen ist, werden die einzelnen Zeilen so stark verkleinert bis nur noch eine Reihung von entsprechend farbigen Pixeln zu sehen ist. Dabei wird das Layout beibehalten. Mit Hilfe farblicher Unterscheidungen lassen sich bestimmte Teile des Codes differenzieren, wie durch die Zuweisung bestimmter Farben für bestimmte Methoden.

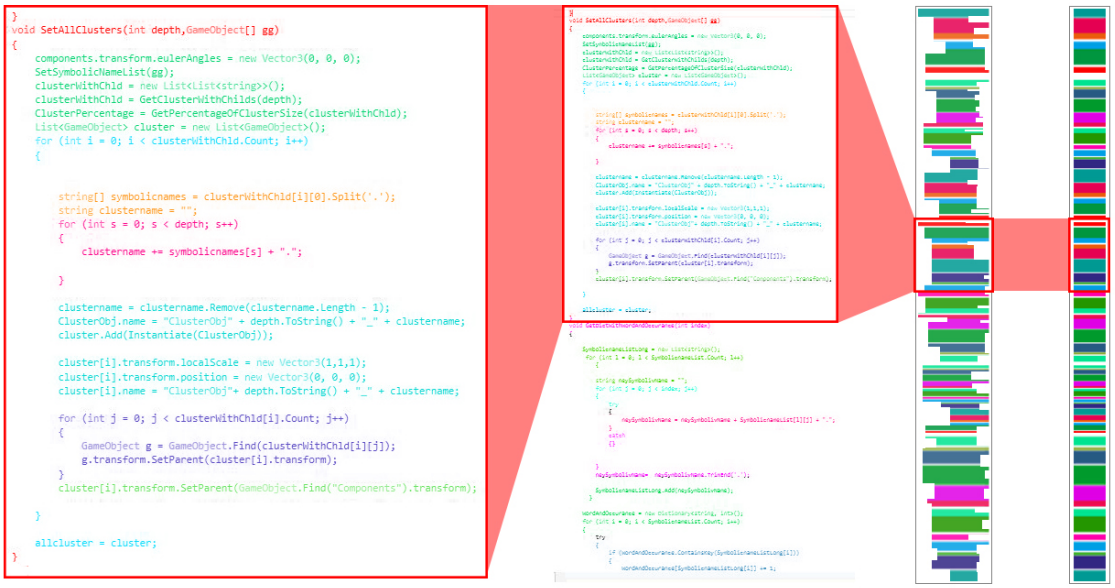

Abbildung 3.8: Line- und Pixelrepräsentation.

Mit Hilfe der *Pixel-Repräsentation* lassen sich weitaus mehr Informationen auf einer bestimmten Fläche darstellen. Das Prinzip ist das gleiche wie bei der Line-Repräsentation, jedoch wird das Layout vernachlässigt, so dass es theoretisch möglich ist eine Codezeile mit nur einem Pixel dazustellen da die Breite keine Information mehr trägt. So stellt eine Spalte genau eine Datei dar. In Abbildung 3.8 ist diese Art der Darstellung auf der rechten Seite zu sehen. Durch die Aneinanderreihung verschiedener Dateien in Form von Pixelspalten von links nach rechts lassen sich diese einfach anhand ihrer Höhe auf ihre Größe hin untersuchen und vergleichen.

3.3.2 Repräsentation auf Klassenebene

Eine weitere mögliche und viel verwendete Methode für die Darstellung der Klassen und vor allem deren Abhängigkeiten bieten die *Unified Modeling Language* (UML) Klassen und Objekt-Diagramme (Abbildung 3.9). UML ist eine grafische Modellierungssprache welche von der *Object Management Group* (OMG) entwickelt wurde. Die Abbildung zeigt ein solches UML Diagramm. Jedes Rechteck stellt eine Klasse dar und ist in drei Teile aufgeteilt: Der Kopf mit dem Klassennamen und die folgenden zwei Abschnitte mit Attributen und Methoden. Abhängigkeiten werden als Graphen gekennzeichnet. Dabei wird zwischen *Aggregation* und *Komposition* unterschieden. Die Aggregation (unausgefülltes Rauten-Symbol) ist eine Kann-Beziehung zwischen zwei Klassen, wo hingegen bei der Komposition (gefülltes Rauten-Symbol) die Abhängigkeit zwingend ist.

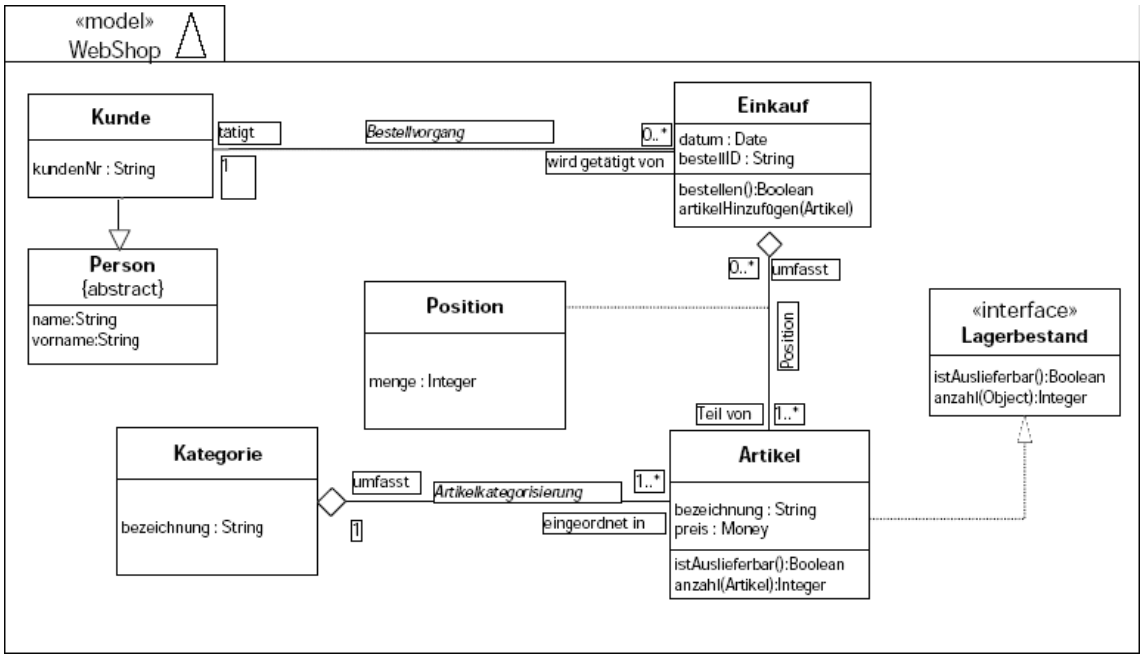

Abbildung 3.9: UML-Diagramm [116].

UML stellt neben den Klassendiagrammen weitere Diagrammtypen wie das Sequence-Diagramm oder das Aktivitäten-Diagramm zur Verfügung. Neben dem UML Klassendiagramm gibt es noch weitere verwandte Typen. Ein Beispiel dafür wäre das *Geon-Diagram* welches im 3D-Raum dargestellt wird (Abbildung 3.10).

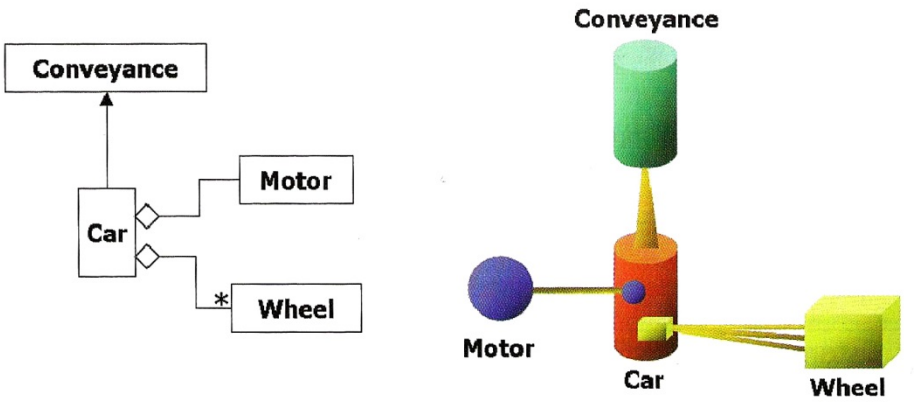

Abbildung 3.10: UML- und Geon-Diagramm im Vergleich [4].

3.3.3 Repräsentation auf Architekturebene

Um Softwarearchitektur zu beschreiben gibt es viele Konzepte zu beachten: Die Organisation, Zusammenhänge einzelner Komponenten, Protokolle für Kommunikation, Services, Zugriffsrechte auf Daten etc. Es gibt einige grundlegende Architekturkonzepte auf die Anwendungen aufbauen können. Eine Architektur muss dabei nicht auf ein Konzept beschränkt sein. Zum Beispiel kann eine *Service Orientated Architecture* (Service-Orientierte Architektur) zugleich auch ein *Layered System* (Ebenensystem) sein wie es bei OSGi der Fall ist [103]. Abbildung 3.11 zeigt einige dieser Grundkonzepte welche im folgendem beschrieben werden. Softwarearchitektur ist eine nicht greifbare konzeptionelle Einheit, daher ist es schwer sie zu verstehen ohne sie visuell darzustellen. Aufgrund dessen ist die Visualisierung von Softwarearchitektur eins der wichtigsten Felder der Softwarevisualisierung [6].

- *Pipes and Filters*: Datenströme „fließen" in einem Filter. Von da aus kann die Abfolge der Arbeitsschritte sortiert und ausgegeben werden. So können die Daten vom Filter modifiziert zum nächsten Filter weitergeleitet werden.
- *Component-Based*: Unterteilt das Softwareprojekt in mehrfach verwendbare Komponenten, auf die zugegriffen werden können.
- *Layered Systems*: Das Softwareprojekt ist in verschiedenen Ebenen sortiert. Jede Ebene kann auf die Funktionalität der unmittelbar darunter liegenden Ebene zugreifen. Einige Systeme erlauben das Zugreifen auf entferntere oder höher liegende Ebenen.
- *Service Orientated Architecture*: Basierend auf Schnittstellen und deren Strukturen wie auch auf allgemeingültiger (wieder-)verwendbarer Services. Dieses Konzept der Softwarearchitektur repräsentiert eine oder mehrere Funktionen als Service. Die Beschreibung der Service-Schnittstelle ist platt-

formneutral. Service-Implementierungen sind wiederverwendbar, gekapselt und lose gekoppelt. Die Kommunikation mit den Services wird über eine standardisierte Infrastruktur realisiert. [13]

- *Blackboards*: Mehrere Teilbereiche eines Softwareprojekts greifen auf eine Hauptkomponente zu. Von ihr aus kann gelesen und geschrieben werden

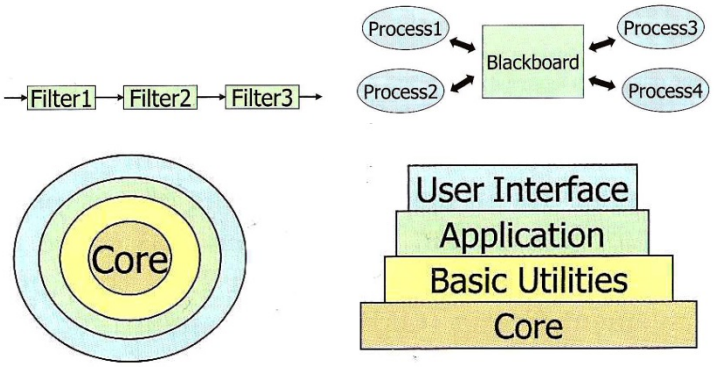

Abbildung 3.11: *Pipes and Filters*, *Blackboards* und *Layered Systems* im einfachen Visualisierungsbeispiel [4].

3.3.4 Visuelle Metaphern

Mit einer visuellen Metapher wird ein bekanntes visuelles Konzept auf die zu erstellende Anwendung übertragen. Ein Beispiel hierfür ist das Node-Link-Diagramm welches in Kapitel 3.1.2 genauer beschrieben wird. Das Konzept des Baumes mit seinen Verzweigungen, Ästen und Blättern wird dabei auf die Datenvisualisierung übertragen und vereinfacht das Verständnis, da es an ein bereits bekanntes Konzept anlehnt. Neben der Anordnung können auch Farbe und Form einzelner Elemente visuelle Metaphern sein. Die Farbe Rot zum Beispiel wird oftmals verwendet um Daten hervorzuheben, für die eine besondere Aufmerksamkeit erwünscht ist.

3.4 Umsetzungsmöglichkeiten interaktiver Datenvisualisierung

3.4.1 Software

Um eine Datenvisualisierung umzusetzen, bietet sich die Hilfe einiger Software-Frameworks an. Die richtige Wahl kann dann getroffen werden, wenn die Art der Visualisierung klar ist. Je nach Dimensionalität, benötigter Interaktionsmöglichkeiten, der Darstellungsumgebung, Anwendungszweck und Programmiersprache lässt sich ein entsprechendes Tool auswählen. Frameworks wie *Data-Driven Documents (*D3*)* sind bereits auf Datenvisualisierungen abgestimmt und bieten eine Reihe an Darstellungskonzepten an, welche abgeändert werden können und mit eigenen Datensätzen befüllt werden können. Mit Hilfe dieses Frameworks

kann eine interaktive Datenvisualisierung durch die Benutzung der Webstandards HTML5, CSS und SVG generiert werden [15].

Solche Frameworks bieten zwar Vorteile durch die schon verfügbaren Darstellungskonzepte, mit denen sich relativ schnell Ergebnisse erzielen lassen. Dies kann jedoch auch Einschränkung bedeuten in Bezug auf die Darstellungen eigener Konzepte. Des Weiteren gibt es relativ wenige Methoden für die Darstellung komplexer, dreidimensionaler Räume. Um etwas in dieser Art umzusetzen eignen sich Frameworks für die allgemeinere Art der Umsetzung von Computergrafiken und Animationen. *OpenGL (Open Graphics Library)* ist eine Programmierschnittstelle, die es ermöglicht, dreidimensionale Objekte zu erstellen und in Echtzeit darstellen zu lassen. Das Tool wird in sehr vielen Bereichen genutzt, und ist daher ein weltweiter Standard. Das Arbeiten hiermit ist allerdings nicht besonders intuitiv und setzt Grundkenntnisse der 3D-Computergrafik voraus [104].

Geeignet ist auch die *Unity-Engine*, welche überwiegend in der 2D- und 3D Spieleentwicklung eingesetzt wird. Sie bietet eine Reihe von Standard-3D-Objekten an, ebenso Funktionen zur Verwendung extern erstellter 3D-Grafiken [105]. Zudem stehen Funktionen für Animation und Physikberechnungen zur Verfügung. Die Grafikdarstellung basiert auf OpenGL und Direct3D, eine Programmierschnittstelle von Microsoft mit der es möglich ist möglichst direkt auf die Hardware eines Computers zuzugreifen [109]. Da Unity eine eigene Software-Oberfläche besitzt, vereinfacht es die intuitive Arbeitsweise und die Verwaltung des Datenvisualisierungs-Projektes. Ein großer Vorteil ist die Möglichkeit das Projekt für mehrere Plattformen zu exportieren. Das bedeutet, dass es ohne viele Modifikationen möglich ist, das Projekt als *Android-* oder *IOS-Applikation* auszugeben. Einige VR-Frameworks, wie zum Beispiel *Google Cardboard*, werden von Unity unterstützt und lassen sich einfach in das Projekt integrieren. Zu beachten ist allerdings, das Unity nicht Programmiersprachen-unabhängig ist, Projekte werden mit C# umgesetzt.

3.4.2 Hardware

Besonders für VR-basierte Repräsentationen eignet sich der Zusatz von Hardware. VR-Brillen ermöglichen eine intuitive Interaktion, da sie die Kopfbewegung des Betrachters erfassen und diese in der Darstellung mit einbezogen werden. Dies erweckt den Eindruck sich selbst im Raum zu befinden. Einige Modelle, wie die *Vive* des Unternehmens *HTC* (*High Tech Computer Corporation*) ermöglichen auch, die Position in einem (begrenzten) Raum mit Hilfe zusätzlicher Sensoren zu ermitteln. So kann in einem Raum auf natürliche Weise (Gehen) navigiert werden [110].

Die Interaktionsmöglichkeiten, die bei einer VR-Brille nur auf die Kopfposition und -bewegung eingeschränkt sind, lassen sich durch Controller erweitern. Neben den fernbedienungs-artigen Modellen, die einige Trigger-Knöpfe und ggf. auch einen 360-Grad-Knopf besitzen, gibt es Modelle die zusätzlich auch die Position und Lage des Controllers an sich erkennen, und diese in den VR-Raum mit einbinden können.

3.5 Lösungen interaktiver Softwarearchitektur-Visualisierungen

Um eine Software visuell zu repräsentieren, können die hier genannten Darstellungsmöglichkeiten verwendet werden. Sinnvoll wird es allerdings erst dann, wenn die verschiedenen Techniken gut kombiniert werden. Techniken der Daten- und Softwarevisualisierung müssen mit dem Konzept der Interaktion ineinandergreifen. Die hier gezeigte Umsetzung einer interaktiven Softwarevisualisierung zeigt zum einem den Einsatz verschiedener Datenvisualisierungstechniken und zum anderen den Einsatz von VR, das auch in der Umsetzung des eigenen Konzepts eine große Rolle spielt.

