Fachbereich IV - Informatik
Lehrstuhl für Softwaretechnik
Prof. Dr. Stephan Diehl

Universität Trier

ChartFlight
Ein System zur Generierung computeranimierter Datenrundflüge

DIPLOMARBEIT

zur Erlangung des akademischen Grades
Diplom-Informatiker

eingereicht von

Rainer Lutz
Matrikel Nr. 773761

Trier, 24. November 2008

Zusammenfassung

Die vorliegende Diplomarbeit beschäftigt sich mit der Visualisierung von Benutzerdaten im dreidimensionalen Raum. Hierzu bietet das 3D Modellierungs- und Animationswerkzeug Blender die Möglichkeit, unter Verwendung der Programmiersprache Python, dreidimensionale Szenen automatisch zu erzeugen und auch zu animieren. Somit war die Grundlage für die Entwicklung des ChartFlight Webservices gelegt, dessen Entwicklung in dieser Diplomarbeit beschrieben wird. Hiermit soll dem Benutzer ermöglicht werden, eigene Daten vor einer beliebigen Hintergrundgrafik in 3D zu präsentieren. Besonders wichtig war hierbei die Tatsache, solche Präsentationen für ein breites Publikum möglichst einfach zugänglich zu machen. Dies bedeutet, dass weder eine längere Einarbeitungszeit in eine 3D Software, noch eine entsprechend leistungsstarke Hardware vonnöten sein muss, um dreidimensionale Animationen zu erzeugen.

Danksagung

An dieser Stelle möchte ich all jenen danken, die durch ihre fachliche und persönliche Unterstützung zum Entstehen dieser Diplomarbeit beigetragen und mir auf unterschiedlichste Weise in deren Verlauf zur Seite standen:

Zunächst bedanke ich mich bei Herrn Prof. Dr. Stephan Diehl für die Ausgabe des Themas und die Betreuung meiner Diplomarbeit in den vergangenen 8 Monaten.

Ebenso bedanke ich mich bei den wissenschaftlichen Mitarbeitern vom Lehrstuhl für Softwaretechnik der Universität Trier, die bei Fragen immer ein offenes Ohr für mich hatten. So begleitete Mathias Pohl den Entwicklungsprozess, Peter Weißgerber half bei der Installation und Konfiguration des Servers und Michael Burch stellte die Daten für das Fußballbeispiel zur Verfügung.

Weiterhin möchte ich meiner Familie - meiner Mutter Annette, meinem Vater Heribert und meiner Schwester Tanja - dafür danken, dass sie mir dieses Studium ermöglichten und mich stets moralisch und finanziell unterstützten.

Das Korrekturlesen der vorliegenden Diplomarbeit übernahmen meine Schwester Tanja, Felix Bott, Nicolas Mertes und Claudia Windeck, wofür ich mich an dieser Stelle recht herzlich bedanken möchte.

Ein besonderer Dank gilt meinen schwedischen Freunden Jessica und Martin und deren Familie, die mir jederzeit mit aufmunternden Worten zur Seite standen und die benötigte Erholung ermöglichten.

Inhaltsverzeichnis

1 Kapitel 1.

Einführung

1.1. Motivation und Zielsetzung

Die Grundidee dieser Diplomarbeit basiert auf den folgenden Fragen:

Wie kann man Daten in einer ansprechenden Form im dreidimensionalen Raum darstellen, ohne den Benutzer mit komplexen 3D-Modellierungswerkzeugen zu konfrontieren?
Wie kann die Renderingtechnik der breiten Öffentlichkeit zur Verfügung gestellt werden?

Genau an dieser Stelle kommt das freie 3D Modellierungs- und Animationswerkzeug Blender ins Spiel. Allerdings gibt es hierbei ein kleines Problem: die Einarbeitungszeit in Blender ist in den meisten Fällen zu groß, um lediglich die eigenen Daten zu präsentieren. Dies bedeutet, es lohnt sich meist nicht für derart einfache Grundkörper und Animationen die komplexe Steuerung einer 3D Software zu erlernen. Zudem muss eine solche Anwendung zunächst auf dem eigenen Rechner installiert werden, was zugegebenermaßen im Falle von Blender weniger ein Problem des verfügbaren Festplattenspeichers ist, sondern mehr eines der Leistung des jeweiligen Computers. Hierbei sind speziell die CPU und der Arbeitsspeicher beim Erzeugen von Bild- und Videodateien zu betrachten, aber auch die im System installierte Grafikkarte kann ein entscheidende Rolle spielen. Um genau diese Probleme zu umgehen, zeigt die vorliegende Diplomarbeit eine Möglichkeit auf, mit welcher Benutzer bequem und ohne lange Einarbeitungszeit eine ansprechende Präsentation ihrer Daten im dreidimensionalen Raum erstellen können. Solche Visualisierungen kennt man beispielsweise von diversen Wetterrundflügen im Fernsehen.
Die Idee ist es nun dem Benutzer über ein Webinterface lediglich die für die Präsentation wichtigen Einstellungen bereitzustellen und somit unwichtige Funktionen auszublenden. Dies

hat den Vorteil, dass der Benutzer tatsächlich nur mit den Einstellungen konfrontiert wird, die er auch wirklich benötigt. Das Webinterface hat zudem einen weiteren Vorteil: es kommt ohne Installation aus und lässt sich deshalb von jedem beliebigen Computer mit entsprechender Internetverbindung verwenden. Um diesen Vorteil zu wahren, werden die Daten nicht in Echtzeit mit Hilfe eines leistungsstarken Computers präsentiert, sondern in Form eines Videos, welches im Nachfolgenden auch als Präsentationsvideo bezeichnet wird. Dieses ist nun mit jedem herkömmlichen Videoplayer abspielbar, erfordert somit keine leistungsstarken Hardwarekomponenten und lässt sich zudem auch recht einfach in eine Folienpräsentation einbinden. Weiterhin wird für das Rendern des Videos nicht der lokale Rechner des Benutzer verwendet und somit blockiert, sondern ein zentraler Server. Dieser übernimmt jegliche Aufgaben beim Erzeugen des Präsentationsvideos. Alles in allem soll dem Benutzer somit eine Möglichkeit geboten werden, die eigenen Daten einfach, aber dennoch ansprechend zu präsentieren.

