DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

Studiengang Informationstechnik

Bachelorarbeit

zur Erlangung des akademischen Grades eines Bachelor of Engineering

3D-Visualisierung von wissenschaftlichen Daten in einer RCP-Anwendung

von Hendrik Abbenhaus

Matrikelnummer: 6337947

Kurs: TINF13ITIN

Bearbeitungszeitraum: 27.06.2016 - 26.09.2016

DLR

Deutsches Zentrum für
Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

Standort: Köln-Porz

Simulations- und Softwaretechnik
Abteilung: Verteilte Systeme und Komponentensoftware

Betreuer:
Dipl.-Math. Sascha Zur
Prof. Dr. Rainer Colgen

Eidesstattliche Erklärung

gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 22. September 2011.

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Köln-Porz, den 20. September 2016

Abstract

The facility Simluation and Software Technology at the German Aerospace Center develops the software RCE (Remote Component Environment). RCE is an open source distributed, workflow-driven integration environment and based on the Eclipse RCP framework. For visualizing two-dimensional data, RCE uses a visualization library that was developed within a student research project. The objective of this thesis is adding functionality of visualizing three-dimensional data to this library.

First a technology for drawing 3d objects must be found. For getting a decision, 14 libraries are compared. The open source library *jzy3D* meets the requirements best. Implementing of *jzy3D* into the visualization library includes three different parts.

In the first part *view and presentation* fundamental structures for visualizations of three-dimensional data are built. In a secound part an availabe plugin for *jzy3D* was used to integrate the created visualization into a RCP application. Within a last part a concept to organize the data efficiently was developed and implemented.

The resulting prototype provides an API for creating visualization for three-dimensional data in a RCP application.

Zusammenfassung

In der Einrichtung Simulations- und Softwaretechnik (SC) am Deutschen Zentrum für Luft- und Raumfahrt (DLR) wird die Software RCE (Remote Component Environment) entwickelt. Sie baut auf dem Eclipse RCP-Framework auf und verwendet zur Visualisierung von zweidimensionalen Daten eine innerhalb einer Studienarbeit entwickelte Schnittstelle. Diese Schnittstelle wird im Rahmen dieser Bachelorarbeit um die Darstellung von dreidimensionalen Daten erweitert. Zunächst musste eine grundsätzliche Technologie für die Umsetzung ausgewählt werden. Dazu wurden 14 Bibliotheken miteinander verglichen und die open source Java-Bibliothek *jzy3D* ausgewählt.
Bei der Implementierung mussten Anforderungen für die Schnittstelle sowie Rahmenbedingungen durch die verwendete Bibliothek berücksichtigt werden. Sie gliedert sich in drei Abschnitte. Innerhalb des ersten Abschnitts *Darstellung und Präsentation* werden strukturelle Grundlagen für eine Darstellung von dreidimensionalen Daten geschaffen. In einem zweiten Teil *Einbindung in eine RCP-Anwendung* wurde eine Einbindung in eine RCP-Anwendung ermöglicht. Dabei wurde auf ein Plugin der Bibliothek zurückgegriffen. In einem letzten Teil wurde ein Konzept für eine effiziente Datenverwaltung entwickelt und implementiert.
Das Ergebnis dieser Arbeit ist ein entstandener Prototyp einer Schnittstelle, die Darstellungen von zwei- und dreidimensionalen Daten innerhalb einer RCP-Anwendung ermöglicht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

API	**A**pplication **P**rogramming **I**nterface
DAWN	**D**ata **A**nalysis **W**orkbe**N**ch
DLR	**D**eutsches Zentrum für **L**uft- und **R**aumfahrt
EPL	**E**clipse **P**ublic **L**icense
GUI	**G**raphical **U**ser **I**nterface
RCE	**R**emote **C**omponent **E**nvironment
RCP	**R**ich **C**lient **P**latform
SC-IVS	**S**imulations- und Softwaretec**h**nik - **I**ntelligente und **v**erteilte **S**ysteme
SWT	**S**tandard **W**idget **T**oolkit
UI	**U**ser **I**nterface

1 Einleitung

Das folgende Kapitel beschreibt zunächst das Umfeld der Arbeit, stellt dann die hinter der Arbeit liegende Motivation dar und benennt die daraus resultierende Aufgabenstellung. Abschließend gibt es einen Ausblick auf die Gliederung der Arbeit.

1.1 Umfeld dieser Arbeit

Die hier vorliegende Arbeit ist beim deutschen Zentrum für Luft- und Raumfahrt am Standort Köln in der Einrichtung Simulations- und Softwaretechnik verfasst worden.

Deutsches Zentrum für Luft- und Raumfahrt (DLR) Das Deutsche Zentrum für Luft- und Raumfahrt e.V. ist die von der Bundesrepublik Deutschland als Raumfahrtagentur beauftragte Forschungseinrichtung und hat die Aufgabe, die deutschen Interessen an der Raumfahrt zu vertreten und umzusetzen. Derzeit sind ca. 8.000 Mitarbeiter an 16 nationalen Standorten (Köln, Berlin, Bremen, Göttingen, ...) und weiteren internationalen Büros (Brüssel, Paris, Tokio und Washington D.C.) beschäftigt. Jeder Standort setzt dabei einen spezifischen Schwerpunkt. Das DLR deckt neben der Forschung in den Gebieten Luft- und Raumfahrt auch die Bereiche Sicherheit, Energie und Verkehr ab und kooperiert weltweit mit anderen Forschungseinrichtungen, Universitäten und der Industrie [1].

Die Einrichtung Simulations- und Softwaretechnik Die Einrichtung Simulations- und Softwaretechnik beschäftigt sich in Zusammenarbeit mit anderen Instituten und Einrichtungen des DLR mit der Forschung und Entwicklung innovativer Software-Engineering-Technologien. So werden innerhalb des DLR neue Softwaretechnologien mit den Themenschwerpunkten verteilte und mobile Systeme, eingebettete Systeme, Visualisierung und High Performance Computing erforscht. Neben den aktuellen Themenschwerpunkten zur Unterstützung der international führenden Projektpartner, ist die Einrichtung auch Bestandteil in internationalen Foren und Standardisierungsgremien [2].

1.2 Motivation

Wissenschaftliche Daten können in der Regel mit Hilfe geeigneter Darstellungen besser ausgewertet und analysiert werden. Bei der Ausführung von Simulationen und Prozessketten fällt häufig eine große Menge verschiedenartiger wissenschaftlicher Daten an. Mit Hilfe von Prozessketten können beispielsweise Optimierungen für unterschiedliche Zusammenhänge durchgeführt werden. Optimierungen erzeugen häufig eine große Menge numerischer Daten, welche sowohl zwei- als auch dreidimensional sein können. Eine bessere Analyse dieser Daten kann mit Hilfe einer geeigneten Visualisierung erleichtert werden. Damit werden Zusammenhänge deutlicher und Fehler innerhalb der Berechnungen können frühzeitig erkannt werden. Fällt ein Fehler bereits während der Berechnung einer Prozesskette auf, können laufende Berechnungen gestoppt und somit Rechenzeit gespart werden. Mit Hilfe der Visualisierung lassen sich nach der Berechnung Endergebnisse überprüfen und später präsentieren. Die Ausführung und Darstellung von Prozessketten kann innerhalb von RCP-Anwendungen, wie beispielsweise dem Simulationsframework RCE geschehen. Für die Darstellung von zweidimensionalen Daten existiert bereits eine Schnittstelle für die Anbindung einer Open Source Visualisierungsbibliothek in eine RCP-Anwendung. Dreidimensionale Daten können bisher über diese Schnittstelle nicht dargestellt werden.

1.3 Aufgabenstellung

Innerhalb dieser Arbeit soll eine vorhandenen Schnittstelle für Visualisierungen im RCP-Bereich um die noch nicht existierende Möglichkeit der Visualisierung von dreidimensionalen Daten erweitert werden. Anwendung findet dies in der RCP-Software RCE, welche innerhalb der Abteilung entwickelt wird. Es soll herausgefunden werden, was mit den Daten, welche von RCE über die Schnittstelle hereingereicht werden, passieren muss, um eine 3D-Visualisierung zu ermöglichen. Am Ende der Arbeit soll es möglich sein, von RCE gelieferte Daten in einer RCP-Anwendung darzustellen. Ausgenommen von der Arbeit sind sämtliche Bedienelemente für die Visualisierung.

1.4 Schwerpunkte der Arbeit

Die Arbeit kann schwerpunktmäßig in drei Teile eingeteilt werden, welche am Ende zusammenwirken müssen.

Datenpräsentation Der erste Teil beschäftigt sich mit der eigentlichen Anzeige bzw. Präsentation der Daten. Hierbei ist zu Beginn der Arbeit noch nicht festgelegt, welche Art von Visualisierungsoberfläche benutzt werden soll. Zunächst geht es um die Struktur und Abgrenzung innerhalb der API zur Darstellung von dreidimensionalen Daten.

RCP Einbindung Einen weiteren Teil stellt die Einbindung der Anzeige in eine Eclipse RCP-Anwendung dar. Je nach genutzter Visualisierungsoberfläche gestaltet sich eine solche Integration als nicht trivial.

Datenverwaltung Die durch die Schnittstelle ankommenden Daten müssen vor ihrer Anzeige verwaltet und organisiert werden, sodass weder Daten verloren gehen, noch redundanter Speicher belegt wird. Außerdem sollen die Daten so verwaltet werden, dass ausgewählte Datensätze nachträglich geändert werden können. Die Verwaltung und Organisation dieser Daten stellt den letzten Teil dar.

1.5 Vorgehensweise

Die Vorgehensweise kann in die hier aufgeführten Punkte unterteilt werden. Diese Einteilung zeigt sowohl eine logische Gliederung, als auch eine zeitliche Aufeinanderfolge der Arbeitsabschnitte, da einzelne Punkte aufeinander aufbauen.

Einführung und Möglichkeiten Zunächst ist es notwendig einen allgemeinen Überblick über die Thematik der Visualisierung zu bekommen. Hierzu sollten die möglichen Arten der Visualisierung und die Möglichkeiten der Umsetzungen deutlich werden.

Anforderungsanalyse Innerhalb der Anforderungsanalyse werden sämtliche Anforderungen zusammengetragen. Dabei werden sowohl die Anforderungen der Anwender als auch technische Anforderungen von Entwicklern betrachtet.

Recherche Innerhalb einer Recherche muss herausgefunden werden, welche Technologien bzw. Frameworks für eine Verwendung in Frage kommen würden.

Technologie Entscheidung Auf Grundlage der Recherche muss eine Entscheidung getroffen werden, welches Frontend bzw. Framework verwendet werden sollte.

Konzeption und Entwurf der Architektur In einem weiteren Schritt wird ein Konzept für die Architektur erstellt, das auch die benötigte Datenstruktur des Frontends berücksichtigt.

Implementierung Innerhalb der Implementierung wird das erstellte Konzept der Architektur umgesetzt.

Kritische Reflexion Abschließend wird kritisch reflektiert, ob das erforderte Ziel erreicht wurde und einen Ausblick auf weitere mögliche Entwicklungen gegeben.

1.6 Überblick

Die hier vorliegende Arbeit ist in fünf Kapitel aufgeteilt. Zunächst werden innerhalb des zweiten Kapitels nötige Grundlagen für das weitere Verständnis erläutert. Diese geben eine Einführung in den aktuellen Stand der Technik und der verwendeten Technologien. Zu Beginn wird die Ausgangssituation, die als Grundlage der Arbeit benutzt wird, dargestellt. Anschließend wird ein kurzer Überblick über angesprochene Technologien und Darstellungsformen gegeben.
In Kapitel drei wird auf Grundlage einer Literatur- bzw. Internetrecherche eine Bibliothek ausgewählt, mit der die Problemstellung umgesetzt wird. Hierbei werden auch alle gesammelten Anforderungen berücksichtigt.
Im praktischen Teil innerhalb des vierten Kapitels wird der Entwurf und die Implementierung des Systems beschrieben. Dieser Abschnitt wird entsprechend der drei Schwerpunkte (siehe Abschnitt 1.4) in drei Teile unterteilt. An dieser Stelle wird der Entwurf und die Implementierung der Software beschrieben.
Abschließend werden innerhalb des fünften Kapitels die Ergebnisse zusammengefasst und ein Ausblick auf denkbare Erweiterungen gegeben.