Das DLR forscht an der Visualisierung von Softwarearchitekturen und entwickelt insbesondere Tools zur Visualisierung von OSGi-basierten Anwendungen [18][19]. Der Fokus liegt dabei auf den Modulen der Software und deren Abhängigkeiten wie auch die Extraktion der dafür benötigten Daten. Bei der Umsetzung der Visualisierung handelt es sich um ein interaktives Browser-Tool, welches mit dem auf Java-Script basierten D3 Framework umgesetzt wurde [15]. Für den VR-Teil der Arbeit wurde Unity genutzt. Abbildung 3.12 bis Abbildung 3.14 zeigen die grafische Umsetzung mit D3 und Abbildung 3.16 die Umsetzung in VR.

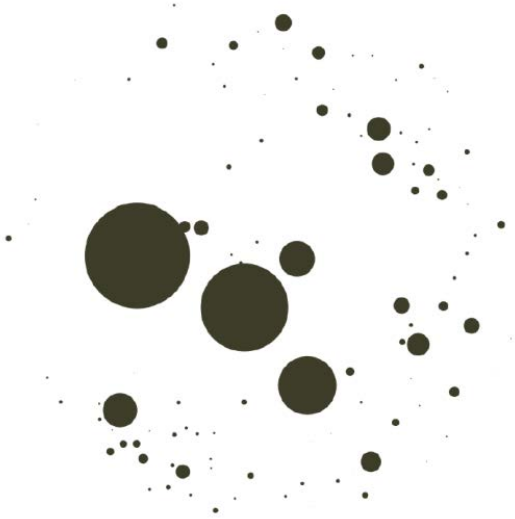

Abbildung 3.12: Darstellung der OSGi-Bundles. Die Größe der einzelnen Kreise stellt die Anzahl der Packages pro Bundle dar [19].

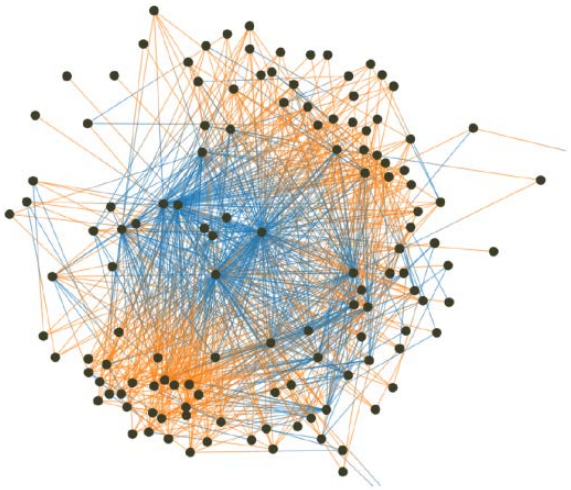

Abbildung 3.13: Package-Abhängigkeiten zwischen den Bundles [19].

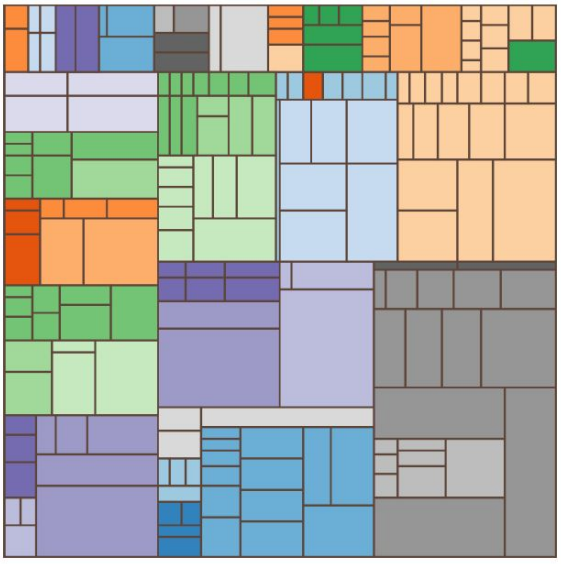

Abbildung 3.15: Treemap-Darstellung eines Bundles, welches die Packages und Klassen zeigt. Gleiche Farbe bedeutet ein Package [19].

Abbildung 3.14: Übersicht aller Services und Servicekomponenten [19].

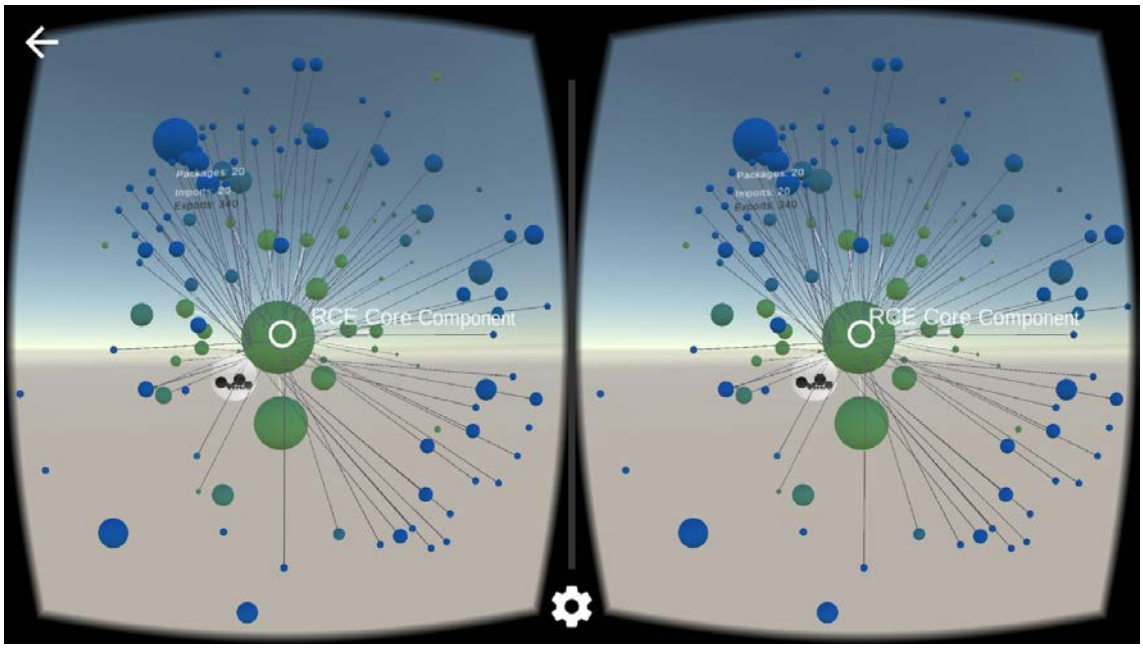

Abbildung 3.16: VR-Version der OSGi Visualisierung. Die Kugeln stellen Bundles dar, welche bei Interaktion transparent werden um die Packages (schwarze Kugeln) zu enthüllen. Die Größe der Bundles steht wie bei der 2D-Ansicht für die Anzahl der Packages [18].

4 Konzept

4.1 Zielgruppe

Die Anwendergruppe wird vorher definiert um die Visualisierung auf ihre Interessen abzustimmen.

- Softwareentwickler sollen mit der Repräsentation das dargestellte Softwareprojekt besser verstehen lernen. Besonders bei größeren Softwareprojekten soll so die Komplexität schneller ersichtlich sein. Die Zusammenhänge gewisser Komponenten sollen visualisiert werden, und damit zu einem besseren Verständnis führen. Für Entwickler in einem Team können weitere Information wichtig sein, wie zum Beispiel wer welchen Code wann bearbeitet hat.
- Menschen, die indirekt an dem Projekt mitarbeiten und keine Softwareentwickler sind, sollen ein besseres Bild des Projektes und dessen Umfang bekommen – auch wenn nicht alle Details für Unerfahrene ersichtlich sein werden, soll es möglich sein gewisse Konzepte zu verstehen.
- Neuen Mitgliedern im Team und/oder Anfängern muss ein erstes Bild des Projektes übermittelt werden. Die Einarbeitungszeit kann anfangs sehr schwer für diese Zielgruppe sein. Verschiedene Abstraktionsgrade der Visualisierung von grob bis fein können helfen, die Betrachter an das neue Projekt heranzuführen.
- Interessierte Menschen, die OSGi lernen und verstehen wollen, können mit Hilfe dieser Darstellung die Grundkonzepte der Architektur vor Augen geführt bekommen. Auch für diese Zielgruppe eignet sich eine vorerst gröbere Ebene der Darstellungsart.

4.2 Was wird dargestellt

Die Daten, die repräsentiert werden, können sehr umfangreich sein und werden daher nicht alle berücksichtigt. Eine klare Definition des Fokus ist daher vorab festgelegt.

- Bundles: Die Anzahl aller Bundles und deren Namen, symbolischen Namen, die Anzahl der Imports und die Anzahl der Exports. Die Packages die ein Bundle importiert, sollen als Imports erkenntlich gemacht werden.
- Alle Packages eines Bundles und deren Namen. Packages, welche exportiert werden, sollen als exportierendes Package erkenntlich gemacht werden.

- Die Klassen innerhalb der Packages sollen dargestellt werden wie auch deren Namen.
- Die zur Verfügung gestellten Services eines Packages sollen dargestellt werden.

Die Arbeit berücksichtigt nicht die dynamische Visualisierung einer Software. In diesem Fall bedeutet das, dass der LifeCycle der Bundles nicht mit einbezogen wird.

4.3 Repräsentation

Die visuelle Repräsentation soll aus einem Zusammenschluss von geeigneten Diagrammen, den einzelnen Objekten und dem Interaktionsmodell bestehen. Da die Datenmengen teilweise sehr groß sein können, soll ein besonderes Augenmerk auf die Wahl der grafischen Variablen gelegt werden welche in Kapitel 3.1 genauer beschrieben sind. Für sich allein lassen sich die grafischen Variablen einfacher erfassen, doch im Vergleich zueinander gibt es mehr und weniger geeignete. Die Einfachheit der Erfassung von grafischen Variablen lässt sich nach McGill und Clevelands Theorie *Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods* [16] wie in Abbildung 4.1 gezeigt auflisten. Für die Erstellung des Konzeptes wird diese Theorie berücksichtigt.

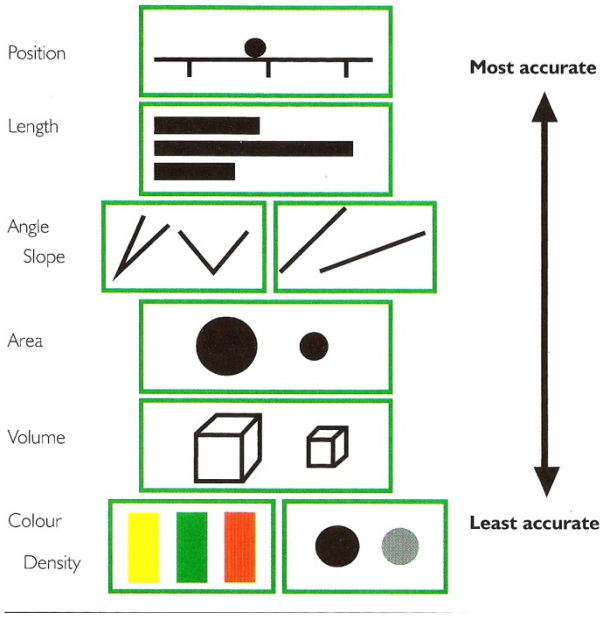

Abbildung 4.1: Grafische Variablen sortiert nach Einfachheit der Erfassung [7].

4.3.1 Visuelle Metapher

Wie in Kapitel 3.3.4 detaillierter beschrieben, hilft es, dem Betrachter eine visuelle Metapher anzubieten. In diesem Fall ist es sinnvoll eine Metapher zu finden die das

Konzept der Modularität von OSGi-basierten Anwendungen gut repräsentiert wie auch deren Abhängigkeiten. Besonders geeignet hierfür ist die Darstellung elektronischer modularer Systeme (Abbildung 4.2): Einzelne Komponenten sind austauschbar, sie können miteinander verbunden verschiedenste Konstellationen aufweisen und besitzen eine Vielzahl von Kontroll-Optionen, wie etwa Potis und Schieberegler, welche sich gut für individuelle Einstellungen eignen. Konzeptionell soll diese Metapher umgesetzt werden, allerdings grafisch einfach gehalten werden, um nicht vom Wesentlichen abzulenken.

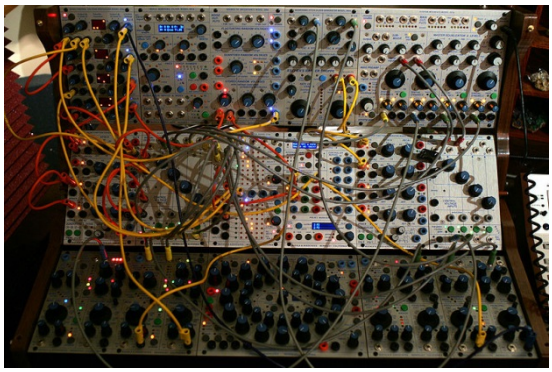

Abbildung 4.2: Modulares Synthesizer-System [116].

4.3.2 Darstellung der Bundles und Packages

Jedes Bundle wird als kubisches Objekt dargestellt. Jedes Package wird als andersfarbiges kubisches Objekt dargestellt, welches sich in der Größe vom Bundle unterscheidet. Es soll zwar erkenntlich werden, dass das Package ein Teil eines Bundles ist, doch soll es in der Darstellung nicht innerhalb eines Bundles liegen, da das die Übersicht negativ beeinflussen kann. Um die Zugehörigkeit der Packages erkenntlich zu machen, sollen diese auf dem Bundle gestapelt werden. Somit kann verdeutlicht werden, dass das Bundle das Package „trägt“. Wenn das Bundle inklusive Packages betrachtet wird, gibt die Höhe des kompletten Objekts Auskunft über die Anzahl der Packages und kann schnell mit in der Umgebung liegende Bundles verglichen werden.

4.3.3 Darstellung der Services

Services, die von einem Package bereitgestellt werden, werden als Scheibenobjekte so in das Package platziert, dass sie überstehen und somit erkennbar sind. Bei einem bereitgestellten Service per Package ist das Serviceobjekt am oberen Rand platziert. Mehrere bereitgestellte Services werden untereinander platziert, mit einem kleinen Abstand, um sie unterscheidbar zu machen. Abbildung 4.3 zeigt die Services in Kombination mit Packages und deren Bundles.

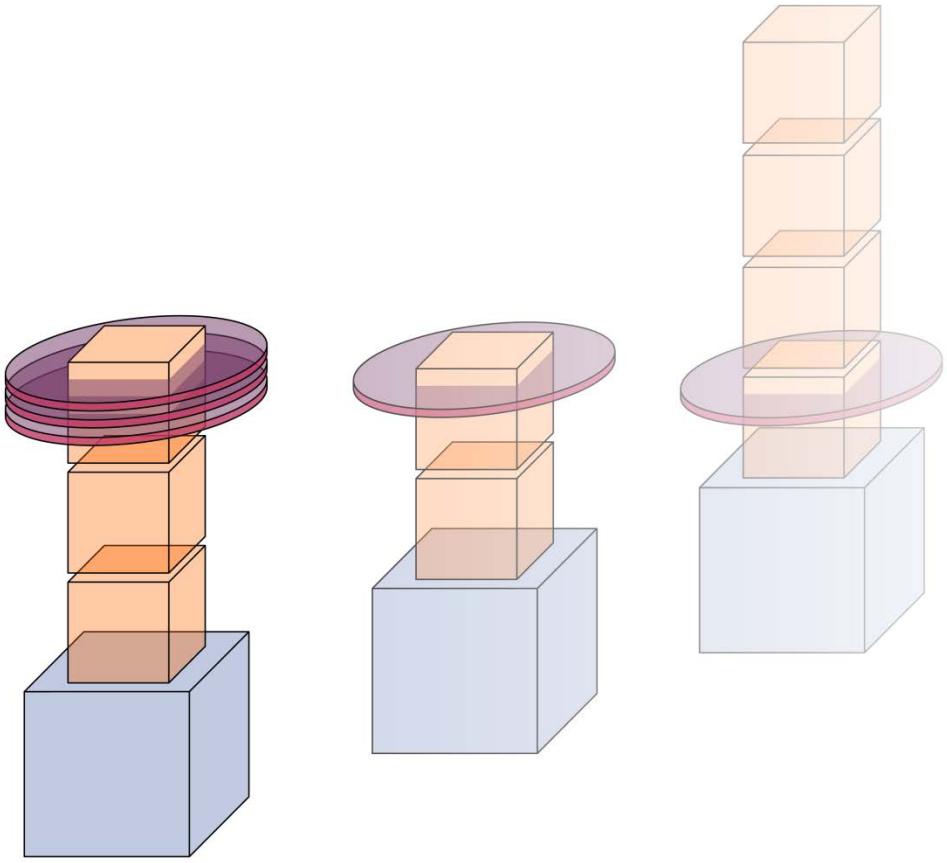

Abbildung 4.3: Darstellung der Bundles (Blau) und deren Packages (Orange). Services, die von einem Package zur Verfügung gestellt werden, sind als Scheiben dargestellt.