1.2. Aufbau der Arbeit

Zum Abschluss dieses Kapitels soll im folgenden Abschnitt der Aufbau der vorliegenden Diplomarbeit vorgestellt werden. Deren Hauptteil gliedert sich in drei größere und ein kleineres Kapitel.
Während das zweite mit dem Titel **Grundlagen** sich hauptsächlich mit dem 3D Modellierungs- und Animationswerkzeug Blender und der Verwendung von Python-Skripten beschäftigt, wird in den darauffolgenden der ChartFlight Service genauer betrachtet. Dieser wurde auf Basis der Grundlagen aus Kapitel 2 entwickelt und wird zunächst in Kapitel 3 anhand eines komplexeren Beispiels vorgestellt. Letzteres erläutert die Funktionsweise und den Prozess des Erzeugens eines neuen Präsentationsvideos und stellt somit den Service aus der Sicht eines Benutzers vor. Am Ende dieses Kapitels ist zudem ein weiteres kurzes Beispiel aufgeführt, um die Vielzahl der Möglichkeiten des ChartFlight Services aufzuzeigen.
Im Gegensatz dazu stellt Kapitel 4 die Architektur des Services vor. Hierin wird sowohl die Benutzerschnittstelle als auch das Backend genauer betrachtet und erläutert, wie diese beiden Komponenten miteinander verknüpft wurden. Letzteres besteht aus den im Hintergrund arbeitenden Python-Skripten in Verbindung mit Blender. Da bei dem Entwicklungsprozess des ChartFlight Services hin und wieder diverse Probleme auftraten, wurde diesem Punkt ein weiterer Abschnitt gewidmet. Abschließend beschäftigt sich der letzte Abschnitt des vierten Kapitels mit der Installation und Konfigurierung des ChartFlight Services.
Kapitel 5 soll erste Ergebnisse bezüglich verschiedener Leistungsdaten des ChartFlight Services präsentieren. Hierin werden unter anderem die Dauer eines Rendervorgangs und die Dateigröße des finalen Präsentationsvideos unter Berücksichtigung diverser Auflösungen gegenübergestellt. Zudem lassen sich dort mit Hilfe weiterer Messungen erste wichtige Erkenntnisse für die Berechnung einer durchschnittlichen Renderdauer bzw. Dateigröße erzielen.

Im abschließenden Kapitel 6 mit dem Titel **Zusammenfassung und Ausblick** wird, wie der Name schon sagt, der Inhalt dieser Diplomarbeit noch einmal in kurzer Form wiedergegeben. Des Weiteren findet sich hier auch ein Ausblick, mit dessen Hilfe zusätzliche Möglichkeiten zur Erweiterung des ChartFlight Services aufgezeigt werden.

2 Kapitel 2.

Grundlagen

Dieses Kapitel beschäftigt sich mit den grundlegenden Applikationen und Skripten, welche für die Entwicklung der vorliegenden Diplomarbeit verwendet wurden und beschreibt dessen Funktion und Stellenwert bei der Implementierung. Ein Großteil des Kapitels bezieht sich auf das 3D Modellierungs- und Animationswerkzeug Blender und dessen Möglichkeit zur Erweiterung mit Hilfe der Programmiersprache Python. Weiterhin sollen sowohl einige Aspekte von Java-Applets und SQLite-Datenbanken als auch die unterschiedlichen Bibliotheken zum Versenden von E-Mails vorgestellt werden. Die folgenden Kapitel beschreiben daraufhin, wie auf Basis dieser Grundlagen der ChartFlight Service entwickelt wurde.

2.1. Blender

2.1.1. Geschichte

Blender wurde Mitte bis Ende der neunziger Jahre vom Animationshaus »NeoGeo« als firmeninternes 3D Modellierungswerkzeug entwickelt. Als »NeoGeo« sich 1998 allerdings aus diesem Geschäft zurückzog, gründete Ton Roosendaal, Hauptprogrammierer von Blender, die Firma »Not a Number« zur Weiterentwicklung der Software. Sowohl »Not a Number« als auch die daraus entstandene Nachfolgefirma »NaN« mussten bald Konkurs anmelden und der Blender Sourcecode schien damit verloren. Um dem entgegen zu wirken, rief Ton Roosendaal am 18. Juli 2002 die Blender Foundation ins Leben, mit dem Ziel den Sourcecode für 100.000 Euro freizukaufen. Bereits knapp einen Monat später, am 7. September 2002, war hierfür genügend Geld vorhanden und Blender konnte daraufhin am 13. Oktober 2002 unter der GNU General Public License veröffentlicht werden. Ab diesem Zeitpunkt erlebte Blender

eine rasante Entwicklung und zählt heute ohne Frage zu den professionellsten Programmen im Open Source Bereich. [War07]

2.1.2. Funktionen und Eigenschaften

Folgende Zeilen wurden der Startseite von http://www.blender.org entnommen:

> »Blender is the free open source 3D content creation suite, available for all major operating systems under the GNU General Public License.«

Diese zugegebenermaßen relativ kurze Beschreibung spiegelt in keinster Weise den Funktionsumfang von Blender wider, liefert allerdings Informationen zur Lizenz und den unterstützten Betriebssystemen.
Blender steht unter der GNU General Public License[1], was kurz gesagt meint, dass Blender nicht nur frei und für jedermann verfügbar ist, sondern auch dessen Quellcode komplett eingesehen und ebenfalls nach eigenen Vorstellungen verändert werden darf. Wie bereits in der oben aufgeführten Beschreibung erwähnt, unterstützt Blender eine Vielzahl von verschiedenen Betriebssystemen. Nachfolgend eine Liste der wichtigsten:

- Windows 98, ME, 2000, XP oder Vista
- Mac OS X ab Version 10.2
- Linux 2.2.5 i386
- Linux 2.3.2 PPC
- FreeBSD 6.2 i386
- Irix 6.5 mips3d
- Solaris 2.8 sparc

Bei Blender handelt es sich um ein 3D Modellierungs- und Animationswerkzeug, welches eine Vielzahl von Funktionen zur Manipulation, Texturierung, Animation sowie zum Rendern des Endergebnisses zur Verfügung stellt. Letzteres beschreibt hierbei die Technik zum Generieren von Bild- oder Videodateien auf Basis einer dreidimensionalen Szene. Im Bereich der Modellierung sind die aus anderer 3D Software üblichen Funktionen, wie beispielsweise das Erzeugen und die Manipulation von Drahtgittermodellen (Meshes) oder das Verwenden von Kurvenobjekten in Form von Bézierkurven oder NURBS verfügbar. Zudem existieren auch speziellere Funktionen, wie der *Sculpt Mode*, welcher sich recht gut zum Erstellen organischer Objekte eignet. Zur Modellierung und Manipulation der Oberfläche eines Objektes werden

[1] Eine von der Free Software Foundation herausgegebene Lizenz mit Copyleft für die Lizenzierung freier Software. Hierbei bezeichnet Copyleft ein Verfahren, das Urheberrecht zu verwenden, um eine unbeschränkte Verbreitung von Kopien und veränderten Versionen eines Werkes zu ermöglichen