2 Grundlagen

Zum besseren Verständnis der Vorgehensweise sind einige Grundkenntnisse nötig. In den folgenden Abschnitten werden diese Grundlagen näher beschrieben. Zunächst wird die Ausgangssituation und damit der aktuelle Stand der Technik beschrieben. Im Anschluss daran erfolgt eine kurze Einführung in das RCP Framework. Da das Ergebnis der Arbeit in der Integrationsumgebung RCE verwendet werden soll, erfolgt auch eine Einführung in RCE selbst. Abschließend werden die für die Visualisierung betrachteten Darstellungen näher beschrieben.

2.1 Ausgangssituation

Ausgangspunkt der aktuellen Untersuchung ist eine im Rahmen einer Studienarbeit der TU Dortmund erstellte Schnittstelle für eine Visualisierung von zweidimensionalen Daten [3]. Mit Hilfe einer festgelegten Vererbungshierarchie wird eine Instanz von *JFreeChart* bereitgestellt. *JFreeChart* ist eine Bibliothek, die bisher zur Darstellung von zweidimensionalen Daten und Diagrammen verwendet wird [3, S.31]. Im Rahmen dieser Arbeit wurden bereits einige Überlegungen zur Darstellung von dreidimensionalen Daten getätigt, welche als Ansatzpunkt für diese Arbeit dienen [3, S.11 ff.]. Allerdings benötigt die Implementierung der vorhandene Schnittstelle für zweidimensionale Daten eine Überarbeitung, da einige Bereiche architektonisch ausgebessert werden müssen. Diese Überarbeitung ist zwar nicht Teil der Aufgabe, fordert aber eine direkte Berücksichtigung einer klaren Struktur im dreidimensionalen Bereich. Unabhängig der eigentlichen Studienarbeit, kann zusätzlich die Dokumentation der Schnittstelle [4] als Grundlage genutzt werden.

2.2 Das Eclipse RCP Framework

Das Eclipse **R**ich **C**lient **P**latform (RCP) stellt ein Open Source Grundgerüst zur Entwicklung von Desktop-Anwendungen bereit. Die Plattform ermöglicht es, Komponenten der Eclipse Entwicklungsumgebung in anderen Anwendungen wiederzuverwenden, die dann um eigene Anwendungsfunktionalitäten erweitert werden können. Als Grundlage für die eigene Anwendung kann beispielsweise das erweiterbare Plug-in-System, welches eine Modularität und Erweiterbarkeit ermöglicht, die Hilfe-Komponente oder der Update-Manager verwendet

werden. Auch kann der konsistente Aufbau der Benutzeroberfläche und das zugehörige einheitliche, durchdachte Bedienkonzept übernommen werden. Dies wird besonders durch die verwendete **E**clipse **P**ublic **L**icense (EPL) unterstützt, die sogar den kommerziellen Einsatz des Frameworks erlaubt [5, S.3 f.].

2.3 Die Integrationsumgebung RCE

Die Gruppe **S**imulations- und Softwaretec**h**nik - **I**ntelligente und **v**erteilte **S**ysteme (SC-IVS) entwickelt im DLR unter anderem die Open Source Integrationsumgebung RCE (**R**emote **C**omponent **E**nvironment). RCE ist eine verteilte Plattform zur Integration von Simulationsanwendungen, steht unter der EPL und basiert auf der Technologie RCP (vgl. Abschnitt 2.2). In RCE können Prozessketten bestehend aus verschiedenen Simulationsprogrammen erstellt und anschließend berechnet werden. Die Prozessketten können verschiedene Anwendungsbereiche abdecken. Ein Anwendungsbereich ist die Luft- und Raumfahrt. Innerhalb und außerhalb des DLR werden beispielsweise mit RCE Flugzeuge mit neuartiger Geometrie entworfen und optimiert [6] [7].
Aktuell können in RCE mit Hilfe einer dafür konzipierten und oben bereits beschriebenen Schnittstelle verschiedene zweidimensionale Visualisierungen dargestellt werden. Eine Darstellung von dreidimensionalen Daten ist aktuell nicht möglich.

2.4 Dreidimensionale Darstellungen

In den folgenden Kapiteln werden unterschiedliche Darstellungsformen von dreidimensionalen Daten angesprochen. Nachfolgend werden die für eine Darstellung in RCE relevanten Diagrammtypen näher betrachtet. Im Falle einer vollständigen Eigenentwicklung ist es wichtig, den Aufbau und die Funktionsweise der hier aufgeführten Darstellungsformen zu verstehen, um sie selbst implementieren zu können.

2.4.1 Surface Plots

Surface Plots sind Diagramme, in denen dreidimensionalen Daten dargestellt werden können. Ein Beispiel einer solchen Darstellung ist in Abbildung 1 aufgezeichnet. Wie dieses Beispiel zeigt, werden innerhalb eines Surface Plots nicht nur einzelne Datenpunkte, sondern eine funktionale

Beziehung zwischen einer definierten, abhängigen Variable Y und zwei unabhängigen Variablen X und Z gezeigt. Für die Konstruktion eines Surface Plots wird zunächst für jeden Punkt eines definierten zweidimensionalen Gitters mit den Koordinaten X und Z ein Wert Y berechnet. Der Abstand der einzelnen Punkte zueinander bzw. die Anzahl der Punkte werden durch den Benutzer definiert und definieren gleichzeitig den Bereich, in dem Daten dargestellt werden. Der errechnete Y Wert ist ein gewichteter Durchschnitt von allen umliegenden Werten. Der dreidimensionale Surface Plot wird nun durch Verwendung dieser Durchschnittswerte konstruiert. Hierzu werden die umliegenden Punkte durch eine ebene Fläche verbunden [8, S.1].

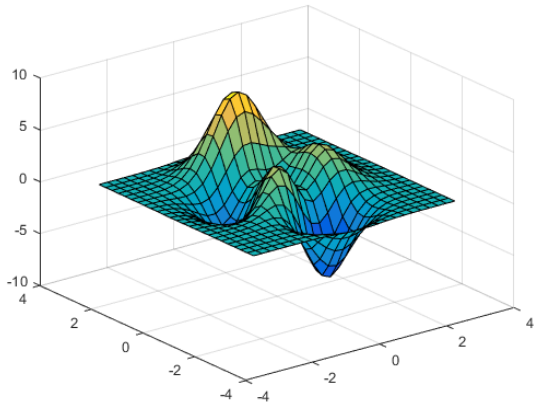


Abbildung 1: Beispiel eines Surface Plots [9]

2.4.2 Scatter Plots

Ein Scatter Plot ist ein mathematisches Diagramm bzw. eine Darstellungsform in dem die Werte einer Datensammlung dargestellt werden können (vgl. Beispiel Abbildung 2). Es benutzt je nach Anzahl der Variablen ein ebenes oder ein räumliches kartesisches Koordinatensystem. Dabei verwendet jede Variable eine eigene Achse. Die Daten werden innerhalb des Koordinatensystems als Punkte dargestellt, wobei die Koordinaten eines jeden Punktes mit den entsprechenden Werten der Variablen übereinstimmt. Folglich werden mit Hilfe der Positionen der Punkte im Raum Abhängigkeiten zwischen einzelnen Variablen deutlich. Zusätzlich gibt es die Möglichkeit, die Punkte mit Farben zu kodieren, um die Anzahl der darzustellenden Variablen um eine zu erhöhen [10, S. 166f.].

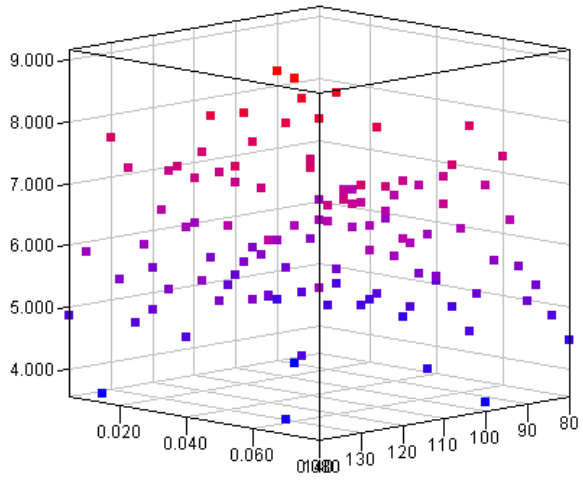


Abbildung 2: Beispiel eines Scatter Plot [11]

2.4.3 Function Plots

Ein Function Plot stellt mathematische Funktionen grafisch dar. Die Bezeichnung Function Plot beschreibt folglich, anders als die bisherigen Bezeichnungen der Surface bzw. Scatter Plots, nicht die Darstellungsform selbst. Sie beschreibt, dass als Datenquelle nicht einzelne Punkte vorliegen, sondern eine Abhängigkeit zwischen mindestens zwei Variablen.

3 Auswahl einer Bibliothek zur 3D Visualisierung

Bevor mit einer konkreten Konzeptentwicklung oder Implementierung begonnen werden kann, muss eine technologische Entscheidungen hinsichtlich des für die Visualisierung verwendeten Frameworks getroffen werden. In den folgenden Abschnitten werden die Möglichkeiten einer Umsetzung evaluiert und auf Grundlage der Anforderungen eine konkrete Entscheidung getroffen.

3.1 Möglichkeiten der Umsetzung

Prinzipiell gibt es zwei Möglichkeiten einer Umsetzung. Auf der einen Seite gibt es die Möglichkeit eine eigene 3D-Visualisierungsbibliothek zu entwickeln. Vorstellbar sind verschiedene Arten, wie beispielsweise eine Umsetzung mit *JavaFX* oder *OpenGL*. Eine Eigenentwicklung hat den Vorteil, dass alle Anforderungen berücksichtigt werden können. Folglich kann im späteren Verlauf der Entwicklung der Funktionsumfang fast beliebig erweitert oder verändert werden. Allerdings ist der zeitliche Aufwand und der Umfang entsprechend hoch.
Die Alternative zu einer Eigenentwicklung ist die Einbindung einer geeigneten Bibliothek. Hier ist der zeitliche Aufwand und der entsprechende Umfang der Implementierung wesentlich geringer als bei einer Eigenentwicklung. Häufig sind die grundlegenden, benötigten Funktionen für eine Darstellung bereits optimiert. Jedoch hängt die Erweiterbarkeit der Schnittstelle der einzelnen Bibliothek davon ab, in wieweit diese für die eigene Verwendung anpassbar ist.

3.2 Anforderungen an eine 3D Visualisierungsbibliothek

Es ist zunächst nötig, relevante Anforderungen auszuarbeiten, um konkrete Entscheidungen für eine entsprechende Technologie fällen zu können. Durch einen bekannten Nutzerkreis von RCE war es bereits in der Vergangenheit möglich, Nutzer konkret nach Anforderungen an solch eine Visualisierung zu befragen. Die Anforderungen des Nutzerkreises lagen entsprechend vor Beginn der Bearbeitung dieser Arbeit vor und lassen sich durch die folgenden Punkte beschreiben:

- Funktionsumfang
  - 3D Scatter Plots
  - 3D Surface Plots
  - Metainformationen bereithalten
    Je nach Frontend ist es wichtig, Metainformationen für die Darstellung bereitzuhalten (zB. die Achsenbeschriftung)
- Aktualisierbarkeit der Datensätze und Darstellung
  Es soll möglich sein, die Visualisierung während einer Berechnung in RCE zu nutzen. Dabei werden die Daten ständig aktualisiert. Eine Aktualisierung der Daten soll unmittelbar zu einer Aktualisierung der Darstellung führen.
- Allgemeines Aussehen/Design
  Die grafische Umsetzung bzw. das Design sollte ansprechend sein und sich angenehm in die Benutzeroberfläche von RCE einpflegen.
- Maus-Integration
  Es sollte möglich sein, mit der Maus innerhalb der Darstellung zu interagieren. Mit Hilfe des Mausrades soll ein Zoomen bzw. Vergrößern der Darstellung möglich sein, eine Auswahl von einzelnen Datenpunkten soll über einen rechts Klick ermöglicht werden und durch gleichzeitiges Klicken und Bewegen soll die Position des Betrachters auf die Darstellung veränderbar sein.