4.3.4 Anordnung der Bundles

Die Basis-Szene soll die Darstellung aller OSGi-Bundles zeigen. Sie sollen im dreidimensionalem Raum an verschiedenen Positionen dargestellt werden. Standardmäßig sollen diese in der X- und Z-Achse zufällig angeordnet werden. Da das Feld vor allem bei größeren OSGi Projekten viele Bundles zeigt, sollen optional weitere Layout-Optionen zur Verfügung stehen.

Optional soll es die Möglichkeit geben, die Bundles zu Clustern. Das bedeutet Gruppieren und Platzieren der Bundles in kleinere Felder. Dadurch soll die Übersichtlichkeit verbessert werden. Die Art der Aufteilung gibt der symbolische Name des Bundles vor. Um den Vorgang der Unterteilung zu ermöglichen, sollte der symbolische Name des Bundles nach der *Reverse-Domain-Konvention* benannt worden sein, welche von der *OSGI-Core-Specification* empfohlen und vorgegeben wird [1][17]. Die Reverse-Domain-Konvention besteht an der ersten Stelle aus einer *Top-Level-Domain*, welche den letzten Abschnitt einer Domain beschreibt. Beispie-

le dafür sind unter anderem „de“ „com“ oder „org“. Abgetrennt mit einem Punkt folgt der zweite Name, welcher den Namen des Unternehmens oder des Projektes darstellt. Wiederum mit einem Punk abgetrennt kommet der Teil, welcher zusammengehörige Packages beschreibt wie „utils“ , „gui“ oder „core“, gefolgt von weiteren Begriffen, die die Definition des Bundles immer weiter eingrenzen. Bei der Aufteilung einzelner Bundle-Gruppen sollen diese Namen verglichen werden. Die Tiefe der Aufteilung kann eingestellt werden. Diese entscheidet, bis zu welchem ‚.‘ verglichen wird. Je nach Anforderung soll es verschiedene Möglichkeiten der Clusterung geben. Die Anordnung im quadratischen Feld soll bewirken, dass die Abhängigkeiten so gut wie möglich erfassbar sind, indem so viele Bundles wie möglich mit einem Blick zu erkennen sind. Die Anordnung in der Reihe soll eine sortierte Anordnung darstellen.

4.3.5 Anordnung im quadratischen Feld

Alle Cluster sollen auf ein quadratisches Feld in der X-Z-Ebene in kleinere, auf der Ebene aufliegende Quader aufgeteilt werden. Die Länge jedes Quaders in der X-Achse ist fest definiert und entspricht der Seitenlänge des Feld-Quadrates. In der Z-Achse soll jeder Cluster-Quader die Länge des prozentualen Anteils des Clusters zum Gesamten betragen. Das bedeutet, dass die Größe des Clusters , die sich an der Anzahl der Bundles definiert, genauso so im Verhältnis steht, wie die Länge der Z-Achse entlang des Clusters zu der gesamten Feldfläche. Dadurch, dass die Seite der X-Achse bei allen Gruppen gleich lang ist, lassen sich die Größenverhältnisse untereinander einfacher erkenntlich machen. Abbildung 4.4 zeigt die Unterteilung einzelner Gruppenfelder. Die dort gezeigten Linien in Orange sollen nicht in der Darstellung gezeigt werden um die Übersicht nicht zu beeinträchtigen. Für die Unterscheidung der Gruppierungen dienen die Farben der Bundles. Die Bundles jeder Gruppe kriegen somit eine eigene per Zufall ausgewählte Farbe zugeteilt und sind dadurch als Gruppe erkenntlich.

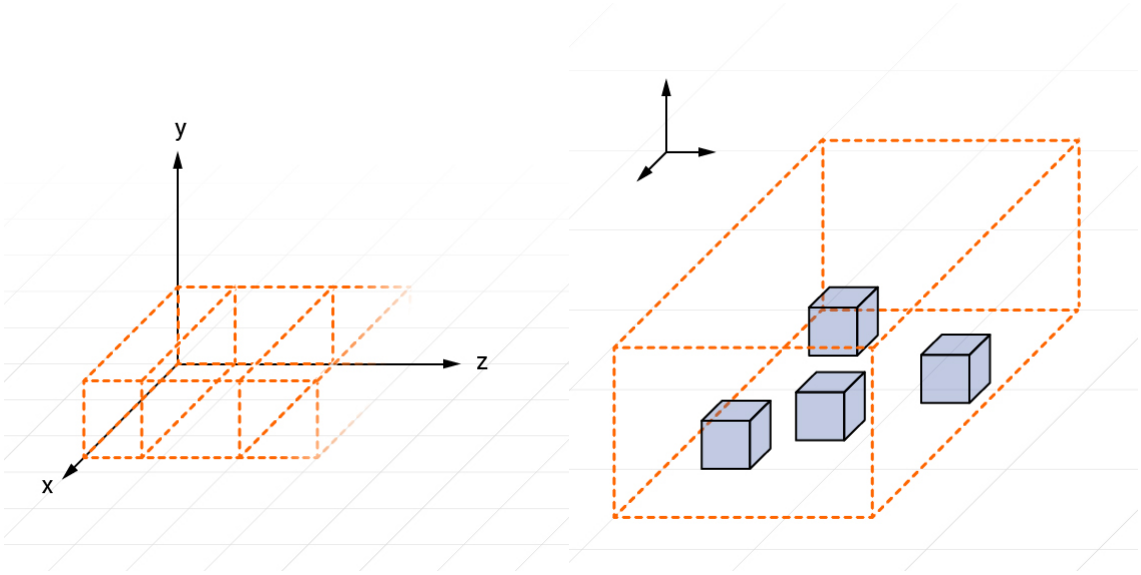

Abbildung 4.4: Clusterung im quadratischen Feld.

Innerhalb eines Clusters werden die Bundles angeordnet wie in Abbildung 4.4 auf der rechten Seite dargestellt. Die Größe des Bundles, welche hier durch die Anzahl der Packages definiert wird, soll anhand der Position in der Z-Achse verdeutlicht werden. Das bedeutet, je mehr Packages das Bundle hat, desto weiter hinten ist es zu sehen. Somit lassen sich bei Betrachtung von vorne alle Bundles ersichtlich machen, da die Bundles, die am höchsten sind, zugleich an hinterster Position stehen. Allerdings soll die Position nicht auf den Punkt genau definiert werden, sondern pro Package-Anzahl ein Bereich, da es wahrscheinlich ist, dass im einem Projekt eine gewisse Package-Anzahl um ein vielfaches öfter auftaucht als eine andere. Würde eine hohe Vielzahl an Bundles in der gleichen X-Ebene stehen, wäre die Übersichtlichkeit nicht mehr gegeben und die meisten Bundles würden sich an bestimmten Positionen optisch überschneiden, während die Bundles mit besonders vielen oder besonders wenigen Packages vereinzelt im Raum liegen und dadurch von der gesamten Darstellung abgekoppelt wirken.

In der X-Achse innerhalb eines Clusters sollen die Bundles zufällig angeordnet werden. Denkbar wäre auch hier eine Positionierung an der Achse entlang, wie zum Beispiel durch Verhältnisse von Imports zu Exports oder der Anzahl von Klassen. Da allerdings ein Bundle mit einer höheren Anzahl an Packages in den meisten Fällen auch eine höhere Anzahl an Klassen hat und oftmals auch eine höhere Anzahl an Exports, wie auch umgekehrt, würden sich die Bundles eher unübersichtlich im Raum verteilen. Die Wahl einer zufälligen Anordnung scheint daher den Anforderungen am meisten zu entsprechen.

Die Position einzelner Bundles in der Y-Achse soll dabei gleichbleibend sein. Dadurch lassen sich Höhenunterschiede, also die Anzahl der Packages einzelner Bundles, besser vergleichen und die Anordnung entlang der X-Achse wird erkenntlicher.

4.3.6 Gereihte Anordnung

Bei dieser Darstellung soll jedes Cluster entlang der X-Achse positioniert werden. Der Abstand zwischen den einzelnen Gruppen wie auch der Abstand zwischen den Bundles in der Z-Achse ist gleich. Zusätzlich soll jede einzelne Reihung einer bestimmten Kurve in der X-Z-Achse anliegen. Dadurch wird ermöglicht, dass auch von einem frontalen Ausblick aus eine möglichst hohe Anzahl von Bundles erkenntlich ist. Als geeignete Anordnung in der X-Z-Ebene erweist sich folgende Kurve:

$$x = z^2/160$$

Innerhalb der Gruppierungen werden die Bundles an der X-Achse entlang gereiht, mit jeweils gleichem Abstand. Die Y-Position einzelner Bundles per Cluster wird dabei ähnlich wie in der X-Z-Achse durch eine Funktion ermittelt. Durch die Versetzung einzelner Bundles dieser Kurve entlang wird die Übersichtlichkeit noch weiter erhöht, da so noch weniger Elemente übereinanderliegen, wenn der Betrachter frontal auf die Darstellung blickt.

$$y = -x^3/360$$

Auch hier werden die Bundles einzelner Cluster mit unterschiedlichen, per Zufall gewählten Farben, versehen. Abbildung 4.5 zeigt die Bundle-Gruppierungen wie auch die einzelnen Bundles, entlang den entsprechenden Kurven angeordnet.

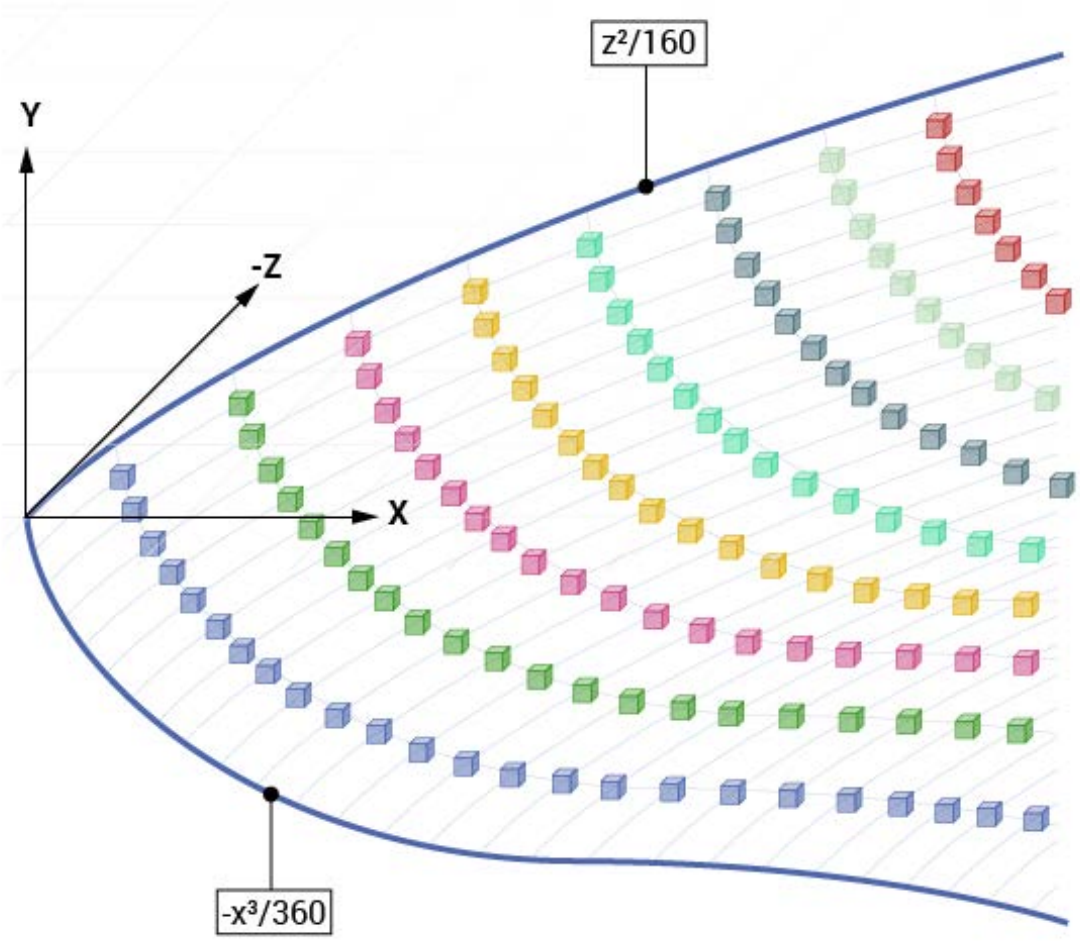

Abbildung 4.5: Bundles und Cluster in der gereihten Anordnung.

4.3.7 Abhängigkeiten von Bundles und Packages

Die Darstellung von Imports und Exports soll im Falle der Anzeige ohne Packages genauso funktionieren wie auch in der Ansicht mit Packages. Werden allein die Bundles angezeigt, werden Bundle zu Bundle Abhängigkeiten dargestellt. Ist die Darstellung der Packages aktiv, werden die Abhängigkeiten folgendermaßen gezeigt:

Imports: Das importierende Bundle wird mit allen exportierenden Packages verbunden.

Exports: Jedes Package des exportierenden Bundles wird mit den Bundles, welche diese Packages importieren, verbunden.

Für die Import-Export-Abhängigkeiten eignet sich die Node-Link-Darstellung, welche in Kapitel 3.1.1 detaillierter beschrieben ist. Die Knotenpunkte sind dabei die einzelnen Packages und Bundle-Elemente. Export- und Importverbindungen sollen farblich voneinander zu unterscheiden sein.

4.3.8 Darstellung der Klassen

Jede Klasse eines Packages soll als zylinderförmiges Objekt dargestellt werden. Das Anzeigen der Klassen soll nur dann möglich sein, wenn man sich in dem *Klassendarstellungsmodus* befindet, der für jedes einzelnes Bundle durch die Bundle-GUI aktiviert werden kann. Jedes Package kann transparent geschaltet werden um so das Innere des Packages zeigen zu können. Pro Klasse soll ein Zylinderobjekt erstellt werden. Die Größe des Zylinders bleibt immer gleich. Ziel ist zudem eine Darstellung, die die Klassenobjekte so wenig wie möglich entfernt von einander zeigt. Der Betrachter soll ohne mehrfaches wechseln der Perspektive möglichst viele Klassen betrachten können.

Bei Aktivierung des Events für die Klassendarstellung welche das Package-Objekt transparent schaltet, sollen alle Klassenobjekte der Reihe nach erstellt werden. Begonnen wird mit dem ersten Klassenobjekt im Mittelpunkt des (transparenten) Package-Objekts. Die weiteren Objekte werden spiralförmig aufwärts gereiht. Das bedeutet, dass jedes weitere Objekt auf einem Punkt auf einer Kreisumfangslinie positioniert wird, mit steigendem Radius und Gradzahl. Damit die Reihung der Klassen nicht mit anderen Objekten in der Umgebung kollidiert, soll ein Maximalradius definiert werden. Für die bessere Betrachtung von allen Seiten soll jedes Klassenobjekt in der Y-Achse gering versetzt werden (Abbildung 4.6).

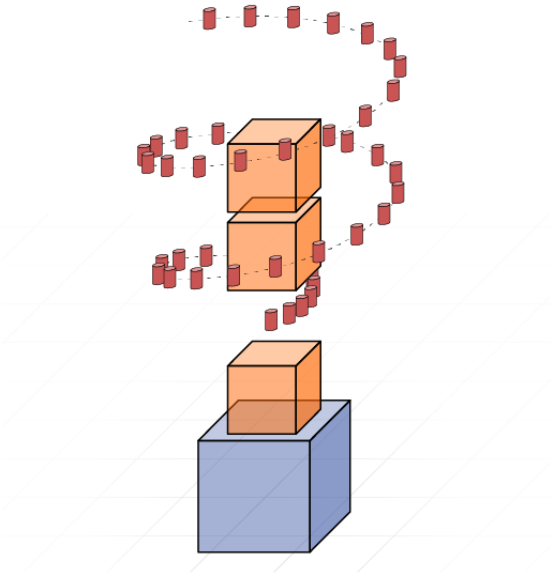

Abbildung 4.6: Anordnung der Klassen.

Der Name der Klasse wird anliegend an das Klassenobjekt mit halber Transparenz angezeigt um die Übersicht nicht zu sehr zu beeinträchtigen. Mit der Selektion des Klassenobjektes wird der Name ohne Transparenz angezeigt und somit hervorgehoben.

4.4 Interaktion und Ablauf des Programmes

4.4.1 Interaktion

Da vor allem beim Google Cardboard neben einem Trigger-Button (Touch auf dem Display, durch einen Trigger an der Brille auslösbar) keine weiteren Eingabeparameter vorhanden sind, soll eine Selektionsmöglichkeit mit Hilfe eines Anvisierungselements, einer Art Fadenkreuz oder hier Pointer genannt, zur Verfügung gestellt werden. Die Selektion von Elementen wird ermöglicht, wenn der Pointer eine gewisse Zeit auf das zu selektierenden Element zielt. Buttons sollen zusätzlich eine Animation zugewiesen bekommen, um die Dauer der Selektion erkenntlich zu machen.