des Weiteren Eigenschaften wie Spiegelungen oder Transparenz mit einer ganzen Reihe von Einstellungsmöglichkeiten speziell für die Verwendung von Raytracing angeboten. Im Bereich der Texturierung stellen Funktionen, wie der *Vertex Paint Mode* oder das *UV-Mapping*, besonders mächtige Werkzeuge dar.
Neben Grundtechniken zur Animation, wie Keyframing, werden auch erweitere Funktionen zur Kinematik oder zum Animieren von Menschen und Lebewesen mit Hilfe von Skeletten zur Verfügungen gestellt. Zudem kann Blender verschiedene physikalische Effekte erzeugen, wie beispielsweise die Simulation von Feuer oder Wasser. Mit dem für die Version 2.45 neu entwickelten Partikelsystem ist es nun auch möglich, komplexere physikalische Berechnungen durchzuführen. Hierzu zählen unter anderem Schwarmsimulationen, die verbesserte Kollisionskontrolle, sowie den ebenfalls verfügbaren Möglichkeiten zur Simulation von Stoffen. Letzteres wird auch als *Cloth Simulation* bezeichnet.
Ebenfalls ist in Blender eine sogenannte *Game Engine* integriert, die es dem Benutzer erlaubt, eigene Echtzeitanwendungen zu erstellen. Diese können wahlweise auch physikalisch beeinflussbar sein. Während der Entwicklung eines quelloffenen Computerspiels unter dem Namen *Project Peach*, wurde die Game Engine komplett neu überarbeitet und ist ab der Version 2.48 verfügbar.
Auch im Bereich des Postprocessing, der Nachbearbeitung von gerenderten Bild- oder Videodateien, weist Blender reichhaltige Möglichkeiten auf. So können Videos mit dem integrierten *Video Sequence Editor* beispielsweise geschnitten oder mit diversen Effekten aufgewertet werden. Ein wesentlich mächtigeres Werkzeug stellt der sogenannte *Node Editor* dar, welcher während der Produktion des Open Movie »Elephants Dream« speziell für Blender entwickelt wurde. Hiermit kann durch geschickte Kombination von filterähnlichen Effekten ein einzelnes Bild auf viele verschiedene Arten manipuliert werden, ähnlich wie in einem Bildbearbeitungsprogramm.
Blender kann mit einer beachtlichen Anzahl an Datenformaten umgehen, so dass auch eine Weiterverarbeitung von Daten anderer Programme möglich ist oder sich aktuelle Ergebnisse dort integrieren lassen. Allerdings ist hierbei zu erwähnen, dass nicht jedes Datenformat vollständig oder in der jeweils neuesten Version unterstützt wird. Dies bringt allerdings in den meisten Fällen kaum Nachteile mit sich. Da viele der Datenformate aber über Python-Skripte sowohl importiert als auch exportiert werden, ist es möglich diese Skripte entsprechend anzupassen. Hierfür sind selbstverständlich die entsprechenden Erfahrungen mit Python und der Blender API nötig. Die einfachere Variante wäre aber ein anderes Skript zu verwenden, welches die entsprechenden Funktionen bereitstellt. Des Weiteren hat Blender mit dem eigenen Datenformat, das die Dateiendung `.blend` verwendet, eine Möglichkeit geschaffen auf ältere oder andere Projekte zugreifen zu können um beispielsweise Modelle oder Materialen zu laden. Diese lassen sich somit bequem in jede beliebige Datei einbinden.
Zum Rendern von Bilddateien oder Animationen stehen dem Benutzer auch hier eine ganze Reihe an Einstellungsmöglichkeiten zur Verfügung. Des Weiteren lässt sich, unter Zuhilfe-

nahme diverser Python-Skripte, auch auf externe Rendering-Software zurückgreifen. Intern werden zudem Techniken wie *Raytracing*, *Radiosity* oder *Ambient Occlusion* unterstützt.

2.1.3. Erweiterbarkeit von Blender

Dieser Abschnitt beschäftigt sich ein wenig genauer mit der Möglichkeit Blender mit Hilfe von Python-Skripten zu erweitern. Da es sich hierbei um einen wichtigen Bestandteil der vorliegenden Diplomarbeit handelt, gilt es im Nachfolgenden die Grundlagen dafür zu legen. Python kann grundsätzlich als imperative Programmiersprache bezeichnet werden. Dies bedeutet, dass ein Programm aus einer Folge von Befehlen besteht, welche in definierter Reihenfolge abzuarbeiten sind. Allerdings ist es auch möglich mit Python objektorientiert oder funktional zu programmieren, wobei vor allem ersteres Einzug in diese Diplomarbeit gefunden hat. Mit ihrem großen Umfang an bereits fertigen Funktionen und Klassen für alle erdenklichen Zwecke, ist Python somit die ideale Programmiersprache um Blender schnell und unkompliziert erweitern zu können. Ein weiterer Vorteil besteht darin, dass Python Programme nicht explizit kompiliert werden müssen. Dies ist vor allem in Bezug auf die vielen verschiedenen Im- und Export Skripte wichtig, welche bei Blender mitgeliefert werden. Diese lassen sich somit recht leicht, durch einfaches Ersetzen des jeweiligen Skripts, auf den neuesten Stand bringen. Hierbei ist es allerdings nicht erforderlich Blender selbst neu zu installieren. Aufgrund dessen ist es auch möglich, weitere Funktionen für Blender entweder selbst zu implementieren oder auf Entwicklungen anderer Programmierer zurückzugreifen.
Blender bietet mit einem internen Editor bereits alle Voraussetzungen um Python Skripte zu erstellen. Über diesen lässt sich auf eine ganze Reihe an Funktionen zugreifen, welche über eine umfangreiche Schnittstelle, die sogenannte Blender Python API, definiert sind. Somit ist es möglich, Blender lediglich über Python-Skripte zu steuern und auch automatisierte Vorgänge durchzuführen. Abbildung 2.1 zeigt den Python Editor von Blender 2.46.

2.1.3.1. Starten von Python Scripten

In Blender stehen gleich mehrere Möglichkeiten für das Starten eines Python-Skripts zur Verfügung. Dies ist häufig vom Verwendungszweck des jeweiligen Skripts abhängig. So werden solche mit integrierter Benutzeroberfläche über ein spezielles Fenster, das sogenannte *Scripts Window*, ausgeführt. Daraufhin erscheint im Normalfall die entsprechende Oberfläche mit deren Hilfe skriptspezifische Einstellungen gesetzt und ebenfalls durchgeführt werden können. Im Falle von Im- oder Exportskripten existiert zudem die Möglichkeit diese über das Dateimenü zu starten, was sich mit Hilfe eines speziellen Headers innerhalb der Python-Datei erreichen lässt. Hier können durch zusätzliche Dialogboxen Einstellungen gemacht werden.
Diese Diplomarbeit verfolgt allerdings einen anderen Ansatz. Die Idee hierbei ist es, Skripte ohne jegliche Benutzeroberfläche zur automatisierten Generierung dreidimensionaler Szenen