Auch aus Entwicklerseite gibt es Anforderungen, die bei einer Entscheidung berücksichtigt werden müssen. Die durch die Nutzer beschriebenen Anforderungen werden um folgende Punkte ergänzt:

- Kompatible Lizenz
  Da das Ergebnis der Arbeit in RCE zum Einsatz kommt, müssen eventuell eingesetzte Bibliotheken lizenztechnisch zu RCE passen. Zur RCE Lizenz EPL passende Lizenzen sind beispielsweise LGPL, BSD, new BSD oder MIT.
- SWT Kompatibilität
  Die Möglichkeit einer Integration in eine RCP-Anwendung muss gegeben sein.
- Aktivität der Entwicklung
  Auftretende Probleme innerhalb der Bibliothek sollten möglichst zeitig behoben werden.

- Ausreichende Dokumentation
  Die Bibliothek und die durch die Bibliothek bereitgestellten Schnittstellen sollten ausreichend dokumentiert sein.

3.3 Visualisierung in anderen Anwendungen

Für die Visualisierung von entsprechenden Daten werden von Nutzern häufig Beispiele aus anderen wissenschaftlichen Programmen genannt. Um ein Verständnis für mögliche Ergebnisse unterschiedlicher Visualisierungen zu bekommen und die Anforderungen der Nutzer besser zu verstehen, macht es Sinn, die von den Nutzern genannten Beispiele und Software-Lösungen zu betrachten.
Beispielsweise unterstützt die Open-Source-Software **D***ata* **A***nalysis* **W***orkbe***N***ch (DAWN) Science* eine Visualisierung von ein-, zwei- und dreidimensionalen Daten [12]. Die Visualisierung wird aktuell innerhalb der Anwendung auf Basis von *OpenGL* mithilfe des Frameworks *jReality* umgesetzt. Zukünftig soll ein Umstieg zu *JavaFX* erfolgen. [13]
Im Zusammenhang mit Visualisierung wird häufig auch die Software *Modelcenter* der Firma *Phoenix Integration* genannt. Sie ist im Gegensatz zu *DAWN Science* weder Open-Source, noch kostenfrei, sondern nur kommerziell verfügbar. Innerhalb der Software werden unterschiedliche Visualisierungen unterstützt. Im dreidimensionalen Umfeld sind dies Surface, Scatter und Contour Plots. Visualisierungen im zwei- und dreidimensionalen Umfeld lassen sich innerhalb der Software durch eine Vielzahl von Funktionen anpassen und konfigurieren. [14]

3.4 Mögliche Java Bibliotheken

Eine der Möglichkeiten einer Umsetzung ist die Verwendung einer Bibliothek (vgl. Abschnitt 3.1). Durch eine Internetrecherche und Betrachtung anderer Anwendungen (vgl. Abschnitt 3.3) ergeben sich mögliche Kandidaten für eine Verwendung einer bereits vorhanden Bibliothek. Auch werden die in der Arbeit von Herrn Marquardt bereits betrachten Kandidaten erneut für eine Verwendung im dreidimensionalen Umfeld betrachtet [3, S. 10]. Es ergibt sich folgende Liste mit Namen der entsprechenden Java-Bibliotheken:

- FXyz
- Orson Charts

- jzy3d
- jReality
- JOGL
- JFreeChart
- jmathplot
- jCharts
- Gral
- charts4j
- Java Plot
- FreeHep 3d
- Java Surface Plot
- JMathtools

Die hier entstandene Liste wird im Folgenden eingegrenzt werden müssen. Ziel ist es eine Entscheidung zu treffen, die entweder eine dieser Bibliotheken berücksichtigt, oder eine komplette Eigenentwicklung zur Folge hat.

3.5 Eingrenzung von Bibliotheken

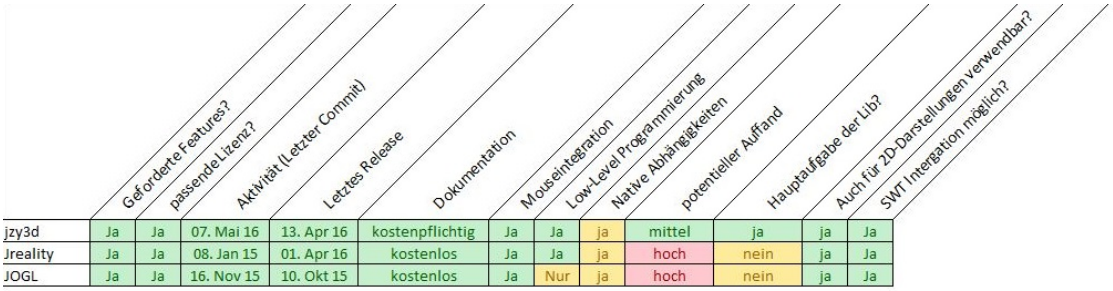
Die in Abschnitt 3.4 entstandene Liste an möglichen Bibliotheken gilt es zu vergleichen. In Abbildung 3 findet eine direkte Gegenüberstellung statt. Innerhalb einer Zeile finden sich Informationen über eine entsprechend am Anfang der Zeile benannte Visualisierungsbibliothek. Innerhalb von verschiedenen Spalten werden eine Auswahl an Ausschlusskriterien zur Auswahl einer Bibliothek aufgeführt. Ein durch die Farbe Rot gekennzeichnetes Feld weist auf ein nicht erfülltes Kriterium hin, ein mit Grün gekennzeichnetes Feld zeigt äquivalent dazu ein erfülltes Kriterium und ein durch die Farbe Gelb gekennzeichnetes Feld deutet auf ein teilweise erfülltes oder optionales Kriterium hin. Dabei werden als Ausschlusskriterien die Anforderungen aus Abschnitt 3.2 betrachtet.
Durch diesen Vergleich der Visualisierungsbibliotheken (siehe Abbildung 3) wird deutlich, dass grundsätzlich nur eine Verwendung von drei Bibliotheken in Frage kommt. Folglich lässt sich die Auswahl zunächst auf *jzy3d*, *Jreality* und *JOGL* eingrenzen.

	3D Scatterplots	3D Funktionsgraphen (Flächen)	Lizenz	Aktivität (Letzter Commit)	Letztes Release	Dokumentation
FXyz	Ja	Ja	GPL 3	21. Mai 16	21. Mai 16	Nur Samples
Orson Charts	Ja	Ja	GPL 3	28. Jan 16	28. Jan 16	kostenlos
jzy3d	Ja	Ja	New BSD	07. Mai 16	13. Apr 16	kostenpflichtige umfangreiche Dokumentation
Jreality	Ja	Ja	BSD	08. Jan 15	01. Apr 16	kostenlos
JOGL	Ja	Ja	BSD	16. Nov 15	10. Okt 15	kostenlos
JFreeChart	Nein	Nein	LGPL	26. Jan 16	31. Jul 14	kostenlos
jmathplot	ja	ja	keine	04. Feb 16	03. Jun 15	keine
jCharts	Nein	Nein	eigene	22. Jul 04	22. Jul 04	kostenlos
Gral	Nein	Nein	LGPL	05. Jun 16	unbekannt	kostenlos
charts4j	Nein	Nein	MIT	13. Jan 12	18. Jan 11	kostenlos
Java Plot	Nein	Nein	LGPL	05. Mrz 15	05. Mrz 15	Nur Javadoc
FreeHep 3d	Ja	ja	LGPL	05. Apr 13	05. Apr 13	kostenlos
Java Surface Plot	Nein	ja	GPL 2	22. Mrz 10	22. Mrz 10	keine
JMathtools	ja	ja	BSD	28. Aug 07	28. Aug 07	keine

Abbildung 3: Direkter Vergleich verschiedener Java Visualisierungsbibliotheken

3.6 Entscheidung

Die endgültige Entscheidung für bzw. gegen die Verwendung einer Technologie hängt allerdings von mehr als den in Abschnitt 3.5 beschriebenen Faktoren ab. In Abbildung 4 werden die bereits eingegrenzten Bibliotheken erneut gegenübergestellt. Es zeigt sich bei näherer Betrachtung der beiden Kandidaten *JOGL* und *JReality*, dass eine Umsetzung zwar möglich, aber relativ aufwändig ist. Dies liegt daran, dass die Hauptaufgaben der entsprechenden Bibliotheken nicht in der Visualisierung von Plots liegen. Vielmehr geben die Frameworks eine weit gefächerte Schnittstelle zur allgemeinen Visualisierung. Ein Großteil der dort definierten Funktionen und Methoden werden zunächst nicht benötigt. Eine Umsetzung mit einem der beiden Bibliotheken würde eine entsprechend große Einarbeitung erfordern. Außerdem müssten viele Komponenten, wie beispielsweise ein Koordinatensystem, eigens entwickelt werden. Der potentielle Aufwand wäre dementsprechend hoch. Anders verhält sich die Bibliothek *jzy3d*. Sie ist auf die Darstellung von unterschiedlichen Plots spezialisiert. Der potentielle Aufwand beschränkt sich bei Verwendung dieser Bibliothek in die Einarbeitung des Frameworks und ist damit geringer als bei den beiden anderen betrachteten Bibliotheken.



	Geforderte Features?	passende Lizenz?	Aktivität (Letzter Commit)	Letztes Release	Dokumentation	Mouseintegration	Low-Level Programmierung	Native Abhängigkeiten	potentieller Auffand	Hauptaufgabe der Lib?	Auch für 2D-Darstellungen verwendbar?	SWT Intergation möglich?
jzy3d	Ja	Ja	07. Mai 16	13. Apr 16	kostenpflichtig	Ja	Ja	ja	mittel	ja	ja	Ja
Jreality	Ja	Ja	08. Jan 15	01. Apr 16	kostenlos	Ja	Ja	ja	hoch	nein	ja	Ja
JOGL	Ja	Ja	16. Nov 15	10. Okt 15	kostenlos	Ja	Nur	ja	hoch	nein	ja	Ja

Abbildung 4: Direkter Vergleich der drei eingegrenzten Kandidaten

Die Entscheidung ergibt sich aus den oben genannten Betrachtungen. Im weiteren Verlauf der Arbeit wird mit Hilfe der open source Java-Bibliothek *jzy3d* eine Visualisierung von dreidimensionalen Daten ermöglicht.

4 Entwurf und Implementierung

Die folgenden drei Abschnitte spiegeln die Einteilung der Arbeit in die wesentlichen Schwerpunkte bzw. Aufgabenteile, die am Ende zusammenwirken müssen, wider. Die nötigen Entwürfe und Umsetzungen der einzelnen Aufgaben werden an dieser Stelle näher beschrieben.

4.1 Darstellung und Präsentation

Innerhalb der folgenden Abschnitte wird die Implementierung der Anzeige bzw. Präsentation der Daten erarbeitet. Es werden zunächst Anforderungen erläutert und die aktuelle Implementierung analysiert. Im Anschluss wird ein Konzept entwickelt, das eine Erweiterung für dreidimensionale Daten ermöglicht. Die Implementierung des entwickelten Konzepts wird im darauffolgenden Abschnitt näher erläutert.