4.4.2 Navigation und GUI

Während der Darstellung soll eine GUI (Graphical User Interface) angezeigt werden, die die Kontrolle der Bundle-Objekte und deren Layout erlaubt. Diese soll daher konstant zugreifbar sein, während sich der Benutzer innerhalb des Feldes umsieht. Da sich der Betrachter durch den Raum bewegen kann, ist es sinnvoll, eine GUI zur Verfügung zu stellen, die den Benutzer erlaubt von jeder Position aus mit ihr interagieren zu können. Eine sinnvolle Lösung ist daher, die GUI unter der Viewer-Kamera anzuhängen. Dadurch stört sie nicht beim Umschauen und lässt sich schnell mit dem Blick nach unten wiederfinden. Je nach Betrachtung sollen unterschiedliche GUIs zur Verfügung stehen. Elemente die wichtiger sind, bzw. mit denen mehrmals interagiert werden, sollen mehr im Zentrum stehen. Elemente die wenig gebraucht werden, wie zum Beispiel die Einstellung des VR-Modes, weniger und werden mehr außerhalb des direkten Blickfeldes positioniert [111]. Im Folgenden werden die einzelnen GUI -Teile beschrieben.

Settings-GUI

Die Einstellungen die sich mit diesem Teil der GUI modifizieren lassen, sollen nicht die Visualisierung direkt betreffen. Daher sind die folgenden Events in dieser GUI untergebracht (Abbildung 4.7):

- Durch die Aktivierung/Deaktivierung des *VR Modes* soll der Benutzer auswählen können ob das Display des Mobiltelefons für das Linke und Rechte Auge unterteilt wird, um mit einem Cardboard betrachtet werden zu können, oder ob die Darstellung ganz-flächig auf dem Display zu sehen ist, für die Darstellung ohne Cardboard.

- Durch den Klick des Buttons *Startposition* soll der Benutzer wieder an die Anfangsposition geführt werden.

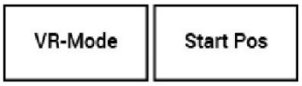

Abbildung 4.7: Buttons der Settings-GUI.

Haupt-GUI

Die hier untergebrachten Einstellungen betreffen die Ansicht und Darstellung der Bundles wie auch deren Layout. Folgende Events werden dieser GUI zugewiesen (Abbildung 4.8):

- Ein- und Ausblenden der Packages mit dem Button *Package On/Off*
- Arrangieren der Cluster nach der im Kapitel 4.3.5 beschriebenen quadratischen Anordnung, ausführbar durch den Button *Cluster*.
- Arrangieren der Cluster nach der im Kapitel 4.3.6 beschriebenen gereihten Anordnung. Ausführbar durch den Button *Cluster Lines*.
- *Top-View*-Button, der alle darzustellenden Objekte um 90 Grad dreht, so dass der Benutzer diese von Oben betrachten kann.
- Der Button *Switch Navigation-GUI/Bundle-GUI* soll die Benutzung der beiden im Folgenden beschriebenen GUIs aktivieren oder deaktivieren, so dass entweder nur die Bundle-GUI oder die Navigations-GUI aktiviert ist.

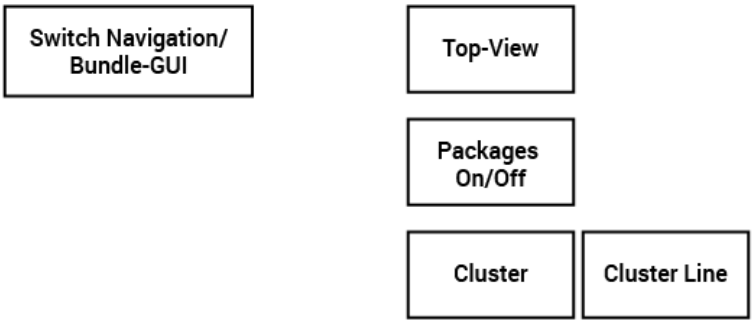

Abbildung 4.8: Buttons der Haupt-GUI.

Navigations-GUI

Die Interaktion durch Navigation durch den Raum soll ermöglicht werden, wenn sich der Benutzer nicht in dem Modus der näheren Untersuchung der Bundles befindet. Daher sollen die Funktionen dieser GUI nur aktiv sein wenn sich der Benutzer in der obersten Ebene befindet. Das bedeutet, dass die Bundle-GUI nicht aktiv ist. Ist der *Move in Scene* Button aktiviert, wird der Button *Rotate Objects* deakti-

viert und umgekehrt, somit ist der Ordnung wegen immer nur eine der beiden Funktionen ausführbar. Folgende Möglichkeiten der Navigation sollen hier ermöglicht werden (Abbildung 4.9):

- Der Button *Move in Scene* aktiviert einen weiteren Button *Move,* welcher sich in der Mitte des Sichtfeldes befindet. Wird dieser selektiert beginnt sich der Viewer mit einer vorgegebenen Geschwindigkeit durch den Raum zu bewegen. Die Bewegungsrichtung ändert sich konstant mit der Blickrichtung. Somit kann während der Fahrt die Richtung durch die Bewegung des Kopfes geändert werden. Eine weitere Selektion des *Move* Buttons stoppt die Bewegung.
- Der Button *Rotate Objects* ermöglicht die Rotation aller Objekte im Raum. Durch ein Rotationskreuz welches nach dem Auslösen des Buttons angezeigt wird, können sich je nach anvisiertem Bereich des Steuerkreuzes alle Objekte in einer vorgegebenen Geschwindigkeit drehen. Dieses Steuerkreuz ist in 6 Abschnitte unterteilt. Drei Abschnitte für die Rotation um die X-, Y- und Z-Achse im Uhrzeigersinn, sowie drei Abschnitte für die Rotation um diese Achsen gegen den Uhrzeigersinn. Solange sich der Pointer über einer dieser Abschnitte befindet, bewegen sich die Objekte entsprechend, bis der Pointer den Abschnitt des Rotationskreuzes verlässt.

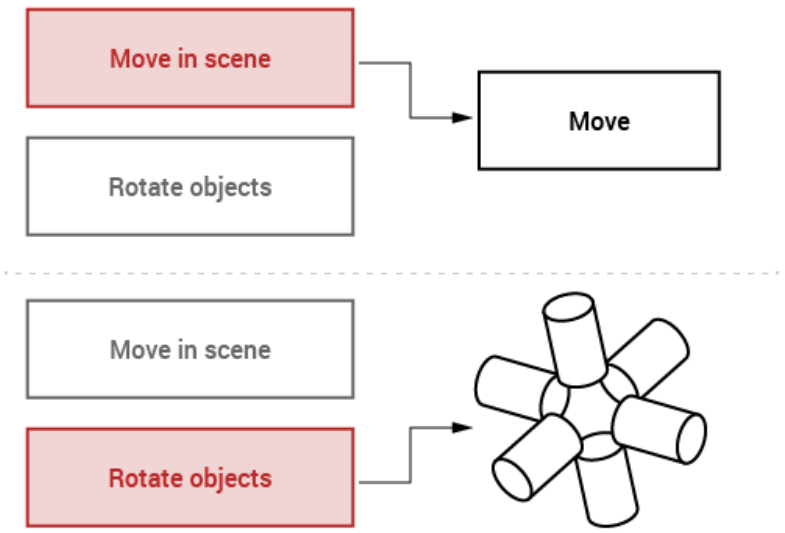

Abbildung 4.9: Buttons der Move-GUI. Oben dargestellt ist die Ansicht, wenn *Move in scene* aktiviert wurde. Unten dargestellt ist das Rotationskreuz, welches erscheint wenn *Rotate Objects* aktiviert wurde.

Bundle-GUI

Neben den Interaktionen die die komplette Darstellung betreffen, soll es auch für einzelne Bundles die Möglichkeit geben, die Repräsentation zu modifizieren. Hierfür soll eine dem Bundle zugewiesene GUI hinzugefügt werden, welche bei der Auswahl eines Bundles erscheint (Abbildung 4.10).

- Das Ein- und Ausblenden der Packages des im Moment aktivierten Bundles kann mit einem Button-Klick ausgeführt werden.
- Anzeigen der Importabhängigkeiten werden mit einem weiteren Button angezeigt oder deaktiviert. Die Packages aller Bundles in dem Projekt werden somit mit einer Linie mit dem im Moment aktivierten Bundle angezeigt. Sollte die Packages-Ansicht deaktiviert sein, wird nur die Verbindung von Bundle zu Bundle dargestellt.
- Das Anzeigen der Exports geschieht nach einem ähnlichen Prinzip wie das Anzeigen der Imports. Linien werden hier andersfarbig gekennzeichnet und zeigen von allen exportierten Packages des aktuell bearbeiteten Bundles zu allen Bundles zu denen exportiert wird.
- *Go to Bundle / Package View* soll die Untersuchung eines einzelnem Bundles ermöglichen. Mit der Selektion des Buttons wird der Viewer besonders nah an das Bundle geführt um dessen Packages weiter zu untersuchen. Daher wird in diesem Zustand auch eine weitere GUI aktiv, welche Interaktionen mit dem Package ermöglichen. Diese GUI wird im Teil *Package GUI* genauer beschrieben.

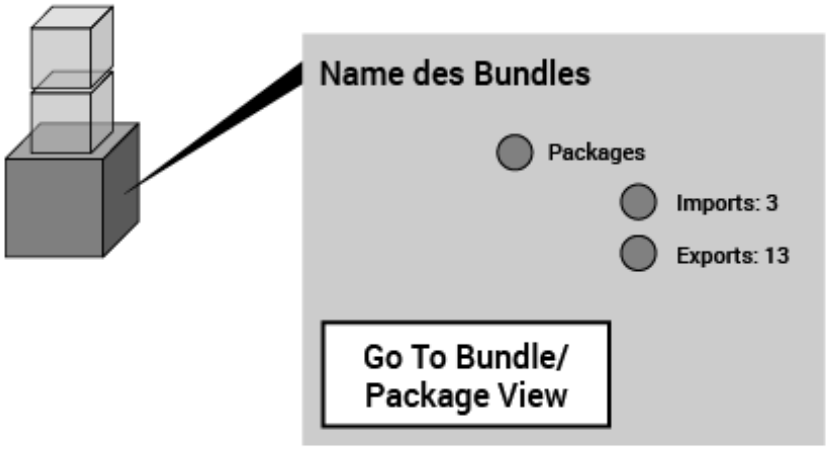

Abbildung 4.10: Bundle-GUI.

Package-GUI

Wurde nach der Betrachtung eines Bundles der Button für die Untersuchung der Packages (*Go to Bundle / Package View*) aktiviert, hat der Betrachter die Möglich-

keit weitere Funktionen die die Darstellung und Modifizierung einzelner Packages betreffen, auszuführen (Abbildung 4.11). Ein Textelement zeigt während der Dauer dieses Zustandes den Bundle-Namen an. Des Weiteren sind folgende Funktionen ausführbar:

- Mit dem Betreten des Packages mit dem Pointer soll das Package genauer untersucht werden. Dadurch wird es transparent und alle in dem Package befindlichen Klassen kommen zum Vorschein. Die Anordnung dieser ist in Kapitel 4.3.8 genauer beschrieben. Durch die Transparenz des Package-Objektes wird auch ein weiteres, kleines würfelförmiges Objekt, welches sich im Inneren eines jeden Packages befindet, sichtbar. Dieses dient als Button, welcher bei Selektion das Package in seinen Ausgangszustand zurückführt.
- Da die Bewegung durch den Raum nur bei Betrachtung der kompletten Ansicht möglich ist und somit hier nicht zur Verfügung steht, soll hier auf eine spezifischere Art die Bewegung durch den Raum ermöglicht werden. Die vorgegebene Nähe ist eine Voraussetzung für das Erkennen aller Objekte innerhalb der Packages und kann daher nicht verändert werden. Die Höhe des Betrachters lässt sich mit der Nutzung der Buttons *Go Up* und *Go Down* verändern. Die Bewegung wird solange ausgeführt wie sich der Pointer über dem entsprechenden Button befindet.
- Durch die Auswahl des Buttons *Leave Package View* verlässt der Benutzer die Ebene der Package-Bearbeitung und kann wieder zwischen Navigation oder Betrachtung der Bundles wählen.

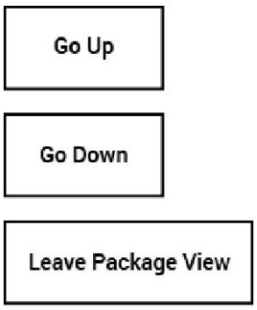

Abbildung 4.11: Buttons der Package-GUI.

Einstellungen durch Kopfbewegungen

Neben der Auswahl verschiedener Events durch die GUI sollen einige Einstellungen mithilfe der Kopfbewegung ermöglicht werden. Der Neigungswinkel des Kopfes wird erfasst sobald dieser sich mehr als 25 Grad nach links oder rechts neigt.

- Die Neigung von über 25 Grad zur linken Seite bewirkt das Rearrangieren der GUI in Blickrichtung des Nutzers. Ursprünglich befindet diese sich unterhalb-mittig der Blickrichtung in der Startposition. Sollte sich der Betrachter um 180 Grad drehen, wäre die GUI nicht mehr zu sehen. Durch die Kopfneigung wird es ermöglicht mit den Objekten via GUI zu interagieren auch wenn sich die Blickrichtung während der Betrachtung verändert.
- Die Neigung von über 25 Grad zur rechten Seite bewirkt das An- und Ausschalten der Bundle-GUI, welche immer erscheint, wenn der Pointer auf ein Package zeigt. Da diese GUI relativ oft erscheint und dies auch teilweise ungewollt während des Umsehens passieren kann, soll hiermit die Übersicht erleichtert werden wenn keine genaueren Informationen zu einem Bundle gewünscht sind.

5 Umsetzung

5.1 Verwendete Technologien

5.1.1 Hardware

Um eine VR-Applikation zu erstellen, ist die Verwendung einer geeigneten VR-Hardwarekomponente unverzichtbar. Da das Gefühl vermittelt werden sollte sich direkt in dem dargestellten Raum zu befinden, eignet sich die Nutzung einer VR-Brille. Aufgrund der Zielgruppe welche in Kapitel 4.1 näher definiert ist, ist es sinnvoll eine einfach zugängliche, kostengünstige Variante zur Verfügung zu stellen, um die Visualisierung so zugänglicher zu machen. Dafür bietet sich die Verwendung der *Google Cardboard* Brille an. Mit ihr lassen sich VR-Applikationen darstellen, indem ein Smartphone in sie eingelegt wird. Unabhängig von der Brille lassen sich die Projekte auch ohne VR-Modus darstellen. Das bedeutet, dass die Anwendung auch funktioniert, indem das Display nicht in linkes und rechtes Auge unterteilt wird, sondern die App über das komplette Display läuft. Dies ermöglicht die Interaktion durch Bewegung im Raum ohne zusätzliche Hardware, ausgenommen dem Smartphone, auf dem die Anwendung läuft.

5.1.2 Software

Umgesetzt wird das Projekt mit der *Unity Game Engine* [105]. Besonders vorteilhaft ist, dass Unity eine SDK für Google Cardboard zur Verfügung stellt [108]. Eine weitere Möglichkeit wäre das Projekt in *OpenG*L umzusetzen. Dies bietet zwar viel mehr Möglichkeiten, doch ein Nachteil ist, dass es weitaus komplexer und zeitintensiver ist um zu einem Ergebnis zu kommen.

5.2 Extraktion der Daten

Bevor mit der Darstellung begonnen werden kann, müssen alle Daten dafür vorhanden sein. Die Extraktion der Daten soll mit einem eigenen Programm umgesetzt werden, welches nur die Infos zusammenstellen soll, welche für die Visualisierung wichtig sind. Das Programm liest bestimmte Daten des OSGi-Softwareprojektes aus und schreibt diese in eine neue Text-Datei im JSON-Format. Um an die Daten zu gelangen gibt es die Möglichkeit das Repository auszulesen, welche im Folgenden erläutert wird.

Auslesen der Programm-Struktur: Die Ordnerstruktur eines OSGi-Projektes sagt einiges über das Projekt aus, da OSGi auf gewisse Konventionen aufbaut, die eingehalten werden müssen [100]. Bundles sind alle Ordner innerhalb des Main-Ordners die den Ordner „META-INF" enthalten, der die Datei „MANIFEST.MF enthält.

Auslesen der Manifest-Dateien: Jedes Bundle hat eine Manifest-Datei, die Information über das entsprechende Bundle enthält. Durch das Auslesen dieser Dateien lassen sich importierte Bundles extrahieren, Packages, die exportiert werden, ermitteln, der symbolischen Namen des Bundles auslesen, Version des Bundles erkennen und vieles mehr. Durch die hier gewonnen Daten lassen sich die Bundles und deren Abhängigkeiten darstellen.

Auslesen der XML-Dateien: Um an die Daten der Services zu gelangen, müssen die entsprechende XML-Dateien ausgelesen werden. Ein Bundle, das einen Service zur Verfügung stellt, besitzt den Ordner „OSGI-INF" welcher die XML-Daten enthält.

Eine weitere Möglichkeit auf die in dieser Arbeit zurückgegriffen wird ist das Auslesen einer bereits vorhandenen Model-Datei. Das vom DLR umgesetzte Tool *Extraktion und Visualisierung von Beziehungen und Abhängigkeiten zwischen Komponenten großer Softwareprojekte* bietet bereits ein Analyse- und Extraktionstool welches eine JSON-Datei durch unter anderem Auslesen des OSGi-Projekt-Repository schreibt [18] [19]. Diese Datei direkt im Visualisierungsprojekt zu nutzen wäre allerdings nicht umsetzbar, da sie bei etwas größeren OSGi-Projekten einen zu großen Umfang besitzt und das Auslesen somit problematisch wird. Aus diesem Grund soll diese *Model-Datei* mit einem externen Programm ausgelesen werden welches dann erneut eine Datei schreibt, welche nur die für die Visualisierung benötigten Daten beinhaltet. Abbildung 5.1 zeigt einen Teil des ersten Bundles des RCE-Projektes in der zur Verfügung stehenden JSON-Datei.