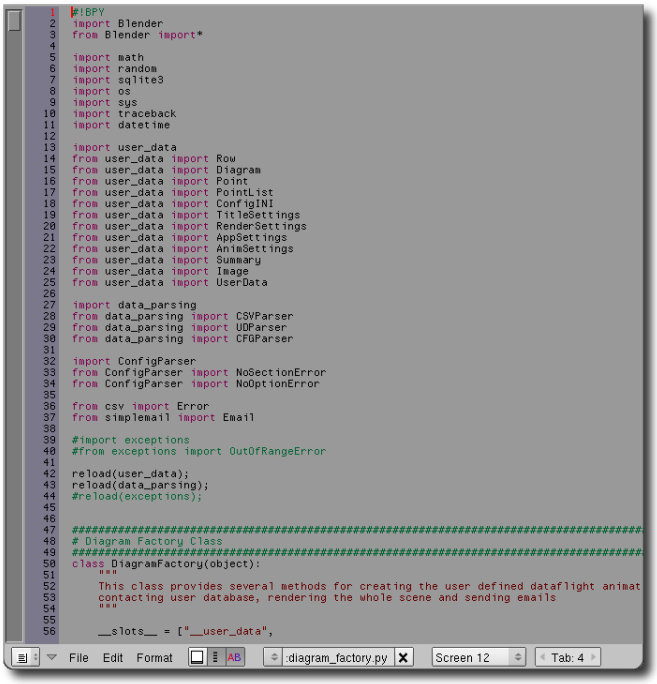

Abbildung 2.1.: Python Editor in Blender

zu verwenden. Für diesen Zweck bieten sich zwei weitere Verfahren an. Zum einen steht das automatische Ausführen von Skripten über die sogenannten *Scriptlinks* und zum anderen die Übergabe des zu startenden Skripts per Kommandozeile zur Verfügung. Beide Methoden führen das angegebene Skript sofort nach dem Starten von Blender aus. Allerdings muss für das Verwenden der ersten zunächst die entsprechende Funktionalität aktiviert, das zu ladene Skript angegeben und diese Änderung als Standardeinstellung gespeichert werden. Anschließend führt Blender beim nächsten Start die entsprechenden Python-Befehle aus. Im Gegensatz hierzu hat die Verwendung von Skripten via Kommandozeile den Nachteil, dass diese vor der Initialisierung der Oberfläche von Blender gestartet werden und somit nicht auf alle Eigenschaften zugreifen können. Hauptsächlich handelt es sich hierbei um solche, die mit Veränderungen der Oberfläche oder modifizierten Standardeinstellungen zu tun haben. Aufgrund dieses Nachteils wurde in der vorliegenden Diplomarbeit die erste Methode, das Starten eines Skripts via Scriptlink, verwendet. Hiermit ist es auch im Nachhinein möglich, diverse Einstellungen, welche nicht über die Blender Python API erreichbar sind, manuell durchzuführen.

Das Starten eines Skripts via Scriptlink soll im Nachfolgenden ein wenig genauer erläutert werden. Hierfür lässt sich das zu startende Skript zunächst im *Python Editor* von Blender öffnen, was mit Hilfe der Option *File* → *Open* des Editor-Fensters geschieht. Dadurch ist das Skript in der aktuellen Instanz von Blender verfügbar. Anschließend kann zum Panel *Scripts* gewechselt und wie in Abbildung 2.2 zu sehen, mit der Schaltfläche *Enable Script Links* deren Verwendung aktiviert werden. Als Nächstes lässt sich mit der entsprechenden

Schaltfläche ein neuer Scriptlink erzeugen und die geöffnete Python-Datei hierfür angeben. Mit der Option *OnLoad* wird Blender daraufhin mitgeteilt, dass dieses Skript immer dann starten soll, wenn Blender die aktuelle Datei öffnet. Speichert man die getätigten Einstellungen nun als Standard ab, so wird das angegebene Skript jedesmal beim Starten einer neuen Blender-Instanz ausgeführt.

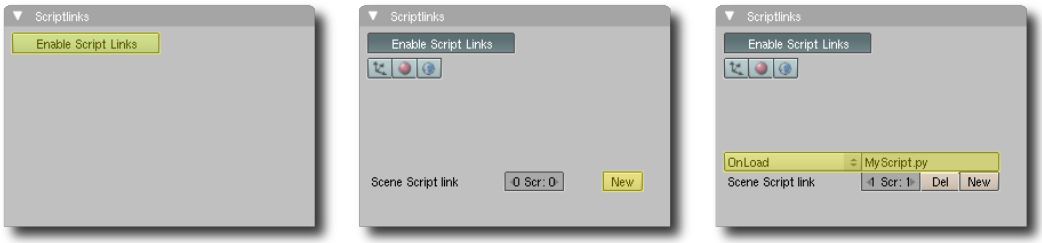

Abbildung 2.2.: Erzeugen eines Scriptlinks

Abschließend soll auf das Ausführen eines Python-Skripts und das Öffnen von Projektdateien im programmeigenen Format (.blend) via Konsole eingegangen werden. Letzteres kann besonders bei der Verwendung von Scriptlinks von Bedeutung sein. In beiden Fälle existieren verschiedene Arten dies zu. So lässt sich Blender beispielsweise normal oder lediglich im Hintergrund und ohne das Öffnen von Fenstern, unter Angabe des Parameters *-b*, starten. Das Arbeiten im Hintergrund bringt aber diverse Einschränkungen mit sich, wie den Nachteil, dass die benutzerdefinierten Standardeinstellungen hier nicht geladen werden. Dies geschieht nur beim Initialisieren der grafischen Oberfläche.
Nachfolgend wird nun das Starten eines Skripts mit Hilfe des zusätzlichen Parameters *-P* und das Öffnen einer .blend-Datei via Konsole am Beispiel gezeigt. Hierbei ist zwischen Windows und Linux Betriebssystemen zu unterscheiden. Allerdings wird davon ausgegangen, dass die Installation von Blender sich jeweils im Verzeichnis *programs* und das zu startende Python-Skript im Ordner `.blender` befindet. Während Linux diesen im entsprechenden Benutzerverzeichnis platziert, stellt Windows ihn im Installationsverzeichnis selbst zur Verfügung.

Windows

```
C:\programs\blender> blender.exe -P .blender\scripts\MyScript.py
C:\programs\blender> blender.exe C:\MyBlend.blend
```

Linux

```
/programs/blender$ blender -P ~/.blender/scripts/MyScript.py
/programs/blender$ blender ~/MyBlend.blend
```

2.1.3.2. Blender Python API

Die Schnittstelle zu Blender wird über die sogenannte *Blender Python API* geregelt, welche den Zugriff auf viele verschiedene Funktionen des 3D Modellierungs- und Animationswerkzeugs ermöglicht. Da sich diese als Grundlage des ChartFlight Services betrachten lässt, beschäftigt sich der nachfolgende Abschnitt damit.
Die Blender Python API besteht aus einem Hauptmodul mit der Bezeichnung *Blender*. Hierin sind alle wichtigen Submodule zu finden, um Blender mit Hilfe von Python-Skripten zu steuern. Die Verwendung der in der API enthaltenen Objekte und deren Methoden ist sogar ohne eine separate Python-Installation möglich. Sobald aber spezielle Funktionen von Python verwendet werden, muss die entsprechende Python-Version auf dem jeweiligen Computer verfügbar sein. Blender erkennt dies aber beim Programmstart und gibt, im Falle einer fehlenden Python-Installation, eine Warnung über die Konsole aus. Abbildung 2.3 zeigt eine Übersicht aller vorhandenen Submodule, von denen die bedeutendsten nachfolgend erläutert werden.

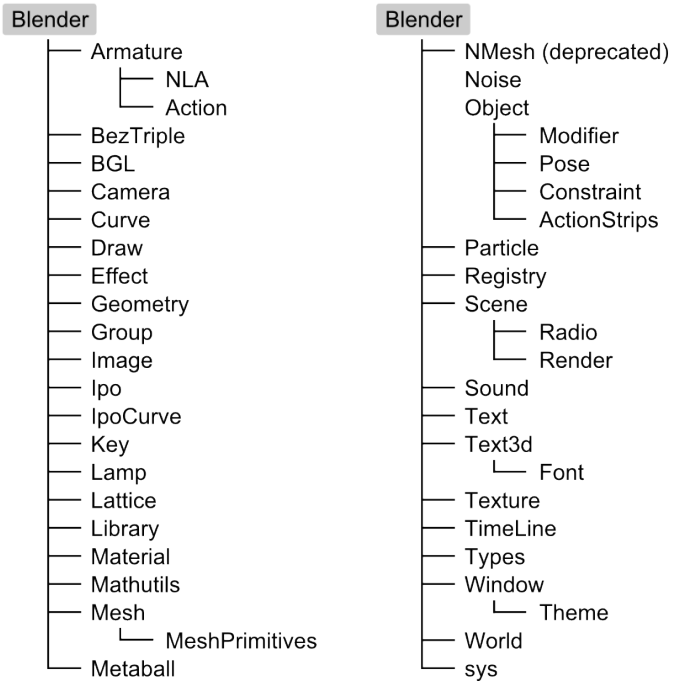

Abbildung 2.3.: Hauptmodul der Blender Python API und dessen Submodule

Die beiden wichtigsten Module zum Generieren von Objekten in Blender sind zum einen das *Mesh-* und zum anderen das *Object-Modul*. Das *Mesh-Modul* beinhaltet Klassen und Funktionen zur Speicherung der Geometrie eines Objektes. Zudem stellt es eine ganze Reihe von Methoden für den Zugriff und die Manipulation der entsprechenden Daten bereit. Somit kann beispielsweise die Position eines beliebigen Knotenpunkts, auch Vertex genannt, mit Hilfe eines Skripts verändert werden. Mit der Klasse **Object**, aus dem gleichnamigen Modul, wird