4.1.1 Anforderungen an die Struktur der Anwendung

Eine klar definierte Struktur einer Anwendung erleichtert bei der Entwicklung einer Software nachträgliche Erweiterungen und Anpassungen. Wie in Abbildung 5 dargestellt dient die hier entwickelte API als Zwischenschicht zwischen den verwendeten Visualisierungsbibliotheken und beliebigen RCP-Anwendungen. Sie enthält sowohl eine Schnittstelle für eine RCP-Anwendung als auch die Implementierung der Darstellungen unter Verwendung der verwendeten Bibliotheken. Die Bibliothek *JFreeChart* wird aktuell bereits zur Visualisierung von zweidimensionalen Daten verwendet. Die Integration der Bibliothek *Jzy3D* in die vorhandene Struktur wird in den folgenden Kapiteln beschrieben. Die Schnittstelle zu RCP-Anwendungen sollte keinerlei Abhängigkeiten zu den unterhalb der API liegenden Bibliotheken haben. Damit wird zukünftig ermöglicht eine Bibliothek durch eine andere zu ersetzen, ohne dass sich die Schnittstelle für RCP-Anwendungen ändert.

4.1.2 Analyse der aktuellen Implementierung

Die aktuelle Implementierung der Darstellung wird in seinen Grundsätzen in Abbildung 6 dargestellt. Als Basis dient die abstrakte Klasse *FreeBaseChart2D*, die grundsätzliche Funk-

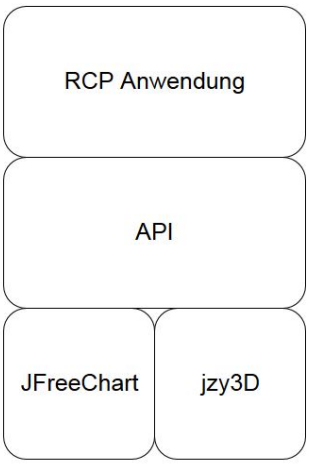


Abbildung 5: Grundsätzliche Struktur der Anwendung

tionalitäten für die Darstellung einer Visualisierung mit *JFreeChart* zur Verfügung stellt. Die Funktionen werden über das Interface *Chart* definiert. Von dieser abstrakten Implementierungsklasse erben die Klassen *FreeChart2D* und *FreeMultiChart2D*. Dabei stellt *FreeChart2D* die finale Implementierung der Visualisierung mit Hilfe von *JFreeChart* dar. Eine Alternative ist eine Darstellung von mehreren Plots nebeneinander über die Implementierung *FreeMultiChart2D*. Sie implementiert zusätzlich das Interface *MutliChart* in dem Funktionalitäten für die Darstellung von mehreren Plots definiert werden. Innerhalb der aktuellen Umsetzung werden dreidimensionale Daten zunächst nicht betrachtet. In einer nötigen Konzeptentwicklungsphase wird ein Konzept gefunden werden müssen, welches die aktuelle Umsetzung um eine dreidimensionale Darstellung erweitert.

4.1.3 Konzeptentwicklung

Zunächst ist es entscheidend ein Konzept zu entwerfen, welches alle Anforderungen und die aktuelle Implementierung mit einbezieht. In den folgenden Abschnitten wird eine geeignete Struktur beschrieben.

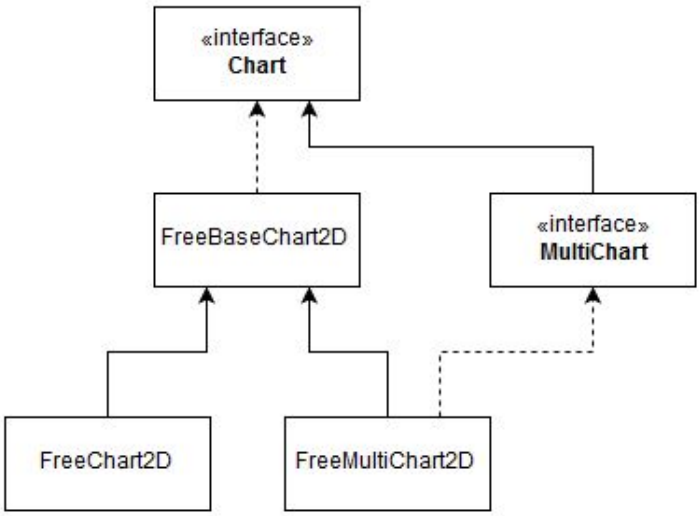


Abbildung 6: Bisheriger Aufbau der Architektur

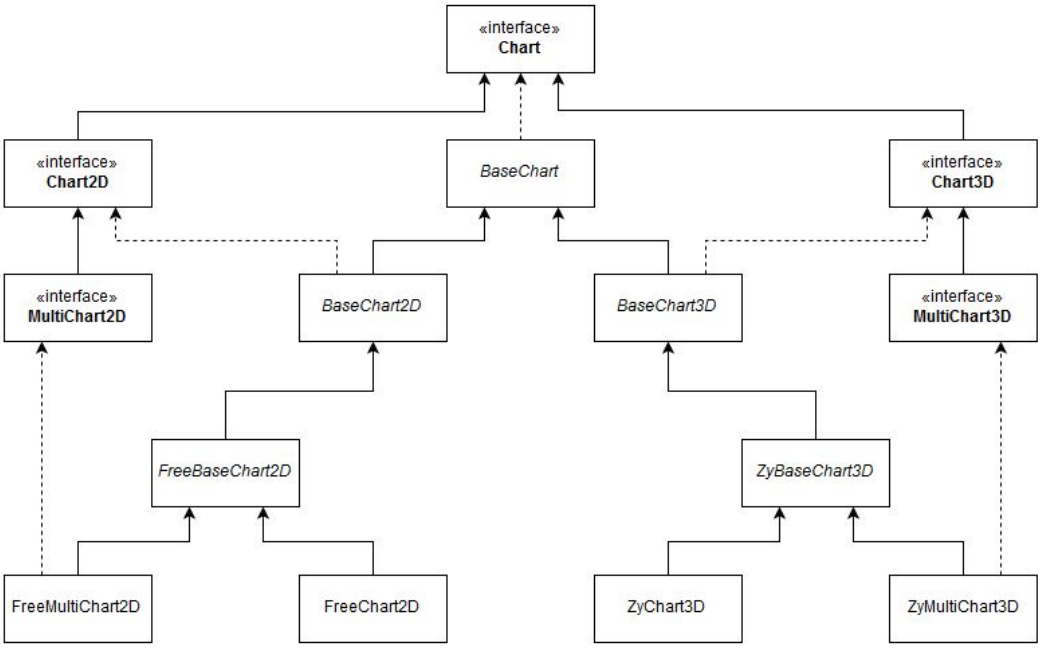


Abbildung 7: Konzept des generellen Aufbaus der Architektur

Klassenstruktur

Das entwickelte Konzept der Klassenstruktur ist schematisch in Abbildung 7 skizziert. Die Schnittstelle wird grundsätzlich, wie in der vorhergehenden Version über das Interface *Chart* definiert. Innerhalb dieses Interfaces befinden sich allgemeine Methoden zur Darstellung (vlg. Anhang A.1). Mit Hilfe der beiden Interfaces *Chart2D* und *Chart3D* werden für die entsprechende Dimension spezifische Methoden bereitgestellt. In der abstrakten Implementierungsklasse *BaseChart* werden alle Funktionen, die sowohl von der Bibliothek, als auch von der Dimension unabhängig sind, implementiert. Für den zweidimensionalen Bereich erbt *BaseChart2D* von *BaseChart* und implementiert zusätzlich *Chart2D*. Analog dazu erbt im dreidimensionalen Bereich *BaseChart3D* von *BaseChart* und implementiert zusätzlich *Chart3D*. Von *BaseChart2D* erbt *FreeBaseChart2D*, die wie in der vorherigen Version eine abstrakte Oberklasse von *FreeChart* und *FreeMultiChart2D* darstellt. Für den dreidimensionalen Bereich erbt analog zum zweidimensionalen Bereich *BaseChart3D* von *BaseChart* und implementiert *Chart3D*. Die Funktionalität für die Bibliothek *Jzy3D* wird in der darauffolgenden Ebene eingebracht. Dafür erbt *ZyBaseChart* von *BaseChart3D*. Wie in der zweidimensionalen Bibliothek wird auch an dieser Stelle zwischen einfacher Darstellung und MultiChart unterschieden. Das Interface *MultiChart3D* enthält aktuell keine Methoden, da eine Darstellung von mehreren Charts innerhalb einer Umgebung innerhalb dieser Arbeit nicht näher betrachtet wird. Es erben sowohl *ZyChart3D* als auch *ZyMultiChart3D* von *ZyBaseChart3D*. *ZyMultiChart3D* implementiert zusätzlich das Interface *MultiChart3D*.

Konfiguration eines Chart

Die Schnittstelle *Chart* (siehe Quellcode 1) stellt allgemeine Methoden eines Charts bereit. Diese dienen dazu, ein Chart erweitert zu konfigurieren. Beispielsweise soll sich mit Hilfe der Methoden *setTitle* (Zeile 3) und *showTitle* (Zeile 4) eine Überschrift definieren bzw. anzeigen lassen.

```
1 public interface Chart {
2   [...]
3   void setTitle(String title);
4   void showTitle(boolean show);
5   void setBackgroundColor(Color color);
6   void setLabel(Axis ax, String l);
7   [...]
8 }
```

Quellcode 1: Ausschnitt aus dem Interface Chart

Die Methode *setBackgroundColor* (Zeile 5) soll eine neue Hintergrundfarbe des Charts definieren. Die Achsenbeschriftung der angezeigten Achsen soll sich über die Methode *setLabel* (Zeile 6) setzen lassen. Dabei wird über den Parameter *ax* die entsprechende Achse angegeben. Das vollständige Interface befindet sich in Anhang A.1.

Konfiguration der einzelnen Plots

Die Schnittstelle definiert verschiedene Methoden zur Erzeugung, Konfiguration und Darstellung von unterschiedlichen Plots. Für die Konfiguration einer Visualisierung werden über die Schnittstelle spezielle Objekte angeboten. Sie können bei der Erzeugung einer Visualisierung als optionaler Parameter übergeben werden. Beispielsweise ist innerhalb der Schnittstelle die Funktion *addSurfacePlot* insgesamt zwei mal mit unterschiedlichen Parametern definiert (siehe Quellcode 2).

```
1 public interface Chart3D extends Chart {
2   int addSurfacePlot(PointDataset3D data);
3   int addSurfacePlot(PointDataset3D data, SurfaceConfig config);
4   int addScatterPlot(PointDataset3D data);
5   int addScatterPlot(PointDataset3D data, ScatterXYZConfig config);
6   int addFunctionPlot(FunctionDataset3D data);
7   int addFunctionPlot(FunctionDataset3D data, FunctionConfig config);
8 }
```

Quellcode 2: Das Interface Chart3D

Der Parameter *data* enthält Informationen zu den darzustellenden Werten (siehe im weiteren Verlauf *Datenhaltung* im Abschnitt 4.3.5). Die Methode mit zwei Parametern (Zeile 3) enthält

zusätzlich den optionalen Parameter *config*. Wird dieser nicht angegeben, soll der entsprechende Plot mit einer Standardkonfiguration erzeugt werden. Mit Hilfe des Konfigurationsobjekts *SurfaceConfig* können im späteren Verlauf unter anderem Farbanpassungen, Einstellungen zur Sichtbarkeit und Transparenz des zugehörigen Plots definiert werden.
Analog zu der Konfiguration von Surface Plots, sollen auch Function und Scatter Plots konfiguriert werden können. Für jeden Darstellungstyp wird hierzu eine eigene Konfigurationsklasse implementiert. Sowohl für Scatter als auch für Function Plots folgen daraus je zwei Methoden. *Chart3D* und *Chart2D* unterscheiden sich im wesentlichen durch ihr Aussehen und ihren Umfang. Diese Unterschiede sind einerseits durch die Historie der Entwicklung von *Chart2D* geschuldet, andererseits bilden beide Interfaces entsprechend ihrer Dimension unterschiedliche Funktionalitäten ab.

4.1.4 Implementierung

Die Implementierung der Darstellung und Präsentation gliedert sich in mehrere Bereiche. Diese Bereiche werden durch die folgenden Abschnitte voneinander abgegrenzt. Die Herausforderung der Implementierung besteht darin, Funktionalitäten der Bibliothek optimal auszunutzen und mit der Schnittstelle zu verbinden. Außerdem sollen über die Schnittstelle definierte Funktionalitäten, auch wenn sie von der Bibliothek primär nicht unterstützt werden, implementiert werden.