Abbildung 5.1: Teil der zur Verfügung stehenden JSON-Datei.

Das Auslesen der Daten aus der zur Verfügung stehenden Model-Datei im JSON-Format, sowie das Schreiben der Daten in das neue Dokument, wird mit Hilfe von *LitJson* umgesetzt. LitJson ist eine Library für C# für das schnelle und einfache Handhaben von JSON-Dateien [106]. Um auch größere Datensätze auslesen zu können, wird die Datei asynchron ausgelesen da das Lesen besonders dann eine gewisse Zeit in Anspruch nehmen kann. Während die Datei ausgelesen wird, werden alle Daten, die für die Visualisierung relevant sind, neu sortiert und in Listen aufgeteilt (Abbildung 5.3). Mit der neuen Anordnung werden sie wieder in ein neues Dokument geschrieben. Abbildung 5.2 zeigt das erste Bundle mit dessen Daten zu Imports, Exports, Packages und deren Klassen und Services, den Namen

des Bundles sowie den symbolischen Namen des Bundles, nachdem diese Daten aus der vorhandenen Datei ausgelesen und neu geschrieben wurden.


```
▼ array [139]
  ▼ 0  {5}
      BundleName : RCE Cluster Component Common
      BundleSymbolicName : de.rcenvironment.components.cluster.common
    ▼ Packages [1]
      ▼ 0  {3}
          PackageName : de.rcenvironment.components.cluster.common
          ProvidedServices : null
        ▼ Classes [1]
            0  : de.rcenvironment.components.cluster.common.ClusterComponentConstants
    ▼ Imports [3]
        0  : de.rcenvironment.core.utils.common.channel.legacy
        1  : de.rcenvironment.core.utils.common
        2  : de.rcenvironment.core.component.api
    ▼ Exports [1]
        0  : de.rcenvironment.components.cluster.common
```


Abbildung 5.2: Teil der erstellten JSON-Datei.

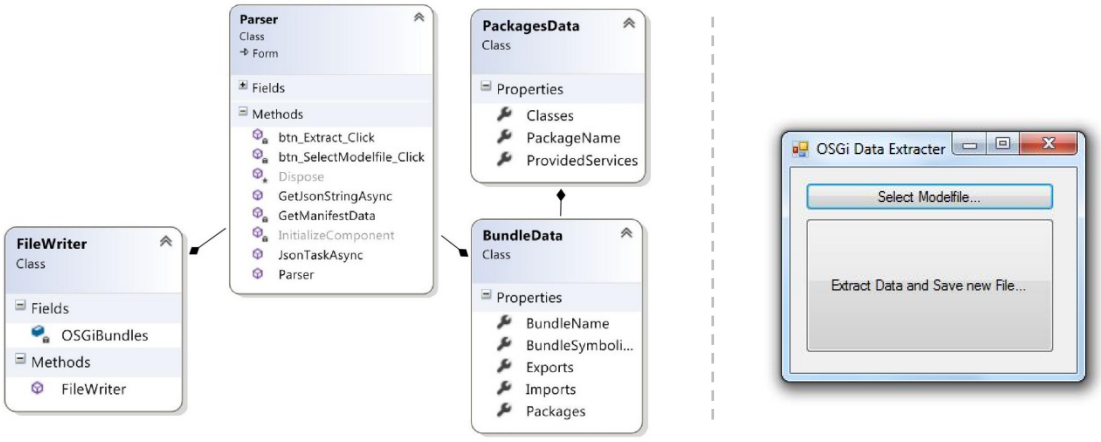

Abbildung 5.3: Klassendiagramm und Oberfläche des Programmes für das Extrahieren der OSGi-Daten aus der Model-File.

5.3 Die Visualisierung

5.3.1 Aufbau der Objekte

Um die Visualisierung der Softwarearchitektur effizient aufzubauen und in Echtzeit lauffähig zu gestalten werden für die Darstellung einzelner Objekte möglichst einfache Formen verwendet. Bundles und Packages haben die Form eines Würfels, auf den ein Bild gelegt wird das alle 6 Seiten eines Würfels zeigt. Durch das Einset-

zen von solchen Texturen als Bilddatei im .jpg oder .png Format kann einiges an Renderzeit eingespart werden, da die Grundform des 3D-Objekts einfach bleibt, und gleichzeitig grafische Details oder Verläufe durch den Inhalt der Textur gezeigt werden. Services werden als Scheiben dargestellt und Klassen als Zylinder, um diese von den anderen Objekten einfacher zu unterscheiden. Neben der Form soll auch Farbe helfen, die Unterscheidung noch einfacher zu gestalten. Unity ermöglicht es die Objekte als Vorlage im *Ressources* Ordner aufzurufen und diese in der Szene zu Initialisieren. Dies ermöglicht es die Objekte vorab in Unity zu entwickeln und während der Laufzeit beliebig oft in der Szene zu duplizieren und zu modifizieren. Die Abbildung 5.3 zeigt das Bundle-Objekt und dessen Komponenten auf die im Unity-Editor per Script zugegriffen werden kann.

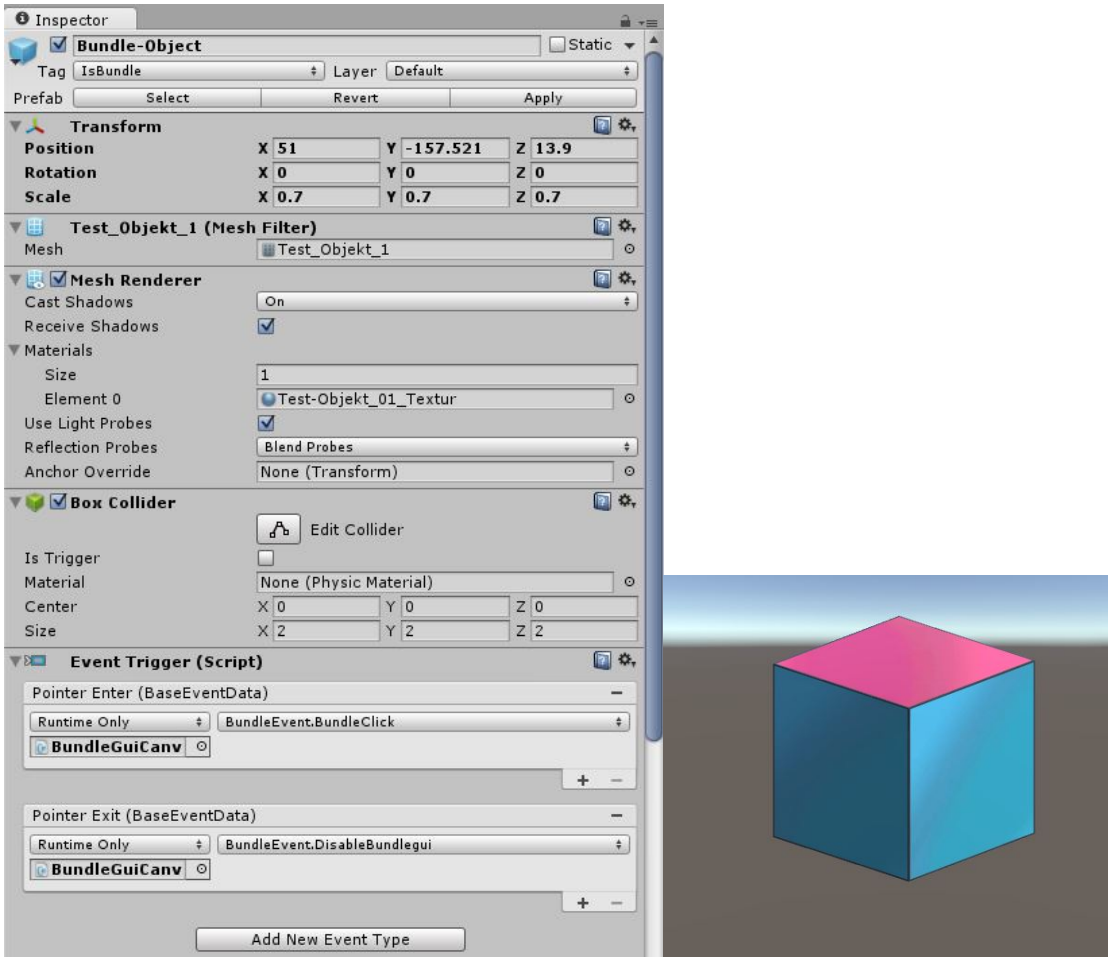

Abbildung 5.4: Das Vorlage-GameObject für das Bundle und dessen Komponenten.

Auf die Möglichkeit des Ladens der Vorlageobjekte aus dem Ressources-Ordner wird während des Initialisierungsvorganges zurückgegriffen. In Abbildung 5.4 werden die einzelnen Vorlageobjekte gezeigt.

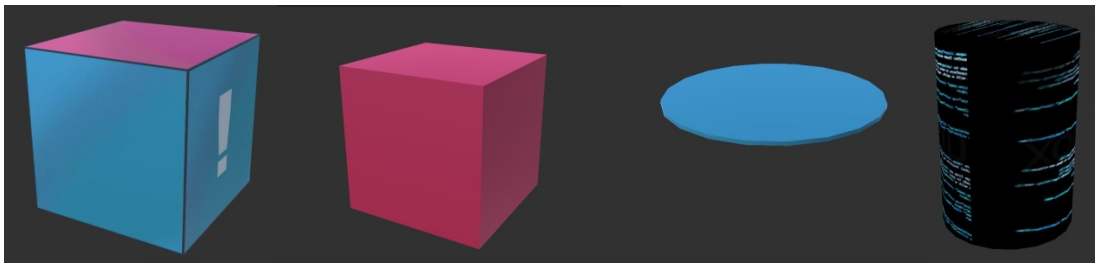

Abbildung 5.5: Bundle-Objekt, Package-Objekt, Service-Objekt und Klassen-Objekt.

5.3.2 Initialisierung

Alle Komponenten des Programms werden im Initialisierungsvorgang geladen. Da das bei komplexeren Projekten eine gewisse Zeit in Anspruch nehmen kann, soll der Programmaufbau geschehen, während sich der Nutzer bereits in der erstellenden Szene umsehen kann. Die GUI wird erst nach dem Initialisierungsvorgang zu Verfügung stehen, da eine Interaktion erst möglich sein soll, wenn alle Komponenten geladen sind.

Mit Hilfe von *Coroutines*, die Unity zur Verfügung stellt, kann es ermöglicht werden, die Objekte aufzubauen während der Betrachter in der Szene interagieren kann [107]. Innerhalb der Coroutine-Funktion vom Typ *IEnumerator* können an beliebigen Stellen *Return*-Werte eingesetzt werden, welche es möglich machen, den bisher ausgeführten Teil der Funktion zurückzugeben bevor diese komplett beendet wurde. *Yield return null* rendert die Szene bis zu diesem Punkt. Weitere Möglichkeiten wären unter anderem auch das Stoppen der Funktion für eine bestimmte Zeit mit yield *return new WaitForSeconds(float)*, welche in diesem Projekt vor allem bei den Events genutzt wird, da diese nach Selektion eine gewisse Zeit warten, bis das Event ausgeführt wird. Während des Wartens wird parallel eine weitere Funktion ausgeführt, die die Anzeige eines Ladebalkens mit gleicher Dauer innerhalb des Buttons ausführt. Abbildung 5.5 zeigt die Rekonfigurierung der GUI-Position anhand der Kamerablickrichtung, umgesetzt mithilfe der Coroutines.

```
IEnumerator ReconfigureGuiWait()
{
    yield return new WaitForSeconds(2);
    GameObject mainGui = GameObject.Find("MainEventsCanvas");
    mainGui.transform.position = Camera.main.transform.forward;
    mainGui.transform.rotation = Camera.main.transform.rotation;
}
```

Abbildung 5.6: Codeausschnitt für die Rekonfigurierung der GUI.

5.3.3 Aufbau der Komponenten

Jede Komponentenart, das bedeutet Bundles, Packages oder Services, sollen unabhängig voneinander geladen werden. Daher werden sie in einer gewissen Reihenfolge aufgebaut indem eine Komponentenart erst geladen wird wenn die vorherige mit dem Laden abgeschlossen hat. In diesem Projekt bedeutet das, dass erst die Bundles, dann die Packages, gefolgt von dem LineRenderer für die Darstellung der Abhängigkeiten, und dann die Services geladen werden. Ausnahme sind die Klassenobjekte, welche erst während der Interaktion mit einem Package geladen werden. Alle anderen Objekte wie auch alle Daten werden vorab geladen. Das bedeutet zwar einen längeren Initialisierungsvorgang, dafür stehen die Daten dann während der Laufzeit konstant zur Verfügung und ein Zugriff ist ohne Verzögerung möglich.

Abbildung 5.6 zeigt den Aufbau des Bundles und nach Abschluss davon den Aufbau der Packages. Es folgt der Aufbau der Services. Ist dieser Vorgang beendet, sind die Objekte wie in Abbildung 5.7 dargestellt.

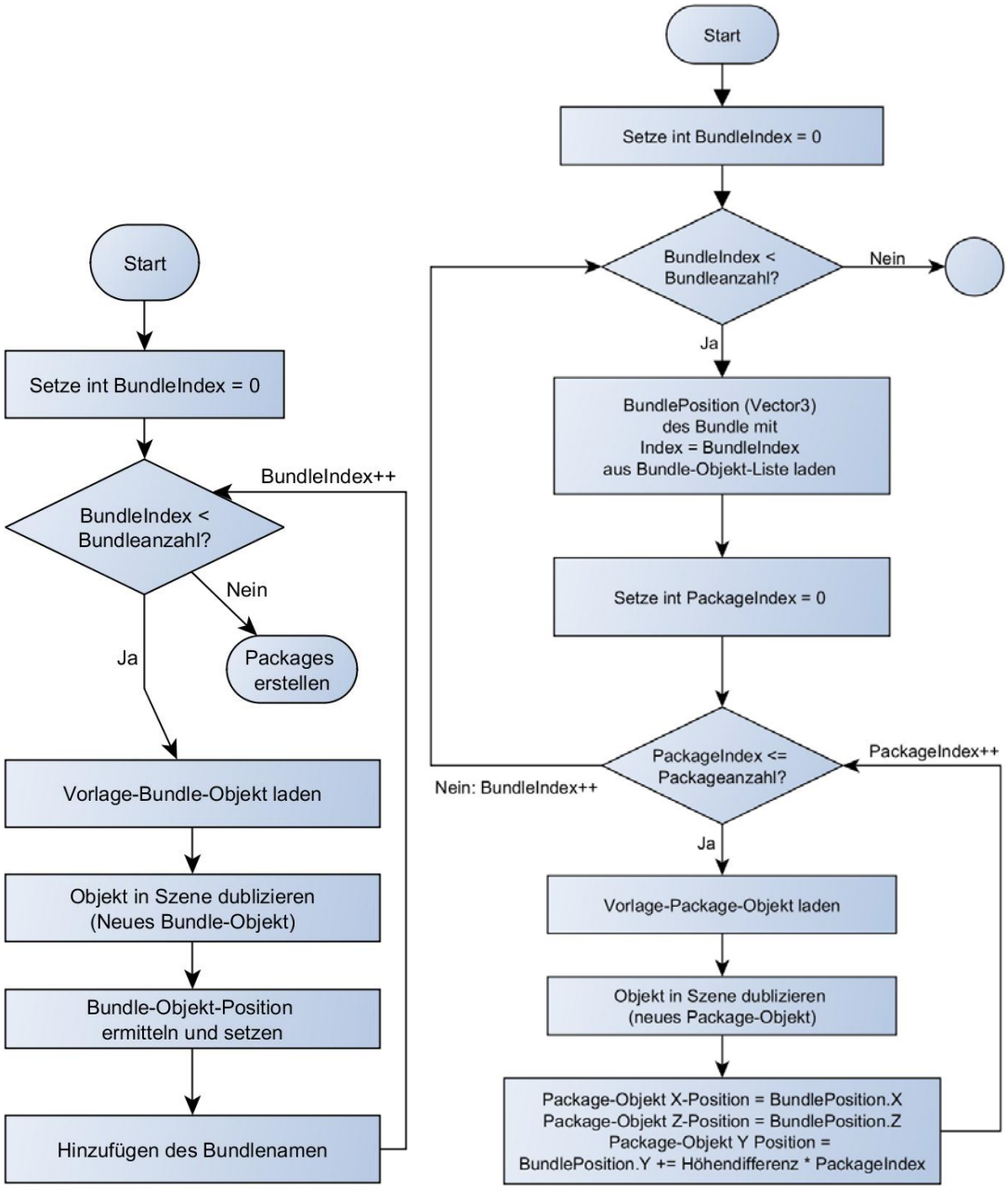

Abbildung 5.7: Aufbau der Bundles (links) und Packages (rechts).