nun ein solches Mesh-Objekt gekapselt und mit zusätzlichen Eigenschaften versehen. Dies betrifft sowohl Möglichkeiten zur Transformation des gesamten Objektes als auch Einstellungen für diverse physikalische Effekte. Des Weiteren können entweder einem Object oder einem Mesh auch Informationen über dessen Oberflächeneigenschaften zugewiesen werden, welche über Klassen des Moduls *Material* verfügbar sind. Diese bieten umfangreiche Manipulationsmöglichkeiten, um auch per Python-Skript die gewünschten Effekte erzielen zu können. Sollen zudem auch Texturen eingebunden werden, geschieht dies über das Modul *Texture*. Hierin befinden sich sowohl Klassen und Funktionen zum Laden von Bildtexturen als auch zum Generieren von prozeduralen Texturen. Ein weiteres wichtiges Modul mit dem Namen *Curve*, stellt die entsprechende Schnittstelle zum Erzeugen, Speichern und Manipulieren von Kurven, wie Béziers oder NURBS, zur Verfügung. Ähnliche Aufgaben führt das Modul *Text3d* in Bezug auf Textobjekte durch, wobei dort ein zusätzliches Submodul für die Verwaltung von Schriftarten zuständig ist. Weitere essentielle Eigenschaften für Kamera, Beleuchtung und Szenenverwaltung werden mit Hilfe der Module *Camera*, *Lamp*, *Scene* und *World* bereitgestellt. Hiermit lassen sich beispielsweise verschiedene Lichtquellen erzeugen oder spezielle Umgebungseigenschaften setzen. Zu guter Letzt soll noch das Modul *Mathutils* erwähnt werden, welches grundlegende Funktionen der Linearen Algebra implementiert.

2.2. SQLite

Bei SQLite handelt es sich um eine Softwarebibliothek, welche eine relationale Datenbank und den entsprechenden Zugriff implementiert. Diese hat im allgemeinen vier wichtige Eigenschaften. Zunächst einmal ist sie in sich abgeschlossen, was heißt, dass sie nur eine minimale Unterstützung von externen Bibliotheken oder dem Betriebssystem benötigt. Aus diesem Grund ist sie auch für die Verwendung in mobilen Geräten wie Handys oder PDAs geeignet. Im Gegensatz zu anderen werden SQLite-Datenbanken nicht als ein eigenständiger Server Prozess gestartet, da deren gesamte Funktionalität in einer einzigen Datei gekapselt ist. Somit kann jede Anwendung, die auf das Dateisystem zugreifen kann, theoretisch auch eine SQLite-Datenbank kontaktieren. Dies kann sowohl Vor- als auch Nachteile mit sich bringen. Ein Vorteil und gleichzeitig eine weitere Eigenschaft wäre beispielsweise, dass SQLite-Datenbanken ohne jegliche Konfiguration sofort verwendet werden können. Die vierte wichtige Eigenschaft betrifft das sogenannte ACID-Prinzip, bei welchem alle Änderungen und Anfragen den Regeln der Atomarität, Konsistenz, Isolation und Dauerhaftigkeit folgen. SQLite unterstützt dies vollständig, sogar im Falle eines Systemabsturzes oder Stromausfalls. Der Zugriff auf SQLite-Datenbanken erfolgt mit Hilfe der Anfragesprache SQL. Hierzu werden, wie auch bei anderen Datenbanken, die meisten Standardbefehle unterstützt. So kann man unter Verwendung der Befehle `INSERT`, `UPDATE` und `DELETE` diverse Daten manipulieren oder durch Senden einer `SELECT`-Anfrage unterschiedliche Informationen filtern. Selbstverständlich stehen auch die entsprechenden Anweisungen zum Erzeugen neuer Tabellen zur

Verfügung. [Hip08]
Die entsprechenden Treiber für SQLite-Datenbanken existieren für die meisten Programmiersprachen. Somit können auch Programme unterschiedlichster Architektur eine Verbindung zu einer solchen Datenbank aufbauen und Anfragen oder Änderungen durchführen. Teilweise sind diese nicht standardmäßig verfügbar, sondern müssen im Nachhinein zusätzlich installiert werden. Nachfolgend wird nun das Kontaktieren einer SQLite-Datenbank mit Hilfe von Python, Java und PHP an einem kurzen Beispiel gezeigt, da es sich hiebei um einen wichtigen Bestandteil dieser Diplomarbeit handelt. Zu erwähnen ist allerdings noch, dass in allen Beispielen eine mögliche Fehlerbehandlung vernachlässigt wurde.
In Python sind alle benötigten Klassen für den Zugriff auf eine SQLite-Datenbank ab Version 3 bereits vorhanden. Ein solche kann mit dem nachfolgenden Code kontaktiert werden.

```
1 conn = sqlite3.connect("database.db");
2
3 cursor = conn.cursor();
4 cursor.execute("INSERT INTO MyTable VALUES (0, 'Val1', 'Val2')");
5
6 conn.close();
```

Java bietet in der Standardinstallation keine Möglichkeit auf SQLite-Datenbanken zuzugreifen, weshalb diese in Form einer entsprechenden Bibliothek zu Verfügung gestellt werden muss. Hierzu können die benötigten JDBC-Treiber von der Website

`http://www.zentus.com/sqlitejdbc/`

heruntergeladen und als zusätzliche Bibliothek eingebunden werden. Daraufhin lässt sich, wie im Nachfolgenden Listing gezeigt, auf eine Datenbank zugreifen.

```
1 conn = DriverManager.getConnection("jdbc:sqlite:database.db");
2
3 PreparedStatement stmt = conn.prepareStatement(
4                       "INSERT INTO MyTable VALUES (0, 'Val1', 'Val2')");
5 stmt.executeUpdate();
6
7 conn.close();
```

Meist sind die für PHP nötigen Datenbanktreiber, auch mit PHP Data Objects (PDO) bezeichnet, schon nach der Installation vorhanden und müssen lediglich in der Konfigurationsdatei `php.ini` aktiviert werden. Hierzu muss unter Windows entweder das Semikolon vor den Zeilen `extension=php_pdo.dll` und `extension=php_pdo_sqlite.dll` entfernt oder diese der Datei hinzugefügt werden. In Linux erscheint anstatt der Zeichenkette `.dll`

die Dateinamenserweiterung `.so`. Sind diese Bibliotheken allerdings nicht vorhanden, müssen sie zunächst entsprechend nachinstalliert werden. Wurden die PDO-Treiber hiernach korrekt erkannt, so lässt sich eine Datenbank mit dem Namen `database.db` auf folgende Art und Weise ansteuern und entsprechend der gewählten Anweisung manipulieren.

```
1 $db_handle = new PDO("database.db");
2
3 $stmt = $db_handle->prepare("INSERT INTO MyTable VALUES (0, 'Val1', 'Val2')");
4 $stmt->execute();
5
6 $db_handle = null; // close connection
```