Package-Hierarchie

Die Vererbungsstrukur, die innerhalb der Konzeptentwicklung entstanden ist, lässt sich ohne Probleme umsetzten. Aus den entstandenen Klassen und Interfaces, entsteht ein Konzept der Hierarchie der Packages (vgl. Abbildung 8). Innerhalb des Packages *chart* sind Interfaces hinterlegt, die die Schnittstelle und ihre Funktionalitäten berücksichtigen. Ein entsprechendes *Chart* Objekt wird durch Verwendung einer Factory erzeugt und kann im Anschluss über das resultierende Interface verwendet werden. Mit Hilfe der von den Interfaces bereitgestellten Funktionen kann die Darstellung der Visualisierung angepasst werden. Teilweise werden den Funktionen Objekte oder Enums, die entsprechende Konfigurationen repräsentieren, als Parameter übergeben. Diese sind im Package *configuration* hinterlegt.
Die Implementierungen der Interfaces finden sich innerhalb des Packages *intern*.
Im Package *impl* finden sich dann erweiterte Implementierungen, die abhängig von der

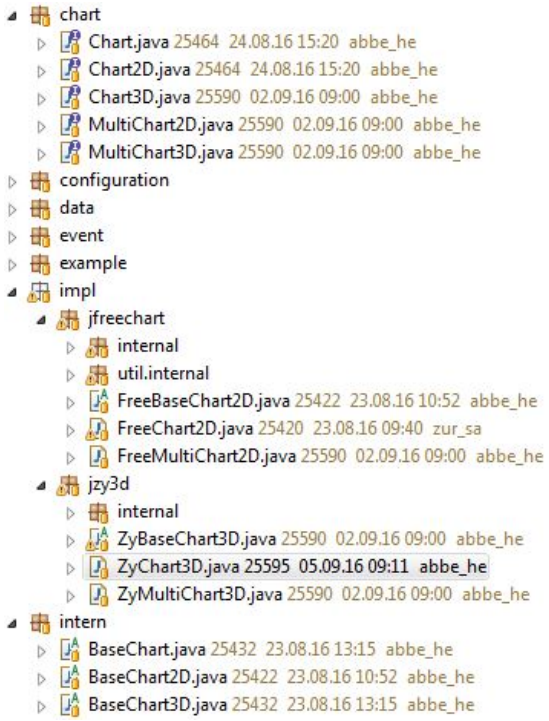


Abbildung 8: Package-Hierarchie der Implementierung

verwendeten Visualisierungsbibliothek sind. Hier finden sich die Unter-Packages *jfreechart* und *jzy3d*, in denen die zugehörigen Implementierungen bereitgestellt werden.
Innerhalb des Packages *event* werden Listener Objekte, die über die Schnittstelle registriert werden können, hinterlegt.
Das Package *data* enthält eine für alle über die Schnittstelle bereitgestellte einheitliche Implementierung für die eingehenden Daten. Innerhalb des Packages sind beispielsweise die Implementierungen *FunctionDataset2D* bzw. *FunctionDataset3D* sowie *PointDataset2D* bzw. *PointDataset3D* enthalten. Diese werden im späteren Verlauf innerhalb der Implementierung der Datenorganisation (Abschnitt 4.3.5) näher erläutert.
Im *example* Package werden unterschiedliche Beispielanwendungen bereitgestellt. Mit Hilfe dieser Beispielanwendungen lässt sich die Verwendung der Schnittstelle besser verstehen.

Ansicht

Von der Bibliothek *Jzy3D* werden zwei unterschiedliche Ansichten unterstützt: Zum einen die orthogonale Ansicht (siehe Abbildung 9), zum anderen die perspektivische Ansicht (siehe Abbildung 10). Die perspektivische Ansicht berücksichtigt, anders als die orthogonale Ansicht, dass Objekte, die vom Betrachter bzw. der Kameraposition weiter entfernt liegen, kleiner dargestellt werden müssen. Im direkten Vergleich der beiden Darstellungsformen sieht man

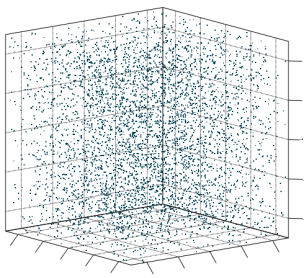


Abbildung 9: Orthogonale Ansicht

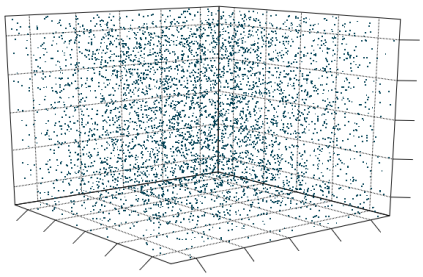


Abbildung 10: Perspektivische Ansicht

beispielsweise, dass die vom Betrachter am weitesten entfernte Achse in der perspektivischen Darstellung kürzer ist, als die anderen Achsen. Im Gegensatz dazu haben alle Achsen innerhalb der orthogonalen Darstellung die selbe Länge.

```
1 chart.getView().setCameraMode(CameraMode.PERSPECTIVE)
```

Quellcode 3: Festlegen des Darstellungsmodus auf eine perspektivische Ansicht

Das Setzen der Ansicht geschieht mit Hilfe der Methode *setCameraMode* (siehe Quellcode 3). Hierbei wird als Parameter der Enum-Wert *CameraMode.ORTHOGONAL* bzw. *CameraMode.PERSPECTIVE* für die orthogonale bzw. perspektivische Ansicht übergeben.
Innerhalb der verwendeten Bibliothek treten vereinzelt Fehler bei der Verwendung der perspektivischen Ansicht auf. In einigen Fällen werden die Beschriftungen der Achsen nicht korrekt dargestellt, in anderen Fällen ist das gesamte Koordinatensystem oder die dargestellte Funktion fehlerhaft. Solange dieser Fehler innerhalb der Bibliothek nicht behoben wurde, wird zunächst von der Schnittstelle nur eine orthogonale Ansicht unterstützt.

Chart Konfiguration

Einzelne allgemeine Konfigurationen des Charts werden über das Interface *Chart* bereitgestellt. Das Konzept sieht vor, dass unter anderem mit Hilfe der Methode *setTitle* ein Titel bzw. eine Überschrift gesetzt werden kann. Der Parameter *title* enthält an dieser Stelle den neuen Wert des Titels als String. Ein Anzeigen und Ausblenden des Titels soll über den Boolean-Parameter *show* der Methode *showTitle* ermöglicht werden. Die Bibliothek unterstützt in der verwendeten Version noch keinerlei Möglichkeit, einen Titel darzustellen. Die Herausforderung an dieser Stelle ist es also eine Implementierung für eine solche Überschrift bzw. einen Titel an die Implementierung der Bibliothek anzufügen.

Das Objekt *AWTChart* stellt, anders als *Chart*, eine Methode *addRenderer* zur Verfügung. Beispielweise kann so eine Implementierung des Interfaces *Renderer2d* mit einem *AWTChart* verknüpft werden. Ein Implementierung des Interfaces *Renderer2d* erfordert ein Überschreiben der *paint* Methode (siehe Quellcode 4), die dann innerhalb *AWTChart* aufgerufen wird.

```
1  ((AWTChart) chart).addRenderer(new Renderer2d() {
2    @Override
3    public void paint(Graphics g) {
4      if (showTitle){
5        if (currentTitleFont != null){
6          g.setFont(currentTitleFont);
7        }
8        g.setColor(currentTitleColor);
9        g.drawString(titleText, 10, 30);
10     }
11   }
12 });
```

Quellcode 4: Erzeugung eines Renderers zur Darstellung von Überschriften

Das Hinzufügen des Renders hat ein Aufrufen der *paint* Methode bei jedem Zeichnen zur Folge. Damit können Einstellungen der darzustellenden Überschrift in globalen Variablen definiert werden. Erfolgen Änderungen an diesen Konfigurationen, werden beim nächsten Aufruf der *paint* Methode und damit beim nächsten Zeichnen die Änderungen übernommen. Zunächst wird mit Hilfe der Variable *showTitle* überprüft, in wieweit eine Darstellung erwünscht ist (Zeile 4). Ist eine Darstellung erwünscht, ist der Variable der boolsche Wert *true* zugeordnet. Mit Hilfe der Instanz *g* der Umgebung *Graphics* können unterschiedliche definierte Elemente innerhalb eines Renderers dargestellt werden. Eine Darstellung von Text wird über die Methode *drawString* ermöglicht (Zeile 9). Als Argumente dieses Methodenaufrufs werden die absolute Position und der darzustellende Text *title* übergeben. Konfigurationen bezüglich der Farbe und Font des Textes werden durch Verwendung der Methoden *setFont* (Zeile 6) und *setColor* (Zeile 8) vor Aufruf der Methode *drawString* ermöglicht.

Da die Schnittstelle keinerlei Abhängigkeiten zu einer verwendeten Bibliothek haben darf, kann der Methode *setBackgroundColor* nicht direkt das Objekt als Parameter der Schnittstelle übergeben werden. Eine Konvertierung der Farbe geschieht analog zu dem im späteren Verlauf beschriebenen Beispiel zu Quellcode 6.

Innerhalb der Schnittstelle ist die Methode *setLabel* definiert. Unter Angabe einer Achse im Parameter *ax* und einem Beschriftungstext als String-Wert im Parameter *l*, lässt sich die Beschriftung der angegebenen Achse verändern. Dabei ist *ax* vom Enum-Typ *Axis*. Die

Implementierung der Funktion *setLabel* (siehe Quellcode 5), prüft in einem ersten Schritt, ob aktuell eine Visualisierung dargestellt wird (Zeile 3).

```
1  @Override
2  public void setLabel(Axis ax, String label) {
3    if (chart == null) {
4      return;
5    }
6    switch (ax) {
7      case X:
8        chart.getAxeLayout().setXAxeLabel(label);
9        break;
10     case Y:
11       chart.getAxeLayout().setYAxeLabel(label);
12       break;
13     case Z:
14       chart.getAxeLayout().setZAxeLabel(label);
15       break;
16     default:
17       break;
18   }
19 }
```

Quellcode 5: Implementierung der Funktion setLabel

Ist dies der Fall, wird der Input-Parameter *ax*, der die Achse enthält, innerhalb einer Switch-Case-Abfrage überprüft (Zeile 6). Ist eine gültige Achse eingegeben worden, kann über das *AxeLayout* die zugehörige Achse gesetzt werden. Für die X Achse geschieht dies durch Aufruf der Methode *setXAxeLabel* (Zeile 8). Für andere Achsen funktioniert dies analog.