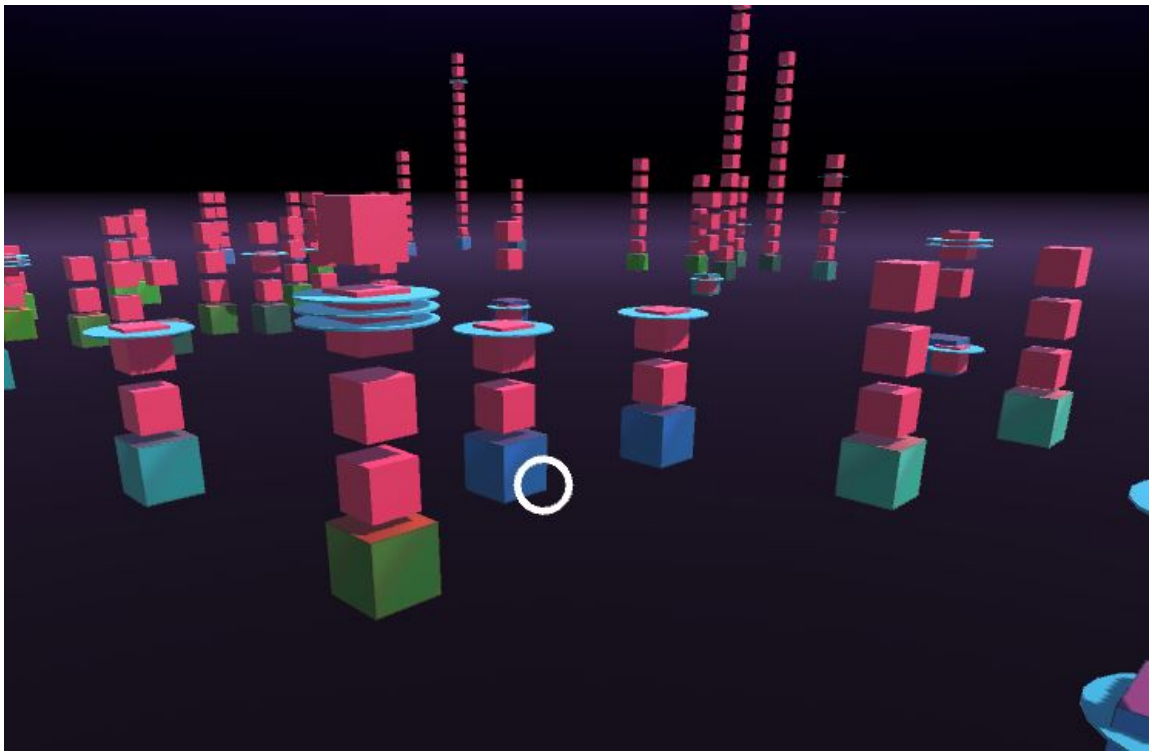

Abbildung 5.8: Screenshot der Visualisierung mit Bundle-Objekten, Package-Objekten und Services als Scheiben innerhalb der Package-Objekte.

Das Laden der Imports geschieht nach dem Generieren der Bundles, Packages und Services. In der JSON Vorlage-Datei befinden sich zu jedem Bundle-Objekt die „Imports“ welche alle Packages-Namen aufgelistet enthalten. Diese werden ausgelesen. Da die Packages-Objekte in der Darstellung den Namen des zugehörigen Packages tragen, wird dieser in dem Projekt gesucht und dessen Position gemerkt. Dieser Vorgang geschieht Bundle für Bundle, so dass jedem Bundle eine Liste bereitgestellt wird, welche die GameObjects enthalten die die Imports darstellen. Der Datentyp GameObject stellt ein Objekt in der Szene dar. Durch den Aufruf des Objektes können alle dem Objekt zugehörigen Komponenten aufgerufen werden. Das bedeutet, dass die GameObject Liste eines Bundles viele Informationen des Import-Objekts bereitstellt, die bei späteren Interaktionen aufgerufen werden können.

Das Laden der Exports geschieht auf ähnlicher Weise wie das Laden der Imports, jedoch ist es um einiges umfangreicher, deren Objekte zusammenzuführen. Die JSON-Datei stellt eine Liste zur Verfügung in der alle exportierten Packages des jeweiligen Bundles aufgelistet werden. Allerdings gibt dies keine Information über das Ziel-Bundle des Exports. Auch kann ein Export mehrere Ziel-Bundles zugeteilt bekommen haben, was auch nicht aus einer Liste der zu exportierenden Package-Namen ersichtlich wird. Aufgrund dieser Problematik muss jeder Name der Export-

liste mit allen Bundles und deren Importliste verglichen werden. Jeder Export wird in einem anderen Bundle als Import gekennzeichnet. Das Bundle, welches den Import enthält, ist das Ziel-Bundle. Enthalten mehrere Bundles den Export als Import, so wird das Package mehrfach exportiert.

5.3.4 Erstellung der Objekte für Abhängigkeiten

Nachdem die Import- und Exportdaten eines jeden Bundles in Form von GameObjects bereitstehen, werden die *LineRenderer*, eine optionale Komponente eines GameObjects welches eine Linie darstellen kann, generiert. Bundles importieren: So ist die Anzahl der LineRenderer eines Bundles so hoch wie die Zahl der Imports des Bundles. Packages werden exportiert: Die Anzahl der LineRenderer der Exports ist somit gleich groß wie die Anzahl der Exports des jeweiligen Packages.

5.3.5 Laden der Klassenobjekte

Die Klassen sind die einzigen Objekte, die nicht schon im Initialisierungsvorgang geladen werden. Grund hierfür ist die besonders hohe Anzahl an Objekten die dadurch generiert werden würde und das Programm dadurch beeinträchtigen könnten. Die Klassen können nur geladen werden wenn der Benutzer ein bestimmtes Bundle selektiert hat und somit das Package selektierbar ist. Dadurch wird verhindert dass sich zu viele Objekte in der Szene befinden, denn die Klassenobjekte werden wieder gelöscht, wenn sich der Benutzer von dem Bundle entfernt, indem er die Bundle-Selektion verlässt. Das bedeutet das maximal alle Klassen eines einzigen Bundles parallel in der Szene vorzufinden sind. Abbildung 5.8 zeigt die Klassen eines Packages in der in Kapitel 4.3.8 beschriebenen Anordnung.

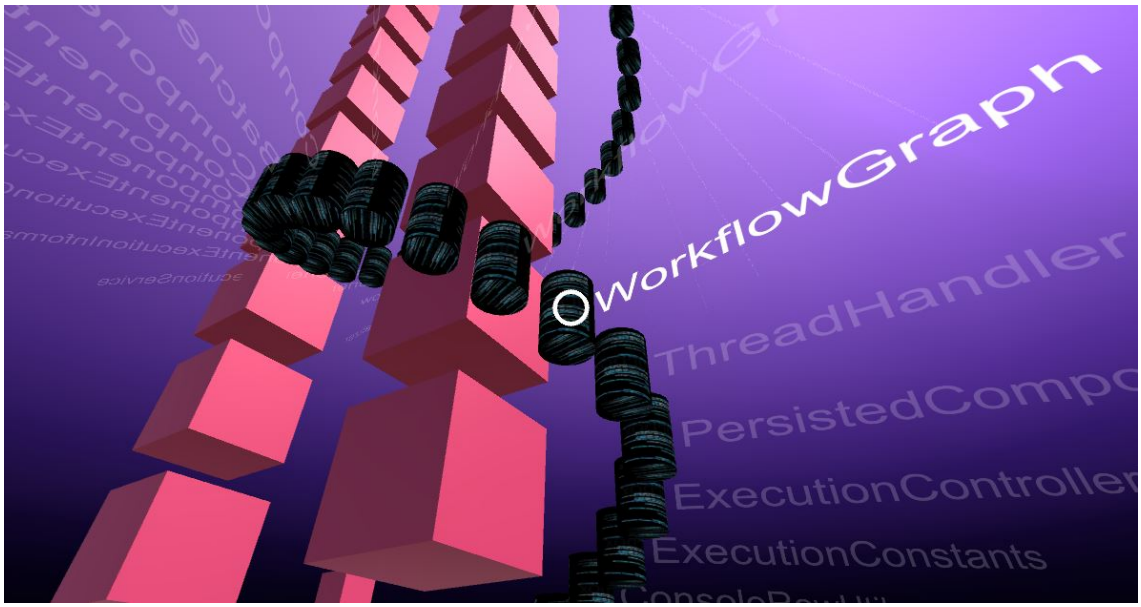

Abbildung 5.9: Klassen eines Packages.

5.3.6 Aufbau der GUI

Die GUI lässt sich aus den von Unity zur Verfügung gestellten *User Interface (UI) Objekten* aufbauen. Diese Objekte können Buttons, Checkboxen, Textfelder oder ähnliches sein. Die aus diesen Objekten erstellte GUI wird im unteren Teil des Kamerasichtfelds der *Main Camera* positioniert und vorerst ausgeblendet. Ist der Initialisierungsvorgang beendet, wird die GUI angezeigt, indem der Renderer für dieses Objekt auf *enabled* gesetzt wird. Jedem Button wird ein Script zugewiesen, welches die Funktion für die Animation des Ladebalkens zur Verfügung stellt. Zeigt der Pointer auf den Button, wird die Ladeanimation gestartet und nach ihrer Beendigung die dem Button entsprechende Funktion ausgeführt (Abbildung 5.10). Verlässt der Pointer vor Beendigung der Ladeanimation den Button, wird die Funktion gestoppt, ebenso wie die Animation. Somit kann gewährleistet werden, dass die entsprechende Funktion nur dann ausgeführt wird wenn der Betrachter eine gewisse Zeit den Pointer über den Button „zielt“ und nicht ungewollt Funktionen ausgeführt werden. Ein „Klick“ ist daher nicht notwendig und der Betrachter kann allein per Kopfbewegung mit der GUI interagieren. Ein weiteres Umsetzungsbeispiel der GUI wird in Abbildung 5.9 gezeigt. Zu sehen ist die Bundle-GUI mit runden Buttons und ein Textfeld Für den Bundle-Name sowie weitere Textfelder für die Anzahl der Imports und Exports.

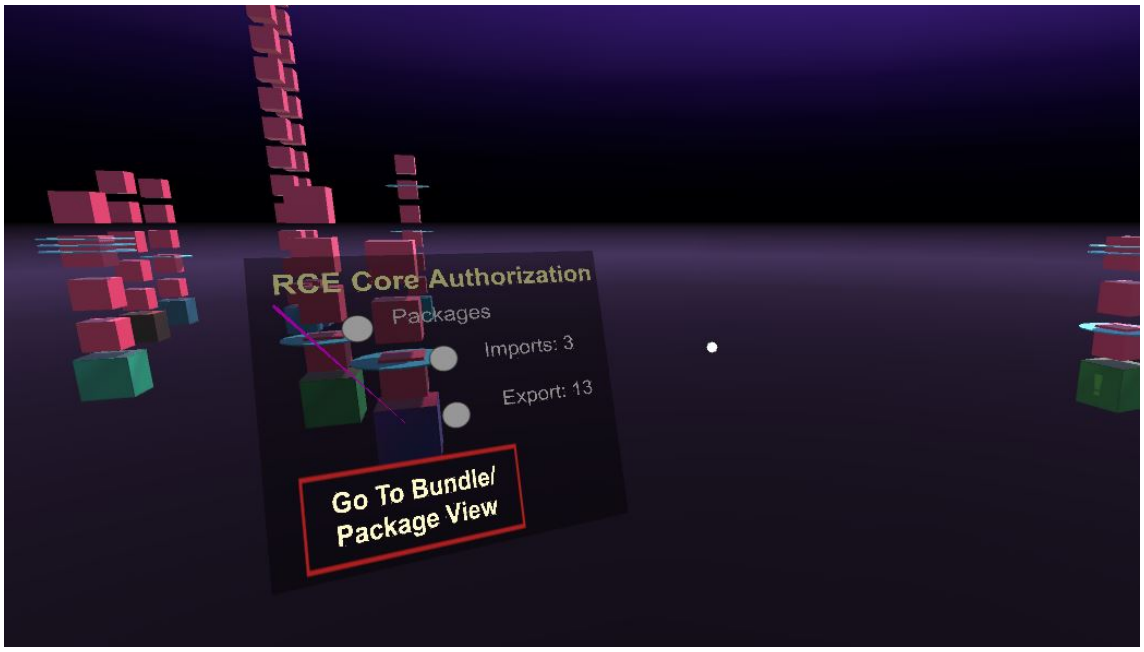

Abbildung 5.10: Bundle-GUI.

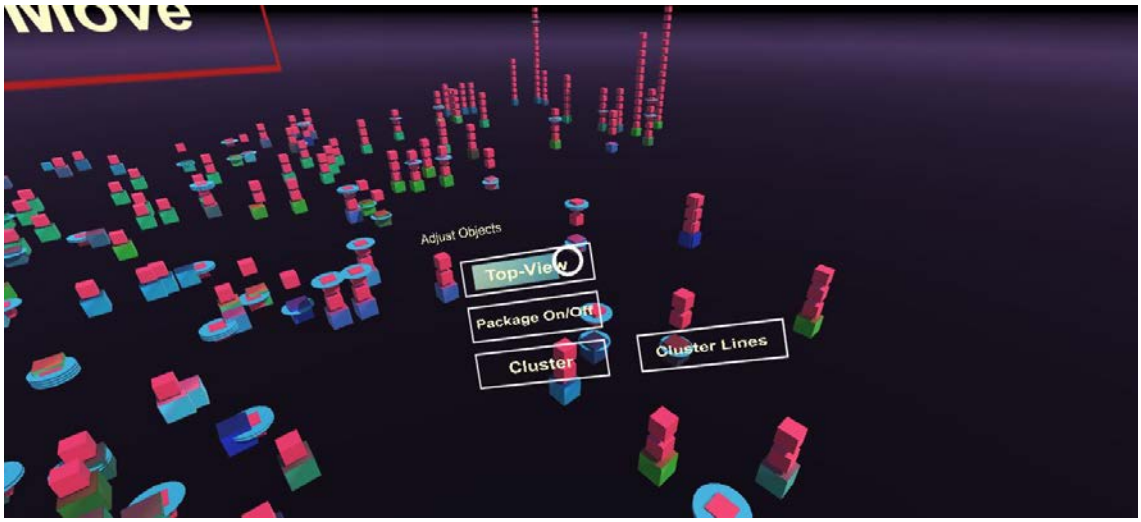

Abbildung 5.11: Button mit Lade-Animation.

5.3.7 Clusterung

Für die Ausführung der Clusterung, welche durch den in Kapitel 4.3.4 beschriebenen Namensvergleich der Bundles geschieht, werden alle Bundles in Gruppen aufgeteilt. Der Name des Bundle-GameObjects in der Szene ist gleich dem symbolischen Namen des repräsentierten Bundles. Die Tiefe des Namensvergleich gibt vor bis zu welchen „.“ der vordere Teil des Strings verglichen wird. Alle Bundles mit Übereinstimmung des Namens bis zu diesem Teil werden in einer eigenen Liste gespeichert. Nachdem jedes Bundle überprüft und zugewiesen wurde gibt es genauso viele Listen wie Cluster. Alle GameObjects der Liste werden nun einem Leeren GameObject untergeordnet, welches als Container dient. Der Name des Containers enthält den Namen des Clusters, also der übereinstimmende Teil der symbolischen Namen.

Sobald die Bundle-GameObjects ihren Containern zugeordnet wurden, wie in Abbildung 5.11 rechts dargestellt, werden zum einem alle Container positioniert, sowie die Bundles innerhalb des Containers. Links zu sehen ist die Clusterung in quadratischer Ansicht.

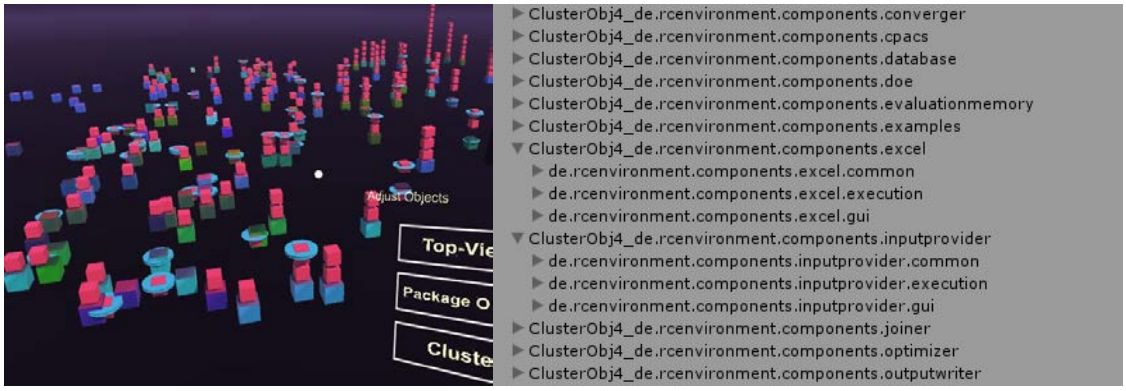

Abbildung 5.12: Gruppierung der Bundles in quadratischer Ansicht und die Bundles in ihren Cluster-Containern.