2.3. Bibliotheken zum Versenden von E-Mails

2.3.1. SimpleMail

Bei SimpleMail handelt es sich um ein Python-Paket, mit dem es möglich ist E-Mails auf einfache Art und Weise zu versenden. Dieses Paket steht unter der GNU Lesser General Public License (LGPL), einer abgewandelten Version der GPL, welche sich speziell für die Verwendung mit Programmbibliotheken anbietet. Mit SimpleMail kann durch das einfache Angeben des Empfängers, der Zugangsdaten und des Inhalts einer E-Mail, eine solche erzeugt und mit dem Aufruf der Methode ***send()*** abgesendet werden. Dies zeigt der nachfolgende Quellcode. [Pen08]

```
1 from simplemail import Email
2
3 Email ( from_address = "sender@domain.de",
4         to_address = "recipient@domain.de",
5         subject = "My subject",
6         message = "This is my message",
7         smtp_server = "smtp.domain.com:25",
8         smtp_user = "sender",
9         smtp_password = "password" ).send();
```

2.3.2. JavaMail

Bei JavaMail handelt es sich um eine plattformunabhängige API zum Versenden von E-Mails mit Java. Da dort zunächst keine Möglichkeit besteht dies zu tun, muss JavaMail als zusätzliche Bibliothek dem aktuellen Klassenpfad hinzugefügt werden. Unter der Adresse

`http://java.sun.com/products/javamail/`

können alle benötigten Jar-Archive bezogen und daraufhin zum komfortablen Versenden von E-Mails verwendet werden. Der nachfolgende Quellcode zeigt dies anhand eines Beispiels.

```
1  Properties properties = new Properties();
2  properties.setProperty("mail.transport.protocol", "smtp");
3  properties.setProperty("mail.smtp.auth", "true");
4  properties.setProperty("mail.smtp.starttls.enable", "true");
5  properties.setProperty("mail.host", "smtp.domain.com");
6  properties.setProperty("mail.protocol.port", 25);
7
8  Session session = Session.getDefaultInstance(properties);
9
10 Message msg = new MimeMessage(session);
11 msg.setFrom( new InternetAddress("sender@domain.de") );
12 msg.addRecipient( Message.RecipientType.TO, new InternetAddress("
       recipient@domain.de") );
13 msg.setSubject("My Subject");
14 msg.setContent("This is my message", "text/plain");
15 msg.saveChanges();
16
17 Transport transport = session.getTransport();
18 transport.connect("sender", "password");
19 transport.sendMessage( msg, msg.getRecipients(Message.RecipientType.TO) );
20 transport.close();
```

Zunächst werden diverse Einstellungen bezüglich des Transportprotokolls in einem Objekt der Klasse **Properties** gespeichert. Darunter befinden sich die Art des benötigten Protokolls, die Möglichkeit der Authentifizierung und die Verschlüsslung via TLS. Ebenso wird hier der Hostname in Zeile 5 und die Portnummer in Zeile 6 angegeben. Anhand dieser Einstellungen wird nun mit der statischen Methode ***getDefaultInstance()*** der Klasse **Session** eine Instanz dieser zurückgeliefert, welche zum Erzeugen und Versenden einer E-Mail benötigt wird.
Ersteres geschieht in den Zeilen 10 bis 15. Dort wird zunächst eine neue Nachricht erzeugt, was auf Basis des in der Variablen `session` gespeicherten Objektes der gleichnamigen Klasse geschieht. Daraufhin lassen sich über die Methoden ***setFrom()*** und ***addRecipient()*** Sender und Empfänger der E-Mail angeben. Hierfür wird jeweils ein Objekt der Klasse **InternetAddress** benötigt. In den beiden nachfolgenden Zeilen können nun Betreff und Inhalt samt Textkodierung der Nachricht gesetzt und abschließend in Zeile 15 alle Änderungen permanent gespeichert werden.
Zum endgültigen Versenden der E-Mail wird zunächst eine Instanz der Klasse **Transport** benötigt. Auch diese lässt sich mit Hilfe des zuvor generierten Session-Objektes und der

Methode ***getTransport()*** erzeugen. Daraufhin wird in Zeile 18 über dieses Objekt eine Verbindung zum Mail-Server des Senders aufgebaut und diese entsprechend authentifiziert. Mit der Methode ***sendMessage()*** und der Angabe des Message-Objektes, sowie des Empfängers, kann nun die zuvor erstellte Nachricht versendet und anschließend die verwendete Instanz der Klasse **Transport** geschlossen werden.

2.4. Java-Applets

Als Java Applets bezeichnet man eine spezielle Form von Java Programmen, welche von einem Browser mit der entsprechenden Funktionalität gestartet werden können. Dazu wird das Java-Applet mit einem besonderen <applet>-Tag, ähnlich einem Bild, in den HTML-Quellcode eingebunden. Damit wird dem Browser mitgeteilt, dass es sich hierbei um ein solches handelt. Dieses wird nun auf den jeweiligen Computer heruntergeladen und in der browsereigenen Java Virtual Machine ausgeführt. Daraufhin kann das Applet verwendet werden und nahezu jede beliebige Aufgabe durchführen, sofern diese nicht gegen geltende Sicherheitsbestimmungen verstößt. Hierzu zählt beispielsweise das Lesen oder Schreiben von Dateien im Gastsystem. Nachfolgend soll anhand eines einfachen Beispiels die Funktionsweise eines Java-Applets erläutert werden.

```
1  public class SampleApplet extends JApplet {
2
3      private int locX;
4
5      public void init() {
6          this.setSize(300, 200);
7          this.locX = 10;
8      }
9
10     public void paint(Graphics g) {
11         g.clearRect(0, 0, 300, 200);
12         g.fillRect(locX, 10, 10, 10);
13
14         try { Thread.sleep(50);} catch (InterruptedException e) {}
15
16         if (this.locX < 190) {
17             this.locX++;
18             this.repaint();
19         }
20     }
21 }
```

Wie in Java üblich, müssen auch Applets mit Hilfe von Klassen realisiert werden, wobei sie entweder von `java.applet.Applet` oder `javax.swing.JApplet` abgeleitet werden. Diese Klassen bringen die nötige Funktionalität mit, um Java Applets auf Basis des *Abstract Window Toolkits (AWT)* bzw. *Swing* zu entwickeln. Hierbei spielen vor allem die Klassenmethoden ***init()***, ***paint()*** und ***repaint()*** eine wichtige Rolle. Während in der Methode ***init()*** jene Operationen implementiert werden, die lediglich einmalig beim Starten des Applets auszuführen sind, wird die ***paint()***-Methode dazu verwendet, die Oberfläche des Applets zu zeichnen. Hierbei führt die oben genannte Methode die entsprechenden Operationen immer dann aus, wenn es mit Hilfe eines Aufrufs von ***repaint()*** verlangt wird. Auf diese Weise sind nun Änderungen an der aktuellen Oberfläche und sogar Animationen möglich.
Im obigen Beispiel soll nun ein Rechteck unter Verwendung der Methoden ***paint()*** und ***repaint()*** innerhalb des Applet von links nach rechts animiert werden.
Die Position des Rechtecks wird in der Variablen `locX` gespeichert und in der ***init()***-Methode mit einem Startwert versehen. Daraufhin wird beim Ausführen des Applets automatisch die ***paint()***-Methode aufgerufen, welche die Zeichenfläche zunächst löscht und dann das Rechteck zeichnet. Nach einer kurzen Wartezeit von 50 Millisekunden in Zeile 14, wird für den Fall, dass sich das Rechteck noch innerhalb der Zeichenfläche befindet, dessen Position geändert und mit ***repaint()*** die Zeichenmethode erneut aufgerufen.