Plot Konfiguration

Für die Konfiguration der einzelnen Plots werden entsprechende Konfigurationsobjekte implementiert, die für den entsprechenden Plot-Typ zugeschnittene Funktionalitäten und Anpassungsmöglichkeiten bereitstellen. Für Scatter Plots wird das Objekt *ScatterXYZConfig*, für Surface Plots das Objekt *SurfaceConfig* und für Function Plots das Objekt *FunctionConfig* implementiert. Das über die Schnittstelle übergebene Konfigurationsobjekt muss innerhalb der Implementierungen der Funktionen mit den Funktionalitäten der Bibliothek verbunden

werden. Am Beispiel der Methode *addFunctionPlot* wird diese Umsetzung besonders deutlich (siehe Quellcode 6).

```
1  @Override
2  public int addFunctionPlot(FunctionDataset3D data, final FunctionConfig
       config) {
3    [...]
4    final Shape surface = (Shape) tesselator.build(grid.apply(
         getMapperByDataset(data)));
5    surface.setFaceDisplayed(config.isFaceDisplayed());
6    surface.setWireframeDisplayed(config.isWireframeDisplayed());
7    surface.setWireframeColor(
8      new Color(config.getWireframeColor().getRed(),
9        config.getWireframeColor().getGreen(),
10       config.getWireframeColor().getBlue()));
11   surface.setWireframeWidth(config.getWireframeWidth());
12   chart.getScene().getGraph().add(surface, true);
13   return addDataset(data, surface);
14 }
```

Quellcode 6: Umsetzung der Konfigurationen am Beispiel einer FunctionConfig

Zunächst wird der Plot selbst erzeugt (Zeile 4). Die Bibliothek gibt bei der Erzeugung eine Instanz eines *Shape*-Objektes zurück, an dem weitere Einstellungen vorgenommen werden können. Entsprechende Setter-Methoden werden mit den Rückgabewerten der Getter-Methoden der Konfiguration aufgerufen. Diese Übersetzung ist unproblematisch, solange ein Typ eines Wert einer Konfiguration unabhängig von der Bibliothek ist. Dies ist beispielsweise im Falle von String-, Boolean- oder Integer-Werten immer erfüllt. Die Methode *setWireframeColor* erwartet jedoch als Parameter ein *Color* Objekt der verwendeten Bibliothek (Zeile 7). Da die Schnittstelle keinerlei Abhängigkeiten zu einer verwendeten Bibliothek haben darf, kann das Objekt nicht direkt als Parameter der Schnittstelle erwartet werden. Innerhalb der Schnittstelle wird ein Eclipse eigenes *Color* Objekt verwendet, welches bei Verwendung in der Bibliothek konvertiert werden muss. Objekte vom Typ *org.eclipse.swt.graphics.Color* ermöglichen im gleichen Sinne das Speichern von Farbwerten, wie das von der Bibliothek erwartete *org.jzy3d.colors.Color* Objekt. Eine Konvertierung erfolgt in der aktuellen Implementierung dadurch, dass die für die zu konvertierende Farbe anhand ihres roten, grünen und blauen Anteils in das neue Format übernommen wird. Da eine solche Konvertierung häufiger

als zunächst angenommen auftritt, wird der Konvertierungs-Algorithmus in eine separate Methode ausgelagert.
Nach vollständigem Abschluss der Konfiguration, kann das anfangs erzeugte *Shape*-Objekt an die aktuelle Darstellung (Zeile 12) und zu den anderen Datensätzen angefügt werden (Zeile 13).

4.2 Einbindung in eine RCP-Anwendung

Innerhalb dieses Abschnitts wird die Integration der Visualisierungs-Schnittstelle in ein RCP-Umfeld beschrieben. Zunächst werden die grundsätzlichen Anforderungen, die für eine solche Darstellung erforderlich sind, näher erläutert. Anschließend werden die von der Bibliothek zur Verfügung gestellten Funktionalitäten erläutert, bevor die konkrete Implementierung beschrieben wird.

4.2.1 Beschreibung der Problematik

Damit die entwickelte Anwendung in einer RCP-Anwendung lauffähig wird, muss sich die Struktur an den Komponenten des Eclipse RCP-Frameworks orientieren. Innerhalb der Eclipse-Plattform wird die Benutzeroberfläche mit Hilfe des **S**tandard **W**idget **T**oolkit (SWT) realisiert. Es stellt eine Abstraktionsschicht für die Integration von nativen Benutzeroberflächen-Elementen in die eigene Anwendung dar. Es werden Elemente des jeweiligen Betriebssystems bereitstellt und können in der eigenen Anwendung verwendet werden [5, S.3f.]. Die Benutzeroberflächenelemente, die für eine Darstellung der Visualisierung benötigt werden, müssen in ein SWT-Kompatibles Format gebracht werden, um sie in eine RCP-Anwendung integrieren zu können.

4.2.2 Funktionalitäten der Bibliothek

Neben den Hauptfunktionalitäten stellt *jzy3d* weitere optionale Methoden in getrennten Bibliotheken zur Verfügung. Beispielsweise können die Kernfunktionalitäten aus *jzy3d-api* mit Hilfe von *jzy3d-javafx* erweitert werden, sodass eine Verwendung innerhalb von *JavaFX* ermöglicht wird. Mit Hilfe der Erweiterung *jzy3d-swt* wird eine Unterstützung für SWT-Komponenten ermöglicht [15] [4, S.35]. Das Einbinden innerhalb eines SWT Kontext ist vergleichbar zu der Einbindung der zweidimensionalen Darstellung mit Hilfe der Bibliothek *JFreeChart*. Sowohl bei *jzy3d* als auch bei *JFreeChart* wird eine Einbindung in einen SWT-Kontext mit Hilfe einer so genannte SWT-Bridge ermöglicht [3, S.11].

4.2.3 Implementierung

Eine einfache Darstellung einer Visualisierung innerhalb eines SWT-Kontexts wird durch Verwendung der Erweiterung der Bibliothek erzielt. In den folgenden Abschnitten werden die entsprechenden Implementierungen vorgestellt und ggf. auftretende Probleme und deren Lösungen diskutiert.

Einbinden in einen SWT-Kontext

Das Einbinden in einen SWT-Kontext erfolgt mit Hilfe der *Bridge* innerhalb der Erweiterung *jzy3d-swt* (siehe Abschnitt 4.2.2). Diese *Bridge* stellt die Funktion *adapt(Composite containerSWT, final Component componentAWT)* bereit, mit Hilfe der sich AWT-Komponente in einen SWT Container integrieren lassen. Die Visualisierungsbibliothek *jzy3d* unterstützt bereits AWT-Komponenten. Im Quellcode 7 ist eine beispielhafte Verwendung der *Bridge* aufgezeigt.

```
1 chart = AWTChartComponentFactory.chart(Quality.Nicest, "awt");
2 Bridge.adapt(parent.getShell(), (Component) chart.getCanvas());
```

Quellcode 7: Verwendung der AWT-SWT-Bridge

Zunächst erzeugt eine Factory eine AWT-Komponente (Zeile 1). Im Anschluss wird mit Hilfe der *Bridge* das Element an die bereits vorhandene *Shell* des bereits vorhandenen SWT-Containers angefügt (Zeile 2). Ein einfaches Anfügen einer Visualisierung in einen SWT-Kontext ist damit implementiert. Aus dieser Implementierung resultiert ein Problem. Die Schnittstelle sieht es nicht vor, aufgerufene Funktionen von Benutzerinteraktionen außerhalb der Bibliothek zu konfigurieren. Benutzerinteraktionen können beispielsweise das Drücken einer Taste, das Klicken der Maus oder das Bewegen des Mausrades sein. Ein Bewegen des Mausrades hat üblicherweise ein Vergrößern oder Verkleinern der aktuellen Darstellung (auch Zoomen genannt) zur Folge. In der aktuellen Implementierung der Bibliothek wird bei Bewegung des Mausrades die Skalierung der Y-Achse geändert. Eine Konfiguration dieses Methodenaufrufs ist nicht vorgesehen. Die entstandenen Probleme können nur durch eigene Implementierungen ermöglicht werden.

Erstellen einer CustomAWTView

Innerhalb der Klasse *AWTView* ist unter anderem die Logik implementiert, die eine Skalierung bei Drehung des Mausrades hervorruft. Eine eigene Implementierung dieser Funktionalität wird mit Hilfe einer *CustomAWTView* (siehe Quellcode 8) realisiert. Dazu erbt *CustomAWTView* von *AWTView* und überschreibt die entsprechenden Methoden.

```
1  public class CustomAWTView extends AWTView {
2      protected float zoomfact = 1;
3      private static final float MIN_ZOOM_FACT = 0.5f;
4      private static final float MAX_ZOOM_FACT = 5f;
5      [...]
6      @Override
7      public void computeCameraRenderingSphereRadius(GL gl, GLU glu,
           ViewportConfiguration viewport, BoundingBox3d bounds) {
8          if (viewmode == ViewPositionMode.TOP) {
9              final float xdiam = bounds.getXRange().getRange();
10             final float ydiam = bounds.getYRange().getRange();
11             final float radius = Math.max(xdiam, ydiam) / 2;
12             cam.setRenderingSphereRadius(radius);
13             correctCameraPositionForIncludingTextLabels(gl, glu, viewport
                   );
14         } else {
15             cam.setRenderingSphereRadius((float) bounds.
                   getTransformedRadius(spaceTransformer) * zoomfact);
16         }
17     }
18
19     @Override
20     public void zoom(float factor) {
21         float newFact = zoomfact * factor;
22         if (newFact > MIN_ZOOM_FACT && newFact < MAX_ZOOM_FACT){
23           zoomfact = newFact;
24         }
25     }
26 }
```

Quellcode 8: Eigene AWT-View

Wird durch Bewegung des Mausrades die Funktion *zoom* aufgerufen (Zeile 20), hat der Parameter *factor* entweder einen Wert größer oder kleiner 1, je nachdem in welche Richtung das Mausrad bewegt wurde. Die Multiplikation dieses Faktors mit dem aktuellen absoluten Faktor ergibt den neuen absoluten Faktor, sofern der in *MIN ZOOM FACT* definierte Wert (Zeile 3) nicht unterschritten bzw. in *MAX ZOOM FACT* definierte Wert (Zeile 4) nicht überschritten wird (Zeile 22). Der aktuelle absolute Faktor wird innerhalb der globalen Variable *zoomfact* innerhalb der Klasse zur Verfügung gestellt (Zeile 2).
Für jedes Zeichnen der Visualisierung wird unter anderem die Funktion *computeCameraRenderingSphereRadius* aufgerufen (Zeile 7). Hier wird der Radius der Kamera auf den Punkt, auf den sie ausgerichtet ist, gesetzt. Wird der Radius verringert, befindet sich die Kamera näher am Objekt. Es wird folglich der von der Bibliothek berechnete und normalerweise verwendete Wert mit dem durch die *zoom* Funktion veränderten *zoomfact* multipliziert (Zeile 15).
Das Abfangen der Benutzerinteraktionen geschieht innerhalb des *CameraMouseController*s. An der entsprechenden Stelle wird jedoch in der unveränderten Implementierung der Bibliothek nicht die *zoom* Methode der *CustomAWTView* aufgerufen. Daraus folgt, dass ein Aufruf der *zoom* Funktion innerhalb eines eigenen *CustomCameraMouseControllers* (siehe Quellcode 9) erfolgen muss.

```
1 public class CustomCameraMouseController extends AWTCameraMouseController
      {
2     [...]
3     @Override
4     public void mouseWheelMoved(MouseWheelEvent e) {
5         stopThreadController();
6         final float factor = 1 + (e.getWheelRotation() / 10.0f);
7         ((CustomAWTView) chart().getView()).zoom(factor);
8     }
9 }
```

Quellcode 9: Eigener CustomCameraMouseController

Durch Vererbung der Klasse *AWTCameraMouseController* wird die Funktion *mouseWheelMoved* durch Implementierung des *MouseWheelListener*s bereitgestellt, die überschrieben wird (Zeile 4). Der Aufruf der Funktion *stopThreadController()* stoppt alle noch laufenden Animationen (Zeile 5).

Die Berechnung des Faktors erfolgt über den im *MouseWheelEvent* bereitgestellten Wert *wheelRotation*, der über die entsprechende Getter-Methode abgerufen werden kann (Zeile 6).

Eine Erzeugung von eigenen Views ist durch die Bibliothek bisher nicht vorhergesehen. Für die Erzeugung der *CustomAWTView* wird eine eigene Factory benötigt (siehe Quellcode 10).

```
1  public class CustomAWTChartComponentFactory extends
       AWTChartComponentFactory {
2
3      public static Chart chart(Quality quality, String toolkit) {
4          final CustomAWTChartComponentFactory f = new
               CustomAWTChartComponentFactory();
5          return f.newChart(quality, toolkit);
6      }
7
8      @Override
9      public View newView(Scene scene, ICanvas canvas, Quality quality) {
10         return new CustomAWTView(getFactory(), scene, canvas, quality);
11     }
12 }
```

Quellcode 10: Eigene CustomAWTChartComponentFactory

Durch Vererbung von *AWTChartComponentFactory* bleiben sämtliche Factory-Methoden erhalten, einzig die Methoden *newView* und *chart* müssen überschrieben werden. In der Funktion *newView* kann an Stelle einer *AWTView* die eigen erstellte *CustomAWTView* erzeugt werden (Zeile 10).