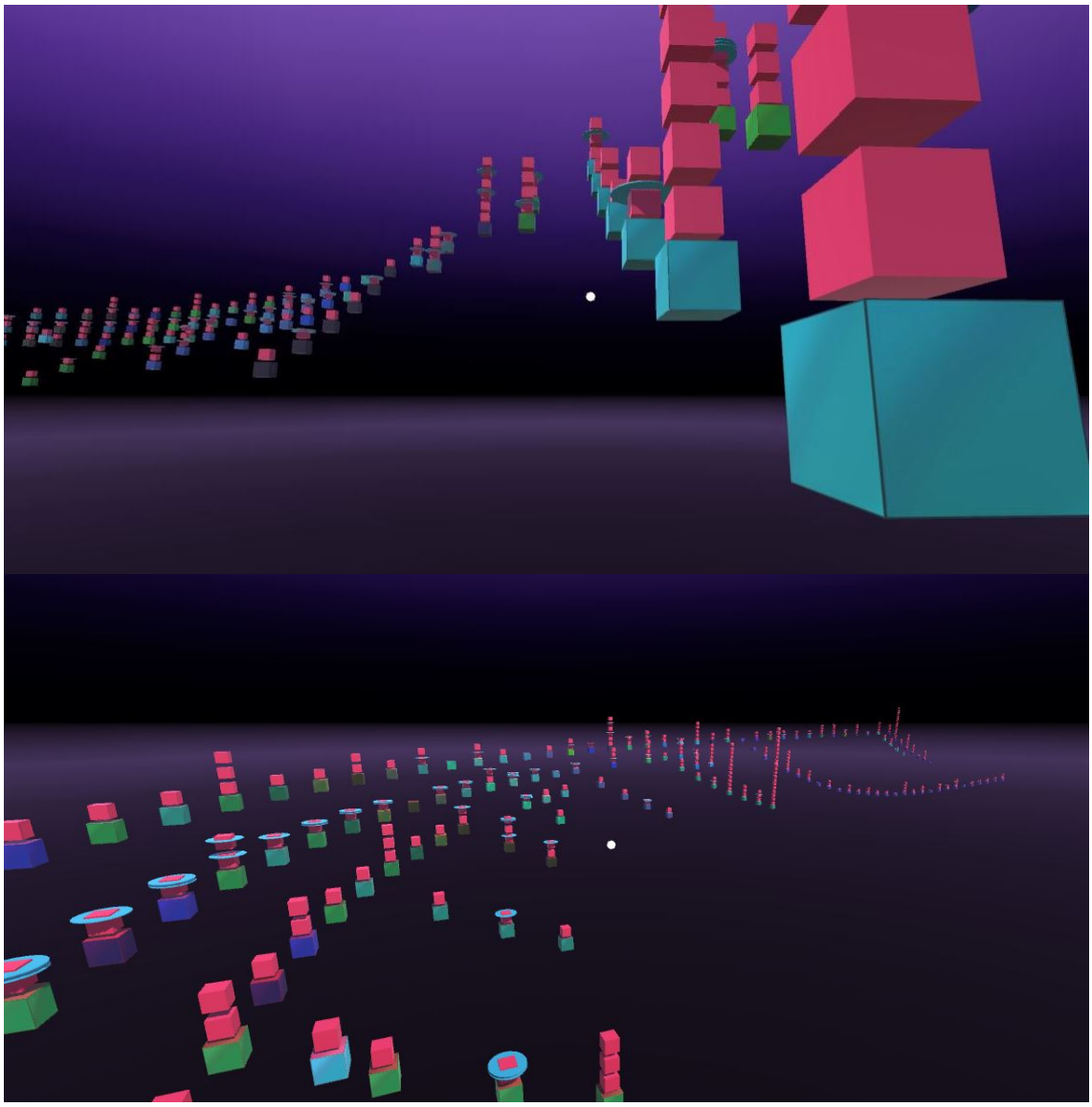

Abbildung 5.13: Gruppierung der Bundles in gereihter Ansicht.

Die Zuordnung der Bundles zu den Containern bleibt während der Laufzeit gleich. Nur die Position der einzelnen Bundles wie auch die der Cluster in der Szene verändert sich, wenn die Objekte zum Beispiel wie in Abbildung 5.12 gezeigt in die gereihte Ansicht wechseln.

5.3.8 Package-Darstellung An/Aus

Die Anzeige aller Packages wird hier aktiviert und deaktiviert. Jedes Package besitzt ein *Tag,* ein String das dem GameObject hinzugefügt werden kann. Wird das Event ausgelöst, werden alle Renderkomponenten welche definieren ob das Objekt zu sehen ist oder nicht mit dem Tag „*IsPackage*“ aktiviert wie auch die der zugehörigen Service-Objekte mit dem Tag „*IsService*“, falls sie vorher deaktiviert waren. Oder deaktiviert, falls sie vorher aktiviert waren.

5.3.9 VR Mode

Eine von *GoogleVR* bereitgestellte Funktion ermöglicht das An- und Ausschalten des VR-Modes. Diese wird aufgerufen und gesetzt wenn das Event des Buttons ausgelöst wurde. Die Aufteilung des Bildschirms für die Betrachtung auf dem Cardboard ist vorgegeben (Abbildung 5.13). Mit dem Hinzufügen eines von Google bereitgestellten Skripts zu der Kamera wird diese Anzeige regulär dargestellt. Ist der VR Mode nicht aktiviert, wird die Darstellung wie in Abbildung 5.14 gezeigt.

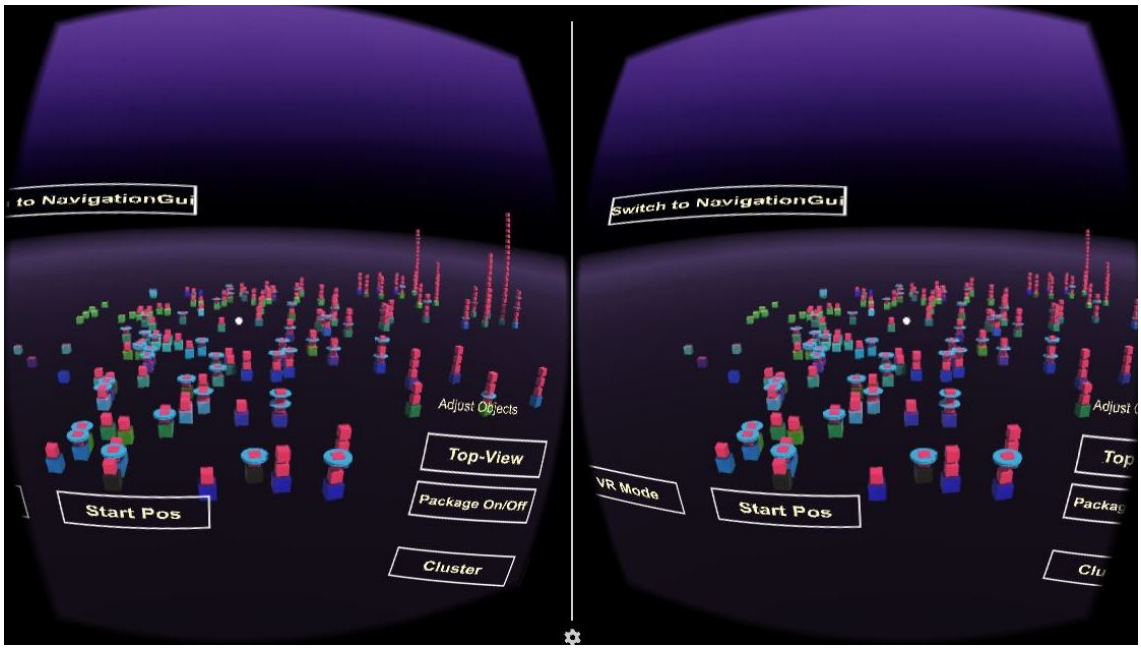

Abbildung 5.14: Ansicht für Google Cardboard.

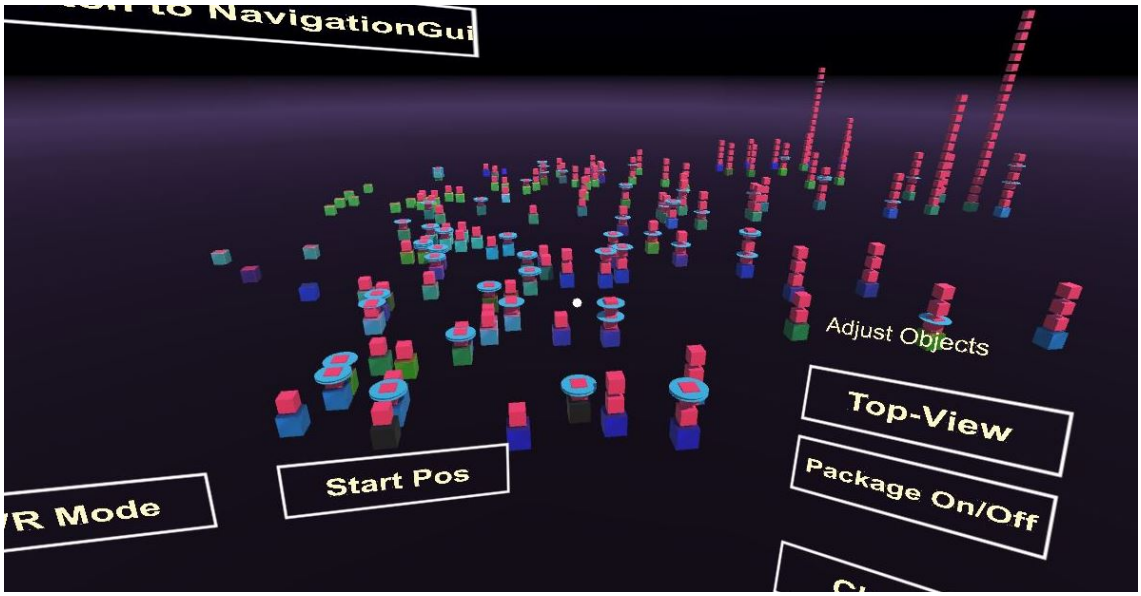

Abbildung 5.15: Ansicht für Nutzung ohne Google Cardboard.

5.3.10 Anzeigen der Imports

Importabhängigkeiten werden mit Linien dargestellt, die via LineRenderer erzeugt werden. Wird das Event ausgelöst, werden von jeder Position (Vector3) der importierten Packages zur Position des selektierten Bundles Linien gezeichnet. Je LineRenderer kann jedoch nur eine Linie erstellt werden, und Objekte können nur einen LineRenderer zugewiesen bekommen. Daher werden beim Startvorgang jedem Packages-importierenden Bundle genauso viele mit LineRenderern bestückte Objekte angefügt, wie die Anzahl der Imports ist. Das Zeichnen der Linien geschieht wie in Abbildung 5.15 gezeigt. Alle Imports sind bereits im Initialisierungsvorgang in GameObject-Listen abgespeichert worden welche nur noch aufgerufen werden müssen, nachdem ein Bundle selektiert wurde. Die Position aus welcher in der Import-Liste ausgelesen werden muss ist gleich dem Index des selektierten Bundles.

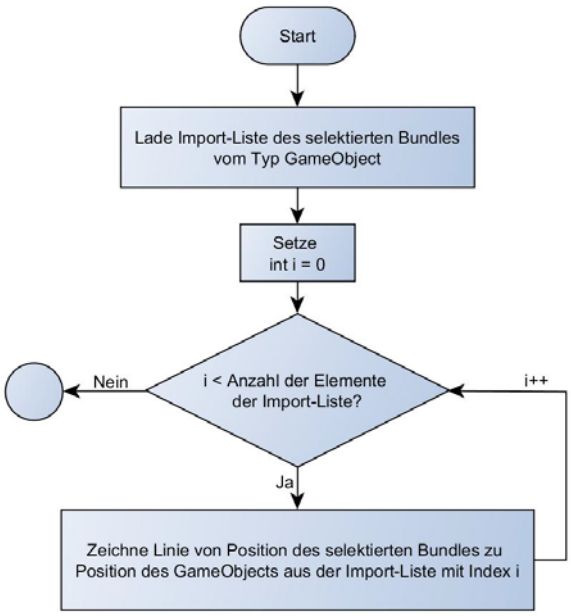

Abbildung 5.16: Ablauf der Linienerstellung bei Imports.

Das erste Import-Objekt in der Liste (mit dem Index 0) bekommt den ersten LineRenderer des Bundles zugeordnet (*lnrender_imp0*). Die Anzahl der Import-Objekte in der Liste ist immer gleich der Anzahl der LineRenderer des Bundles, daher kann die Zuweisung der folgenden LineRenderer aufsteigend vorgenommen werden bis alle Linien gezeichnet wurden. Nach Abbildung 5.17 hat das Bundle *de.rcenvironment.components.evaluationmemory.common* genau 4 Imports. In Abbildung 5.16 wird die Umsetzung der Import-Abhängigkeiten anhand eines Bundles mit 15 Imports gezeigt.

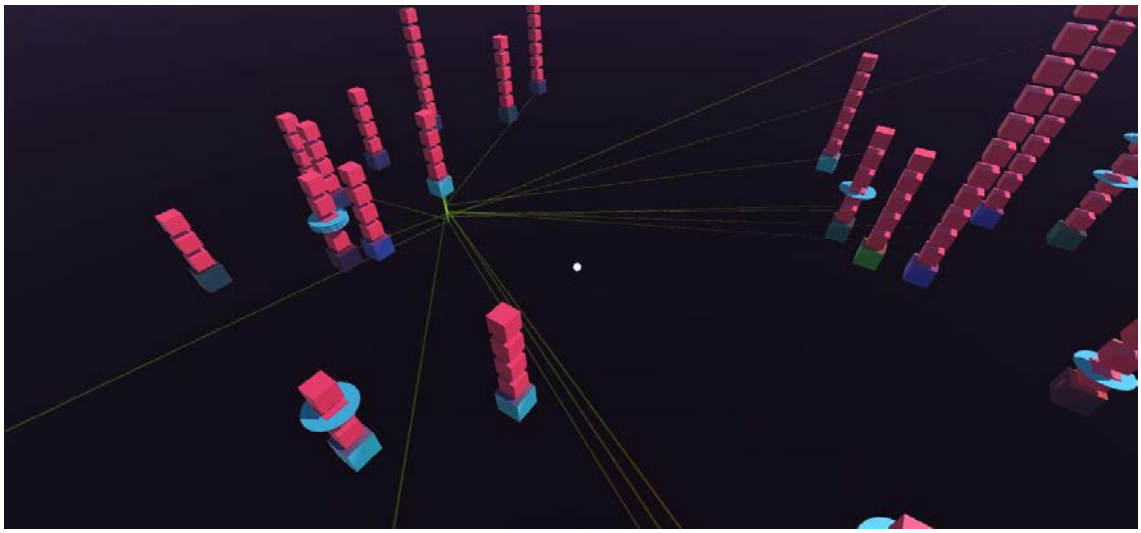

Abbildung 5.17: Imports eines Bundles.

5.3.11 Anzeigen der Exports

Auch hier werden die Abhängigkeiten mit Linien dargestellt. Das Prinzip ist das gleiche wie bei der Anzeige der Import-Abhängigkeiten. Der Unterschied ist in diesem Fall, dass nicht die LineRenderer der Bundles die Linien zeichnen, sondern die der Export-Packages. In Abbildung 5.17 ist zu sehen, dass dem Package *de.rcenvironment.components.evaluationmemory.common(Clone)* zwei LineRenderer für den Export untergeordnet sind (lnrender_exp0, lnrender_exp1), das bedeutet das das Package zweimal exportiert wird. Abbildung 5.18 zeigt die Export-Abhängigkeiten aller Packages eines Bundles mit vielen Exports.

Abbildung 5.18: Anordnung der einzelnen Elemente mit LineRenderer in Unity.

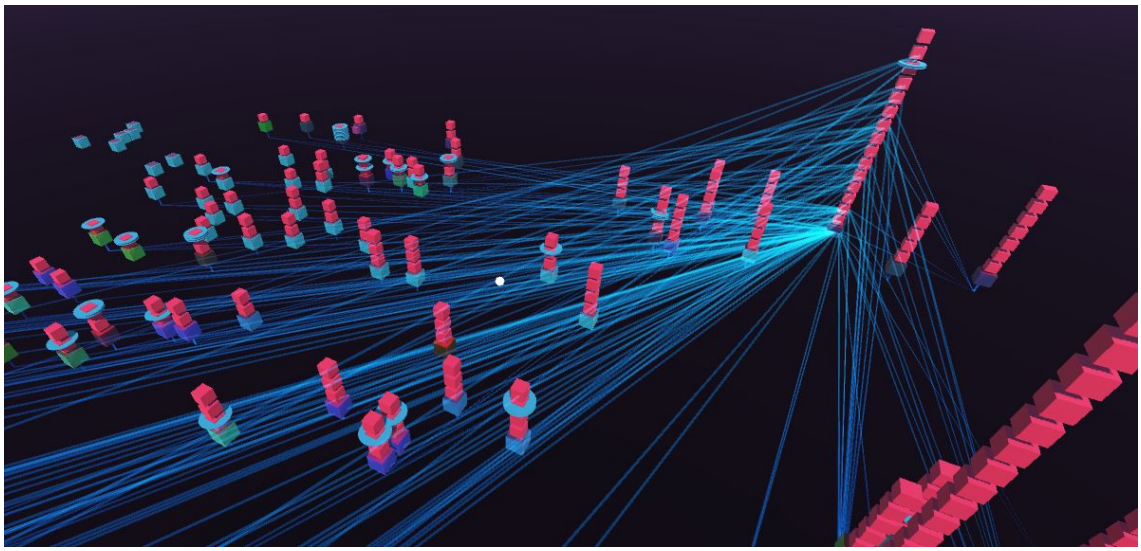

Abbildung 5.19: Export-Abhängigkeiten eines Bundles.