3 Kapitel 3.

ChartFlight aus der Benutzersicht

Dieses Kapitel beschreibt den ChartFlight Service aus der Sicht eines Benutzers und erklärt dessen Möglichkeiten auf einfache Art und Weise einen individuellen Auftrag zu erstellen. Im ersten Abschnitt wird die Benutzeroberfläche Schritt für Schritt beschrieben und zudem die jeweiligen Einstellungsmöglichkeiten diskutiert. Hierbei handelt es sich um eine Webseite mit diversen HTML-Formularen. Abschnitt 3.4 zeigt, auf welche Art die entsprechenden Daten, wie beispielsweise die zugrundeliegende Karte oder die jeweiligen Diagrammdaten, für die Weiterverarbeitung aufbereitet werden müssen. Daraufhin wird das Ergebnis und dessen Besonderheiten im gleichnamigen Abschnitt vorgestellt und zu guter Letzt, mit Hilfe weiterer Beispiele, die flexible Anwendung des ChartFlight Services aufgezeigt.
Die in diesem Kapitel verwendeten Daten basieren auf den Wahlergebnissen der Bundestagswahlen aus den Jahren 2002 und 2005. Entnommen wurden diese der Webseite des Bundeswahlleiters[1], auf welcher in verschiedenen Unterkategorien die Ergebnisse der entsprechenden Jahre eingesehen werden können. Das Kartenmaterial basiert auf einer dem »Wikimedia Commons Atlas of Germany«[2] entnommenen Vektorgrafik, welche speziell für dieses Beispiel aufbereitet und in eine Rastergrafik umgewandelt wurde. Ausgenommen hiervon sind die Daten aus Abschnitt 3.6, deren Ursprung jeweils explizit angegeben wird.

3.1. Der Aufbau eines Präsentationsvideos

Zum besseren Verständnis der nachfolgenden Abschnitte soll zunächst kurz der Aufbau eines Präsentationsvideos dargestellt werden. Dieses gliedert sich, je nach Einstellung, in die

[1] http://www.bundeswahlleiter.de
[2] http://commons.wikimedia.org/wiki/Atlas_of_Germany

folgenden vier bis fünf Teilsequenzen.

- Titel
- Gesamtübersicht (Intro)
- Hauptteil
- Gesamtübersicht (Outro)
- Schlussdiagramm (optional)

Der Titel bietet die Möglichkeit, dem Betrachter Informationen über die dargestellten Daten zu geben. Beispielsweise, um welche Art von Daten es sich handelt oder wo diese ihren Ursprung haben. Zudem findet sich hier Platz für den Autor und das Datum. Die Dauer des Titels kann im Teilformular für Animationseinstellungen innerhalb eines bestimmten Rahmens reguliert werden. Diese reicht von Beginn des Präsentationsvideos bis zum vollständigen Ausblenden des Titels.
Hierauf folgt anschließend die erste Gesamtübersicht, das Intro, bei welcher die komplette Karte ohne jegliche Diagramme gezeigt wird. Auch hier ist der entsprechende Zeitraum anpassbar. Um daraufhin zum Hauptteil des Präsentationsvideos zu gelangen, wird die Kamera von dieser globalen Ansicht zu der Position des ersten Diagramms bewegt. Solche lokalen Ansichten sind für jedes einzelne Diagramm verfügbar und werden nacheinander angesteuert, bis alle Diagramme abgearbeitet wurden. Die Animation eines solchen Diagramms kann zusätzlich in die Animation der einzelnen dargestellten Teildiagramme aufgeteilt werden. Ein Teildiagramm bezeichnet beispielsweise ein einzelnes Kreisdiagramm, den Ring eines Ringdiagramms oder eine Reihe von Balkendiagrammen. Die Menge aller dieser Teildiagramme ergibt somit wieder das gesamte Diagramm an einer bestimmen Stelle. Betrachtet man ein Teildiagramm genauer, so kann es ebenfalls aufgeteilt werden. Ein einzelnes Kreis-/Ringsegment oder ein Balken kann als Element eines solchen aufgefasst werden. Im Falle eines Liniendiagramms erfolgt die Aufteilung ein wenig anders. Hier kann zunächst der Hintergrund samt Achsen des Koordinatensystems als eine Einheit und danach jeweils eine Linie als einzelnes Element dieses Diagramms betrachtet werden.
Wurden alle Diagramme abgearbeitet, so erfolgt erneut eine längere Bewegung der Kamera. Diesmal allerdings vom letzten Diagramm zur globalen Gesamtübersicht, dem Outro. Hier sind nun im Gegensatz zum Intro alle generierten Diagramme zu sehen, was einen Vergleich derer ermöglicht. Sowohl die Zeit für den Kameraflug als auch die Anzeigedauer der Gesamtübersicht können entsprechend angepasst werden.
Wurde zudem ein Schlussdiagramm gewählt, so folgt dieses direkt im Anschluss an das Outro. Anderenfalls endet das Präsentationsvideo an diesem Punkt. Das Schlussdiagramm besteht aus drei Teilsequenzen. Als erstes schwenkt die Kamera ein wenig entlang der Vertikalen, um im oberen Bereich des Bildschirms Platz zu schaffen. Danach werden dort sämtliche Elemente des Schlussdiagramms eingeblendet. Abschließend definiert die letzte Sequenz, wie

lange es von diesem Zeitpunkt an dauert, bis das Ende des Präsentationsvideos erreicht ist. Alle diese Zeiträume können selbstverständlich angepasst werden.