4.3 Effiziente Datenverwaltung

Innerhalb der folgenden Abschnitte wird beschrieben, wie die in die Schnittstelle ankommenden Daten möglichst effizient verwaltet werden können. Zunächst wird beschrieben, welche grundsätzlichen Ziele dabei verfolgt werden. Anschließend wird analysiert, wie die bisherige Implementierung die Daten verwaltet. Daraus kann in einem nächsten Schritt ein Konzept entwickelt werden, welches die effiziente Verwaltung von dreidimensionale Daten umsetzt. Abschließend wird die Implementierung beschrieben.

4.3.1 Grundsätzliche Ziele

Ein wesentliches Ziel ist es, dass beliebige RCP-Anwendungen mit Hilfe der in dieser Arbeit entwickelten Schnittstelle unterschiedliche Visualisierungen darstellen können. Die Schnittstelle selbst verwendet dazu geeignete Bibliotheken (vergleiche auch Abbildung 5). Die von einer RCP-Anwendung bereitgestellten Daten müssen vor ihrer Anzeige verwaltet und organisiert werden, sodass sie weder verloren gehen, noch redundanter Speicher belegt wird. Es ist beispielsweise nicht sinnvoll, dass die Daten an drei verschiedenen Stellen dupliziert werden. Dies tritt auf, wenn die Daten in der RCP-Anwendung, der Schnittstelle und der verwendeten Bibliothek gespeichert werden. Außerdem sollen die Daten so verwaltet werden, dass ausgewählte Datensätze nachträglich geändert werden können und diese Änderungen unmittelbar in der Darstellung sichtbar wird.
Für eine effiziente Umsetzung ist es also zunächst nötig, ein Konzept zu entwickeln. Innerhalb dieses Konzepts müssen aktuelle Implementierungen sowie Vorgaben und Beschränkungen der Bibliothek berücksichtigt werden.
Parallel zur Entwicklung der Konzepte und Implementierungen werden außerhalb dieser Arbeit zusätzlich Entwicklungen an der Schnittstelle für zweidimensionale Daten vorgenommen und vorhandene Implementierungen überarbeitet. Dies ist erforderlich, da eine Umstrukturierung der API auch hier Anpassungen erfordert.

4.3.2 Bisherige Umsetzung

In der bisherigen Umsetzung werden die über die Schnittstelle verarbeiteten Daten innerhalb von Objekten der Bibliothek *JFreeChart* gespeichert. Diese Objekte werden als *Series* bezeichnet. Es gibt beispielsweise *PointDataset2DSeries*, in denen einzelne Punkte hinterlegt werden

können. Ein *PlotDescriptor* Objekt enthält genau eine *Series* und weitere Informationen zur Id des Datensatzes bzw. des Plots. Der Zugriff auf einen *PlotDescriptor* erfolgt über eine zentrale Hashmap *plotDescriptorMap*, in der eine Id auf einen *PlotDescriptor* gemappt wird. Dies geschieht innerhalb der Klasse *FreeChart2D*.

4.3.3 Beschränkungen durch die Bibliothek

Ähnlich wie *JFreeChart* ermöglicht die Bibliothek *Jzy3D* eine Darstellung von einzelnen Punkten in Plots. Aus diesen Punkten wird innerhalb der Bibliothek ein *AbstractDrawable* erstellt, welches zu einer *Scene* angefügt werden kann. Um ein Element aus der *Scene* zu entfernen, ist es nötig, das entsprechende, zu löschende *AbstractDrawable* anzugeben. Aus einem *AbstractDrawable* lässt sich allerdings keinerlei Information zu den eingegeben Daten rekonstruieren. Aus diesem Grund ist es notwendig, die entsprechend eingegeben Daten und das daraus entstehende *AbstractDrawable* vorzuhalten.

4.3.4 Konzeptentwicklung

Unter Berücksichtigung der Beschränkungen sowie der bisherigen Umsetzung (vgl. Abschnitte 4.3.2 und 4.3.3) wird im Folgenden ein Konzept entwickelt, welches den Anforderungen beider Bibliotheken entspricht. Die Konzeptentwicklung gliedert sich in die logischen Abschnitte *Datasets*, *Listener* und *Datenhaltung*. Diese logischen Abschnitte werden im Folgenden voneinander abgegrenzt.

Datasets

In die Schnittstelle gereichten Daten werden innerhalb von *Dataset*s gespeichert. Diese *Dataset*s bekommen beim Einfügen eine eindeutige Id zugewiesen, die an die entsprechende Anwendung zurückgegeben wird. Durch diese eindeutige Id ist es nicht mehr nötig, dass eine RCP-Anwendung die Daten selbst weiter speichert. Sie kann sich den entsprechenden Datensatz über eine Funktion innerhalb der Schnittstelle unter Angabe der entsprechenden Id des gewünschten Datensatzes erneut zurückgeben lassen. Auch wenn durch die Bibliotheken eigene Datensatzobjekte bereitgehalten werden, wird an dieser Stelle eine eigene Implementierung vorgenommen. Nur so wird sichergestellt, dass die API keinerlei Abhängigkeiten zu den verwendeten Bibliotheken hat.

Listener

Die Eingabedaten werden über die API in Form von *Datasets* verwaltet. Eine Änderung eines *Datasets* sollte eine Aktualisierung der Anzeige der Visualisierung nach sich ziehen. Es muss sichergestellt werden, dass eine Veränderung innerhalb des *Datasets* entsprechend an die Darstellung weitergeleitet wird. Eine Aktualisierung sollte idealerweise durch das Ändern des *Datasets* selbst ausgelöst und nicht manuell bei jedem Ändern des *Datasets* aufgerufen werden müssen. Es ist erforderlich eine Registrierung von *DatasetListener* innerhalb des *Datasets* zu ermöglichen. Über dieses Interface kann eine entsprechende Funktion bei Änderung des *Datasets* aufgerufen werden. Da die Logik für eine Registrierung von *DatasetListener* für alle denkbaren *Datasets* identisch ist, ist es an dieser Stelle sinnvoll, eine Oberklasse zur Übernahme dieser Aufgabe zu erstellen.

Datenhaltung

Das Konzept der Datenhaltung wird in einem vereinfachten Klassendiagramm (siehe Abbildung 11) verdeutlicht. Innerhalb dieses Diagramms ist ein Teil der Vererbungshierarchie dargestellt. In einer abstrakten Oberklasse *BaseChart* wird in einer HashMap den angefügten *MutableDatasets* eine eindeutige Id zugeordnet. Die Zuordnung erfolgt mit Hilfe des inkrementellen Zählers *nextId*. Mit Hilfe der Funktionen *addDataset* und *getDataset* können entsprechende Datensätze angefügt bzw. wieder ausgegeben werden. Die Klassen *BaseChart2D* und *FreeBaseChart2D* sowie *BaseChart3D* sind nur der Vollständigkeit halber aufgeführt. Für die dreidimensionale Visualisierung wird zu den, in *BaseChart* gespeicherten Datensätzen, je ein zugehöriges *AbstractDrawable* gespeichert. Dies wird innerhalb des ebenfalls hier vorhandenen *Chart* Objekts dargestellt. Für die zweidimensionale Visualisierung werden Informationen zur Darstellung in Objekten vom Typ *Series* der Bibliothek selbst gehalten.

4.3.5 Implementierung

Die folgenden Abschnitte beschreiben die grundlegende Implementierung. Dabei sind logisch getrennte Bereiche der Implementierungen durch entsprechende Absätze gekennzeichnet.

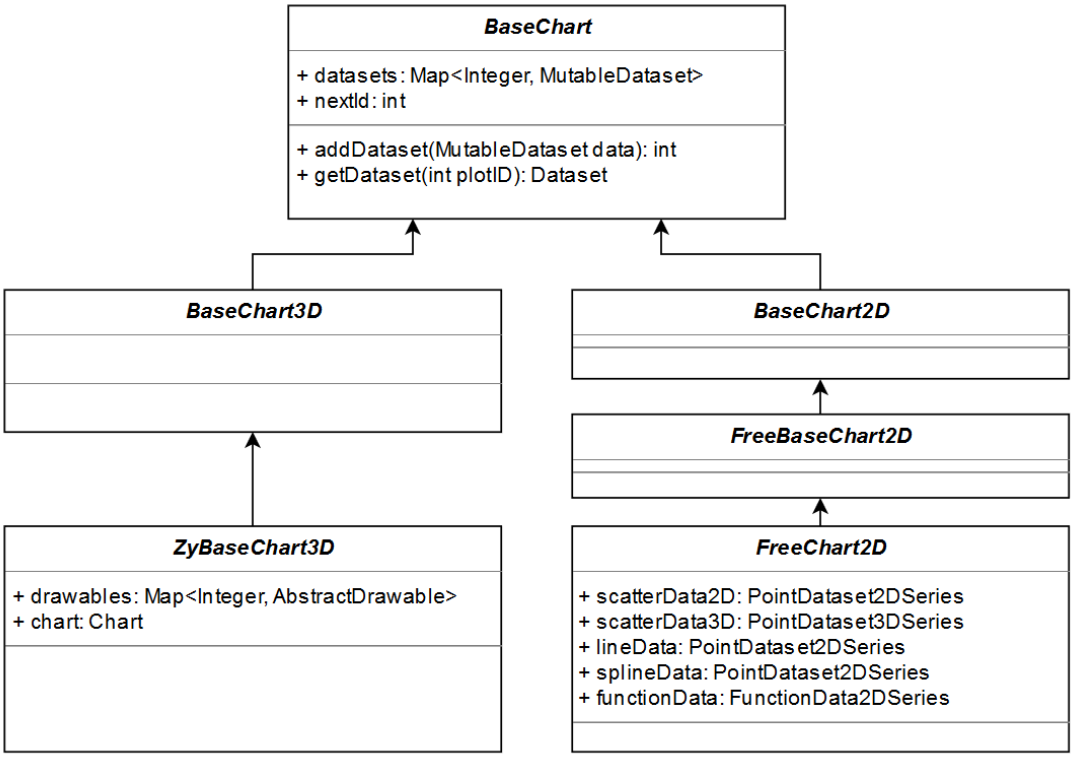


Abbildung 11: Konzept zur Speicherung von Datensätzen

Listener

Um eine Registrierung von Listenern innerhalb *Datasets* zu ermöglichen, wird zunächst eine abstrakte Klasse *MutableDataset* erstellt. In dieser wird die Funktionalität für die Registrierung der *DatasetListener* ermöglicht. Die registrierten *DatasetListener* werden innerhalb eines Hashsets hinterlegt. Das Interface *DatasetListener* definiert die Funktion *datasetChanged*. Mit Hilfe der Funktion *fireDatasetChangedEvent* (siehe Quellcode 11) wird eine Benachrichtigung aller Listener ermöglicht.

```
1 public void fireDatasetChangedEvent() {
2   for (final DatasetListener listener : listeners) {
3     listener.datasetChanged(this);
4   }
5 }
```

Quellcode 11: Die Funktion fireDatasetChangedEvent

Dazu wird über das Hashset *listeners* iteriert (Zeile 2) und die entsprechende Funktion *datasetChanged* jedes *DatasetListeners* aufgerufen (Zeile 3). Aus softwaretechnischer Sicht entspricht diese Umsetzung dem Observer-Entwurfsmuster. Es wird an dieser Stelle als Grundlage für *FunctionDatasets* und *PointDatasets* verwendet, die im folgenden beschrieben werden.