5.3.12 Fortbewegung

Die Fortbewegung wird mit Hilfe einer Positionsänderung, die Frame für Frame aufgerufen wird, umgesetzt. Unity stellt dabei eine Update-Funktion zur Verfügung die immer genau einmal pro Frame aufgerufen wird und mit eigenen Funktionen gefüllt werden kann. Somit ist es möglich ein Script an den Viewer anzuhängen, das diese Funktion mit der Positionsänderung enthält. Dabei wird in jeder Update-Funktion die aktuelle Position, welche vom Datentyp Vector3 ist, genommen und mit einem Richtungsvektor addiert. Damit wird die Bewegung in Blickrichtung Frame für Frame ausgeführt. Ein boolescher Wert wird auf *true* gesetzt wenn die Bewegung ausgeführt werden soll, damit die Bewegung innerhalb der Update-Funktion nur greift, wenn es erwünscht ist.

5.3.13 Rotation der Objekte

Für die Rotation aller Objekte wird ein Rotationskreuz verwendet, welches in Kapitel 4.4.2 genauer beschrieben ist. Anders als bei den Buttons der GUI wird die Rotation ausgelöst wenn auf das entsprechende Rotationselement gezeigt wird. Abbildung 5.19 zeigt das Kreuz, unterteilt in den drei Achsen, welche jeweils eine andere Farbe haben. Je Achse gibt es zwei Rotationselemente, welche die Rotation im bzw. gegen den Uhrzeigersinn auslösen. Ähnlich wie bei der Fortbewegung wird die Rotation über die Update() Funktion ausgelöst und nur solange, wie der Pointer auf das Rotationselement zeigt.

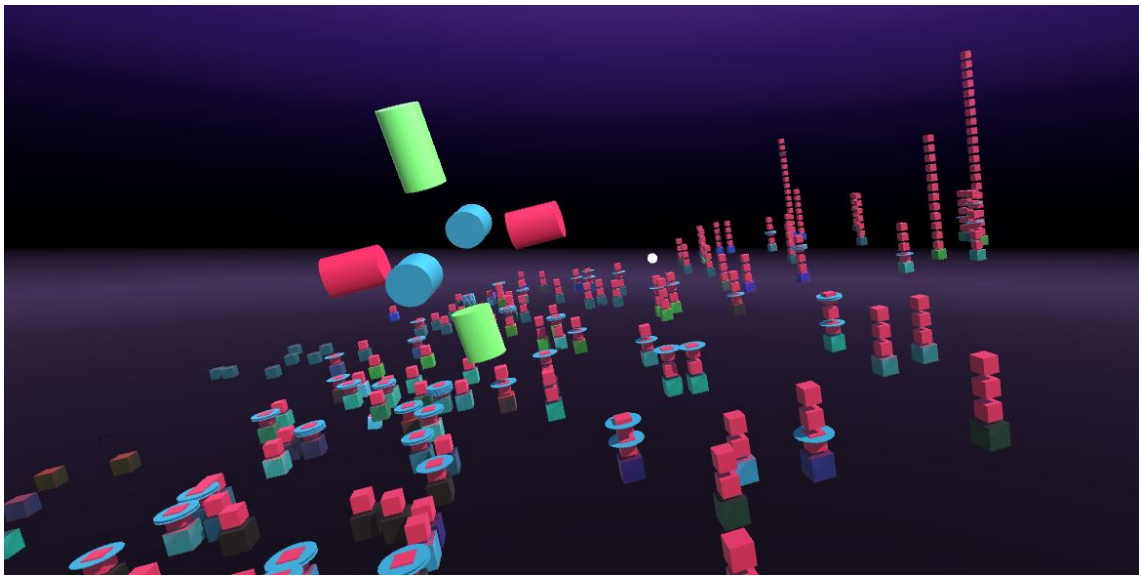

Abbildung 5.20: Rotationskreuz.

5.3.14 Aufbau der Applikation

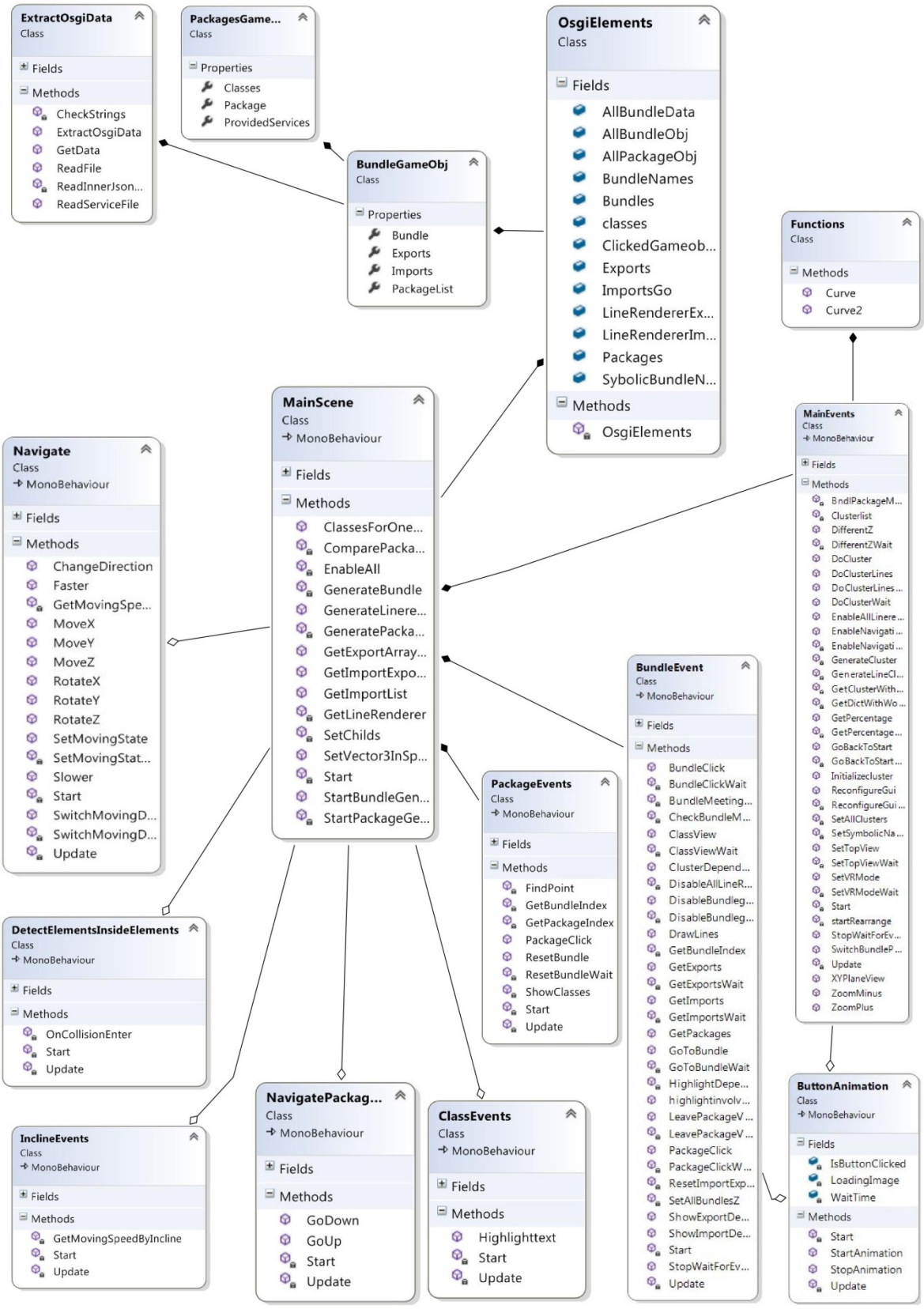

Abbildung 5.21: Klassendiagramm des Visualisierungsprojektes.

6 Zusammenfassung und Ausblick

Für die Darstellung von OSGi-basierten Softwareprojekten wurde ein Konzept entwickelt und umgesetzt um dessen Aufbau besser zu verstehen. Zuerst wurden die Grundlagen von OSGi erläutert, die für das Verständnis des Konzeptes wichtig sind. Die einzelnen Komponenten und deren Zusammenhänge wurden erklärt wie auch das Schichtenmodell von OSGi. Im zweiten Kapitel wurden die theoretischen Grundlagen der Datenvisualisierung dargelegt. Insbesondere wurden die Visualisierung von Software und Softwarearchitektur, Datenvisualisierung im dreidimensionalen Raum sowie der Einsatz von VR in diesem Zusammenhang betrachtet. Ein Hauptaugenmerk wurde auf speziell für dieses Projekt sinnvolle Konzepte gelegt, wie die Visualisierung hierarchischer Daten und Netzdarstellungen. Für das Konzept wurde vorerst eine Auswahl getroffen, welche Daten dargestellt werden und welche vorerst nicht mit einbezogen werden. Auf dieser Basis wurde ein Extraktionstool implementiert, welches die Daten aus einer bereits vorhandenen Model-Datei liest und in eine neue JSON-Datei schreibt. Diese neu erstellte Datei enthält alle Daten des OSGi-Projektes, welche für die Visualisierung wichtig sind. Der Kern dieser Arbeit war die Implementierung des VR-Visualisierungstools, das sowohl mit Google Cardboard betrachtet werden kann, als auch allein mit einem Smartphone. Die mit Unity umgesetzte App stellt damit alle für die Visualisierung relevanten Bestandteile des OSGi-Projektes dar. Als Beispiel für ein geeignetes darzustellendes OSGi-Projekt diente die vom DLR entwickelte Software RCE.

Da eine Vielzahl an Daten nicht mit einbezogen sind bietet das Projekt noch einiges an Erweiterungspotential. Auch die dynamische Visualisierung der Software wurde nicht mit einbezogen.

Auch die Evaluierung durch Benutzerstudien ist noch nicht berücksichtigt worden. Vor allem für die Weiterentwicklung der Visualisierung sind Feedbacks von Nutzern die der Zielgruppe angehören entsprechen wichtig.

Im Feld der Datenvisualisierung mit VR gibt es noch viel Forschungsbedarf, vor allem da Hardware für die Betrachtung von VR-Applikationen erst in dieser Zeit auf den Markt kommt und daher erst jetzt für jedermann erreichbar ist.

Anhänge

Anhang 1

1 Datenträger mit folgendem Inhalt:

- Dieses Dokument im PDF-Format
- Umsetzung des Extraktionstools für das Auslesen der OSGi-Daten mit ausführbarer .exe Datei sowie das Projekt mit Sourcecode und den benötigten Packages (LitJson 0.7.0).
- Umsetzung der Softwarevisualisierung mit ausführbarer .apk Datei für Android sowie das Projekt mit Sourcecode sowie alle benötigten Daten wie Texturen, Materialien und Plugins.

Literaturverzeichnis

[1] G. Wütherich; N. Hartmann; B.Kolb; M.Lübken. 2008. *Die OSGi Service Plattform.* Heidelberg: dpunkt.verlag

[2] U. Fildebrandt. 2012. *Software modular bauen.* Heidelberg: dpunkt.verlag

[3] M. Ward; G. Grinstein; D. Keim. 2010. *Interactive Data Visulaization*. A K Peters, Ltd

[4] S. Diehl. 2007. *Visualizing the Structure, Behaviour and Evolution of Software*. Springer Verlag

[5] B. Johnson; B. Shneiderman. 1991. *Tree-Maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures*. Proceedings of the 2Nd Conference on Visualization, IEEE Computer Society Press

[6] Y. Ghanam; S. Carpendale. 2015. *A Survey Paper on Software Architectur Visualization.* Department of Computer Science University of Calgary, Canada

[7] R. Spence. 2007. *Information Visualization, Design for Interaction*. Pearson Education Limited

[8] C.Chen; W. Härdle; A. Unwin. 2008. *Handbook of Data Visualization*. Springer Verlag

[9] Uni Rostock. 2008. *Visualizing large trees and graphs using a 3D cone tree layout*. Uni Rostock

[10] T. Ball; S.G. Eick. 1996. *Software Visualization in the Large.* IEEE Computer, Vol. 29

[11] Ji Soo Yi; Youn ah Kang; John T. Stasko; Julie A. Jacko. 2007. *Toward a Deeper Understanding of the Role of Interaction in Information Visualization.* IEEE Computer

[12] J. I. Maletic; J. Leigh; A. Marcus; G. Dunlap. 2001. *Visualizing Object-Oriented Software in Virtual Reality.* IEEE Xplore Conference

[13] M. Herrmann; H-J. Groß; P. Rothländer; M. Flehmig; and I. Melzer. 2005. Daimler AG. SOA Special Interest Group

[14] R. Ebert. 2011. *Entwicklung von Desktop-Anwendungen mit der Eclipse Rich Client Platform 3.7.* Initiale Veröffentlichung für Eclipse 3.6.2

[15] M. Bostock; V. Ogievetsky; and J. Heer. 2011. *D3: Data-driven documents.* IEEE Trans. Visualization & Comp. Graphics vol. 17

[16] W. S. Cleveland; R. McGill. 1984. *Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods*. Journal of the American Statistical Association, Vol. 79, No. 387

[17] OSGi Alliance. 2014. *OSGi Core*. URL: http://www.osgi.org

[18] D. Seider; A. Schreiber; T. Marquardt. 2016. *Visualizing Modules and Dependencies of OSGi-based Applications*. 2016 IEEE 4th Working Conference on Software Visualization (VISSOFT), October 3-4, 2016, Raleigh, USA (akzeptiert)

[19] T. Marquardt. 2016. *Extraktion und Visualisierung von Beziehungen und Abhängigkeiten zwischen Komponenten großer Softwareprojekte*. Masterarbeit, Technische Universität Dortmund. URL: http://elib.dlr.de/105575/

[20] IEEE. 2012. *Compressed Adjacency Matrices: Untangling Gene Regulatory Networks.* IEEE Transactions on Visualization and Computer Graphics, vol. 18

[21] A. Nocaj; U. Brandes. 2012. *Computing Voronoi Treemaps*. Eurographics Conference on Visualization

Website-Verzeichnis

[100] *OSGi Alliance Blog.* URL: http://blog.osgi.org/2007/09/soa-osgi.html (besucht am 14. Juli 2016)

[101] *Hot-Plugging mit Eclipse RCP und* OSGi. URL: https://jaxenter.de/hot-plugging-mit-eclipse-rcp-und-osgi-3-7157 (besucht am 16. Juli 2016)

[102] *RCE*. URL: http://www.dlr.de/sc/desktopdefault.aspx/tabid-5625/9170_read-17513/ (besucht am 20. Juli 2016)

[103] *Architectural Patterns and Styles*. URL: https://msdn.microsoft.com/en-us/library/ee658117.aspx (besucht am 24. Juli 2016)

[104] *Open GL*. URL: https://www.opengl.org/ (besucht am 2. August 2016)

[105] *Unity Technologies*. URL: https://unity3d.com/ (besucht am 7. August 2016)

[106] *LitJson*. URL: https://lbv.github.io/litjson/ (besucht am 20. August 2016)

[107] *Unity Coroutines.* URL: https://docs.unity3d.com/Manual/Coroutines.html (besucht am 25. August 2016)

[108] *Google VR SDK for Unity*. URL: https://developers.google.com/vr/unity/ (besucht am 1. September 2016)

[109] *Direct3D*. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/bb153256%28v=vs.85%29.aspx (besucht am 1. September 2016)

[110] *HTC Vive*. URL: https://www.htcvive.com/de/product/ (besucht am 1. September 2016)

[111] *Immersive Design*. URL: https://backchannel.com/immersive-design-76499204d5f6#.6fp1bz5sa (besucht am 1. September 2016)

[112] *The Plugin Lifecycle*. URL: https://wiki.gxsoftware.com/wiki/display/PD/The+Plugin+Lifecycle (besucht am 1. September 2016)

[113] *Network and Realationship Visulaisation* . URL: http://storiesthroughdata.blogs.lincoln.ac.uk/2012/02/12/network-and-relationship-visualisation/ (besucht am 17. August 2016)

[114] *Diagrams TwoD Sunburst*. URL: http://projects.haskell.org/diagrams/haddock/Diagrams-TwoD-Sunburst.html (besucht am 19. August 2016)

[115] *UML-Klassendiagramme*. URL: http://www.tilman.de/uni/ws03/alp/uml.php (besucht am 21. August 2016)

[116] *Modular Synthesizer*. URL: http://www.sdiy.info/w/Modular_synthesizer (besucht am 25. August 2016)

Abbildungsverzeichnis

Eidesstattliche Versicherung

Ich versichere, dass die vorliegende Abschlussarbeit selbstständig und ohne unerlaubte Hilfe Dritter verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die inhaltlich oder wörtlich aus Veröffentlichungen stammen, sind als solche kenntlich gemacht. Diese Arbeit lag in gleicher oder ähnlicher Weise noch keiner Prüfungsbehörde vor und wurde bisher nicht veröffentlicht.

Berlin, 5. September 2016