FunctionDataset

Mit Hilfe eines *FunctionDataset* (siehe Quellcode 12) können Funktionsgrafen, die über deren Funktion beschrieben werden, definiert werden.

```
1 public abstract class FunctionDataset3D extends MutableDataset {
2     [...]
3     public abstract double getValue(double x, double y);
4     [...]
5 }
```

Quellcode 12: Ausschnitt aus FunctionDataset3D

Die abstrakte Methode *getValue* errechnet einen z Wert in Abhängigkeit von x und y (Zeile 3). Sie muss im Falle einer Implementierung überschrieben werden. Innerhalb der Bibliothek existiert ein *Mapper*, der eine vergleichbare Aufgabe übernimmt. Um eine Funktion innerhalb der Visualisierung darstellen zu können, erwartet die Bibliothek eine Eingabe des bibliothekeigenen *Mapper*s. Dieser kann nicht direkt innerhalb der Schnittstelle verwendet werden, da sonst die API Abhängigkeiten zu der verwendeten Bibliothek aufweist. Die Funktion *getMapperByDataset* (siehe Quellcode 13) konvertieren ein gegebenes *FunctionDataset3D* in einen von der Bibliothek benötigten *Mapper*.

```
1 private Mapper getMapperByDataset(final FunctionDataset3D data) {
2   return new Mapper() {
3     private FunctionDataset3D d = data;
4     @Override
5     public double f(double x, double y) {
6       return d.getValue(x, y);
7     }
8   };
9 }
```

Quellcode 13: Die Funktion getMapperByDataset

Bei Erzeugung eines *Mapper*s wird ein Überschreiben der Funktion *f* erforderlich (Zeile 5). Innerhalb der Funktion wird die Anfrage nach dem z Wert des gegeben x-y-Wert-Paares an die in *d* gespeicherte Instanz des *FunctionDatasets3D* weitergeleitet (Zeile 6).

PointDataset

Ein anderes Beispiel für ein Datensatz ist ein *PointDataset3D* (siehe Quellcode 14). Innerhalb eines *PointDataset3D* können einzelne Punkte zur Darstellung beispielsweise innerhalb eines Scatter Plots gespeichert werden.

```
1  public class PointDataset3D extends MutableDataset {
2    private final String name;
3    private final List<Double> xValues = new ArrayList<>();
4    private final List<Double> yValues = new ArrayList<>();
5    private final List<Double> zValues = new ArrayList<>();
6    [...]
7      public void add(double x, double y, double z) {
8          if (!contains(x, y, z)) {
9              xValues.add(x);
10             yValues.add(y);
11             zValues.add(z);
12             fireDatasetChangedEvent();
13         }
14     }
15     [...]
16 }
```

Quellcode 14: Ausschnitt aus PointDataset3D

Innerhalb des *PointDataset3D* werden einzelne Punkte in Listen gespeichert. Für jede Dimension existiert eine eigene Liste (Zeile $3-5$). Wird beispielsweise die Methode *add* mit x-, y- und z-Wert als Parameter aufgerufen, wird zunächst geprüft, ob (x, y, z) bereits vorhanden ist (Zeile 8). Ist dies nicht der Fall, wird der x-Wert an die Liste der x-Werte angefügt (Zeile 9). Analog dazu werden die y- bzw. z-Werte an die entsprechenden Listen angefügt. Da *PointDataset3D* von *MutableDataset* erbt, wird eine Änderung des Datensatzes durch Aufruf der *fireDatasetChangedEvent* Methode gekennzeichnet (Zeile 12).

Weitere Datasets

Aktuell sind zur Zeit nur die beiden oben erwähnten *Datasets* implementiert (*PointDataset* und *FunctionDataset*). Diese Implementierungen existieren allerdings in unterschiedlichen Ausprägungen. Es sind für *PointDataset* drei verschiedene Versionen implementiert worden:

- PointDataset2D
  Speicherung von einzelnen Punkten mit genau zwei Dimensionen.
- PointDataset3D
  Speicherung von einzelnen Punkten mit genau drei Dimensionen.

- PointDatasetND
  Punkte mit mehr als drei Dimensionen können weder über *PointDataset2D* noch über *PointDataset3D* abgebildet werden. Sollen weitere Dimensionen (umgesetzt beispielsweise durch Farbkodierung) in der Visualisierung erwünscht sein, so muss auf ein *PointDatasetND* zurückgegriffen werden. Bei Erzeugung eines solchen *Dataset*s wird die Anzahl der Dimensionen gesetzt.

Für *FunctionDataset* sind zwei verschiedene Versionen implementiert worden:

- FunctionDataset2D
  Bereitstellen einer abstrakten Funktion mit einem Parameter.
- FunctionDataset3D
  Bereitstellen einer abstrakten Funktion mit zwei Parametern.

Innerhalb von wissenschaftlichen Anwendungen können mit Hilfe von *FunctionDataset*s einzelne Datensätze und Modelle verglichen werden. Wird das Ergebnis einer Berechnung beispielsweise durch verschiedene Datenpunkte mit Hilfe eines *PointDataset* beschrieben, lässt sich ein Vergleich durch eine zusätzliche Darstellung eines *FunctionDatasets* ermöglichen. Dafür muss das mathematische Modell mit dem die Punkte verglichen werden sollen implementiert werden.

5 Fazit und Ausblick

In den folgenden Abschnitten wird zunächst ein Fazit aus den erarbeiten Ergebnissen gezogen. Abgeschlossen wird das Kapitel und die Arbeit mit einem Ausblick auf weitere denkbare und mögliche, zukünftige Entwicklungen. In diesem Zusammenhang werden offene Punkte angesprochen.

5.1 Fazit

Innerhalb des vorgegebenen Bearbeitungszeitraums wurde eine vorhandene Schnittstelle für Visualisierungen von zweidimensionalen Daten um die Darstellung von dreidimensionalen Daten erweitert. Der hierbei entstandene Prototyp ermöglicht eine Darstellung von verschiedenen Darstellungstypen innerhalb einer RCP-Anwendung. Mit Hilfe der entwickelten Struktur ist die bereitgestellte Schnittstelle unabhängig von den verwendeten Bibliotheken. Folglich wird ein Austausch der Bibliotheken ohne Anpassung der sichtbaren Schnittstelle ermöglicht. Mit Hilfe der Schnittstelle kann auf die eingefügten Datensätze in Form von Datasets erneut zugegriffen werden, sodass eine effiziente Verwaltung der Daten sichergestellt wird. Auch lassen sich die eingefügten Datasets bearbeiten, welches ein Aktualisieren der Darstellung zur Folge hat. Alles in allem lässt sich feststellen, dass - entsprechend der Anforderung - ein Prototyp entstanden ist, der grundlegende Funktionalitäten bereits bereitstellt.

5.2 Ausblick

Zukünftig lassen sich weitere Funktionalitäten in den dreidimensionalen Kontext übernehmen. Es fehlt beispielsweise die Implementierung der so genannten *Point Of Interests* (POI), mit der ein Markieren von einzelnen Punkten ermöglicht wird. Auch ist eine Darstellung von mehreren Plots nebeneinander noch nicht umgesetzt. Diese Funktionalitäten wurden aus zeitlichen Gründen und geringerer Priorität nicht vollständig ausgearbeitet. Außerdem können zu den bereitgestellten Funktionen zur Erzeugung von Function, Scatter und Surface Plots zukünftig auch weitere Darstellungsformen interessant sein.

A Anhang

A.1 Das Interface Chart

```
1  public interface Chart {
2    Composite createComposite(Composite parent);
3    void setTitle(String title);
4    void showTitle(boolean show);
5    void setTitle(String title, Font font, Color color);
6    void setBackgroundColor(Color color);
7    void setMarginBackgroundColor(Color color);
8    void showLegend(boolean show);
9    void setLegendPosition(ChartComponentPosition position);
10   void saveAsPNG(File file, int width, int height) throws IOException;
11   void showGridlines(boolean show);
12   void showDomainGridlines(boolean show);
13   void showRangeGridlines(boolean show);
14   void setGridlinesStyle(GridlineStyle style);
15   void setLabel(Axis ax, String label);
16   int addScatterPlot(PointDataset3D data, ScatterXYZConfig config);
17   int addScatterPlot(PointDataset3D data);
18   void removePlot(int plotID);
19   Dataset getDataset(int plotID);
20   void showPointOfInterest(boolean show);
21   double[] getPointOfInterest();
22   boolean isPOIVisible();
23   void setPointOfInterest(double[] data);
24   void setAxisRangeAuto(Axis ax);
25   void setAxisRange(Axis ax, double lowerBound, double upperBound);
26   void setXAxisMargin(double margin);
27   void showColorBar(boolean show);
28   void setColorBarStyle(ColorBarStyle style);
29   void zoomIn(int[] mousePosition, float magnification);
30   void zoomOut(int[] mousePosition, float magnification);
31   void zoomAxisIn(Axis axis, int[] mousePosition, float magnification);
32   void zoomAxisOut(Axis axis, int[] mousePosition, float magnification);
33   void addPOIChangeListener(POIListener poindListener);
34 }
```

Literaturverzeichnis

[1] Deutsches Zentrum für Luft- und Raumfahrt: *Das DLR im Überblick*. Website, 2015. – Online erhältlich unter http://www.dlr.de/dlr/desktopdefault.aspx/tabid-10443/637_read-251; abgerufen am 20. Juli 2016.

[2] Deutsches Zentrum für Luft- und Raumfahrt: *Simulations- und Softwaretechnik*. Website, 2015. – Online erhältlich unter http://www.dlr.de/sc/einrichtung; abgerufen am 22. Juli 2016.

[3] Marquardt, Tobias: *Konzeption und Umsetzung einer Software-Bibliothek zur Visualisieriung wissenschaftlicher Daten*. Studienarbeit, 20^{th} Januar 2016. – TU Dortmund

[4] JZY3D: *Developer Documentation*. PDF-Dokument, 2013. – Online käuflich erhältlich unter http://www.jzy3d.org/guide.php; abgerufen am 18. August 2016.

[5] Ebert, Ralf: *Eclipse RCP - Entwicklung von Desktop-Anwendungen mit der Eclipse Rich Client Platform 3.7*. E-Book, 19^{th} August 2011. – Online erhältlich unter http://www.ralfebert.de/archive/eclipse_rcp/EclipseRCP.pdf; abgerufen am 29. Juli 2016.

[6] Deutsches Zentrum für Luft- und Raumfahrt: *What is RCE?* Website, 2015. – Online erhältlich unter http://rcenvironment.de/; abgerufen am 28. Juli 2016.

[7] Doreen Seider: *RCE*. Website, 2016. – Online erhältlich unter http://www.dlr.de/sc/desktopdefault.aspx/tabid-5625/9170_read-17513/; abgerufen am 28. Juli 2016.

[8] NCSS, LLC Statistical Software: *Chapter 171 - 3D Surface Plots*. PDF-Dokument, 2016. – Online erhältlich unter http://ncss.wpengine.netdna-cdn.com/wp-content/themes/ncss/pdf/Procedures/NCSS/3D_Surface_Plots.pdf; abgerufen am 29. Juli 2016.

[9] The MathWorks, Inc.: *surf 3-D shaded surface plot - Examples*. Website, 2016. – Online erhältlich unter http://de.mathworks.com/help/matlab/ref/surf.html; abgerufen am 29. Juli 2016.

[10] Utts, Jessica M.: *Seeing Through Statistics*. 3rd Edition. Thomson Brooks/Cole, 2005. – ISBN 0–534–39402–7

[11] Wicklin, Rick: *Color scatter plot markers by values of a continuous variable in SAS*. Website, 26^{th} März 2011. – Online erhältlich unter http://www.statsblogs.com/2014/03/26/color-scatter-plot-markers-by-values-of-a-continuous-variable-in-sas/; abgerufen am 29. Juli 2016.

[12] DAWN Science: *Welcome to the Data Analysis WorkbeNch*. Website, 2016. – Online erhältlich unter http://www.dawnsci.org/; abgerufen am 25. Juli 2016.

[13] DAWN Science: *Core Technologies in Dawn (Graphing package)*. Website, 2016. – Online erhältlich unter http://www.dawnsci.org/documentation/design-decisions; abgerufen am 25. Juli 2016.

[14] Phoenix Integration: *Explore*. Website, 2016. – Online erhältlich unter http://www.phoenix-int.com/modelcenter/explore.php; abgerufen am 28. Juli 2016.

[15] JZY3D: *New features in 1.0.0*. Website, 2016. – Online erhältlich unter http://www.jzy3d.org/download-1.0.0.php; abgerufen am 29. Juli 2016.