3.2. Erstellen eines Auftrags

Das Erzeugen eines Auftrags für den ChartFlight Service erfolgt in genau acht Schritten, welche nachfolgend anhand des jeweiligen Teilformulars genauer beschrieben werden. Jedes dieser Teilformulare besitzt diverse Einstellungsmöglichkeiten zur jeweiligen Aufgabe. So lässt sich beispielsweise ein Titelbildschirm generieren oder das Aussehen der Diagramme näher bestimmen. Weiterhin befinden sich in einigen Teilformularen sogenannte **Erweiterte Einstellungen**, welche spezielle Funktionen beinhalten, um präzisere Anpassungen vorzunehmen. Diese lassen sich über die gleichnamigen Schaltflächen ein- bzw. ausblenden. Sind sie eingeblendet, so werden die getätigten Einstellungen übernommen. Für den Fall, dass die erweiterten Einstellungen aber ausgeblendet sind, werden hierfür vorgegebene Standardwerte verwendet. Dies geschieht unabhängig von zuvor durchgeführten Änderungen. Sind die erweiterten Einstellungen also geschlossen, so werden sie beim Abschicken des jeweiligen Formulars auf die Standardwerte zurückgesetzt.
Zudem besitzt jedes Formular drei wichtige Schaltflächen. Mit **Zurück** und **Weiter** kann zwischen den Formularen navigiert werden, wobei letztere die aktuellen Einstellungen speichert. Durch Betätigen von **Zur Übersicht** gelangt man zu einer Zusammenfassung des Auftrags. Diese wird in Abschnitt 3.2.8 genauer erläutert.
Bevor wir nun aber die einzelnen Teilformulare genauer betrachten, wird zunächst das Starten eines neuen Auftrags gezeigt. Hierzu lässt sich durch Betätigen der Schaltfläche **Starten** die Begrüßungsseite des Services aufrufen. Diese stellt neben diversen Informationen auch die entsprechenden Links zum Starten neuer oder Weiterführen bereits begonnener Aufträge bereit. Allerdings kann es durchaus vorkommen, dass die beiden Links, je nach Situation, nicht angezeigt werden. So besteht logischerweise nur dann die Möglichkeit einen Auftrag weiterzuführen, wenn ein solcher zuvor bereits gestartet wurde. Weiterhin besitzt der Service ein Limit für die maximale Anzahl an Aufträgen zur gleichen Zeit. Wurde es überschritten, so nimmt der Service keine weiteren Aufträge mehr entgegen. Dies hat zur Folge, dass auch der Link zum Erstellen eines neuen Auftrags nicht verfügbar ist und stattdessen ein Informationstext mit der entsprechenden Erklärung eingeblendet wird. Im Normalfall sollte aber ein Link mit dem Beschriftung *»Starten eines neuen Auftrags«* vorhanden sein, über welchen sich daraufhin ein Auftrag starten lässt.
Konnte dies erfolgreich durchgeführt werden, so folgt zunächst der Haftungsausschluss. Dieser klärt den Benutzer über verschiedene Gegebenheiten bezüglich der hochgeladenen Daten und der Benutzung des ChartFlight Services auf. Nach dessen Lektüre muss der Benutzer den aufgeführten Bedingungen zustimmen, bevor daraufhin eine weitere Schaltfläche erscheint. Mit dieser kann nun zum ersten Teilformular gewechselt werden.

3.2.1. Schritt 1 - Der Titelbildschirm

Das erste Formular bietet ein ganze Reihe von Einstellungsmöglichkeiten bezüglich des Titelbildschirms, welcher die Aufgabe hat zusätzliche Informationen über die dargestellten Daten zu liefern. Es gibt zwei Arten einen solchen Titelbildschirm zu verwenden, welche über die Auswahlbox mit dem Namen **Titeltyp** selektiert werden können.
Die erste Möglichkeit betrifft das Hochladen einer Bilddatei, welche das fertige Präsentationsvideo als Titelgrafik verwendet. Hierbei sind keine weiteren Einstellungen nötig. Ist diese Option gewählt, so erscheint ein kurzer Text, der darauf hinweist, dass die Datei zu einem späteren Zeitpunkt hochgeladen wird. Allerdings gibt es hierbei einige Richtlinien beachten, um beispielsweise Verzerrungen auszuschließen. Weitere Informationen über Titelgrafiken lassen sich den Abschnitten 3.2.5 und 3.4 entnehmen.
Als Zweites steht das Generieren eines Titelbildschirms zur Verfügung. Hiermit ist es möglich, über diverse Einstellungen, einen individuellen Titel zu erzeugen. Speziell für diese Option sind auch die **Erweiterten Einstellungen** verfügbar, welche weiter unten beschrieben werden. Zunächst betrachten wir aber die normalen Einstellungsmöglichkeiten im oberen Bereich des Formulars. Ähnlich der ersten Folie einer Beamerpräsentation lassen sich hiermit sowohl Titel, Untertitel und Autor als auch Datum und Fußzeile angeben. Die jeweiligen Beschreibungen sind lediglich in die dafür vorgesehenen Textfelder einzutragen, wobei die mit einem Stern (*) gekennzeichneten Felder ausgefüllt werden müssen. Abbildung 3.1 zeigt einen aus den vorgegebenen Eingaben generierten Titelbildschirm. Hierbei ist recht einfach zu sehen, dass standardmäßig Titel, Untertitel und Autor zentriert in der Mitte des Titelbildschirms zu finden sind. Das Datum und der Text aus dem Eingabefeld **Fußzeile** werden hingegen in der linken bzw. rechten unteren Ecke des Bildschirm positioniert.

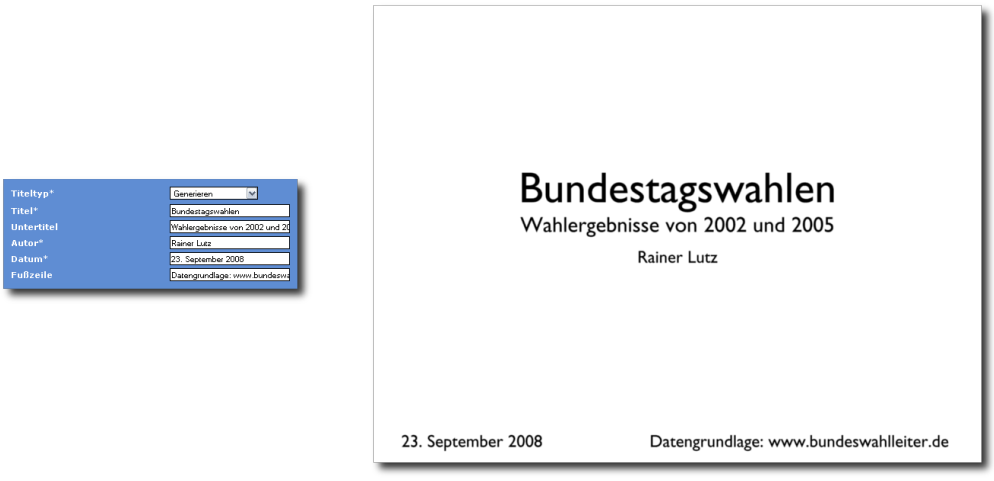

Abbildung 3.1.: generierter Titelbildschirm

Wie schon er zuvor erwähnt, existieren für den Titeltyp ***Generieren*** zusätzliche Konfigurationsmöglichkeiten, welche über die Schaltfläche **Erweiterte Einstellungen** eingeblendet

werden können. Zunächst lassen sich mit Hilfe der Eingabefelder **Textfarbe** und **Hintergrundfarbe** die für den Titelbildschirm verwendeten Farben wählen. Es gibt zwei verschiedene Arten dies zu tun. Zum einen kann, bei bekanntem Hexadezimalwert einer Farbe, diese sofort mit einer führenden Raute (#) angegeben werden. So steht beispielsweise *#000000* für Schwarz und *#FFFFFF* für Weiß. Zum anderen lässt sich aber auch der dafür vorgesehene Farbwähler mit dem Namen Tigra Color Picker[3] verwenden. Dieser wird mit dem Symbol einer Farbpalette gestartet, welches sich hinter den jeweiligen Eingabefeldern für Farben befindet. Daraufhin öffnet sich ein zusätzliches Browserfenster, in dem zwischen verschiedenen Farben gewählt werden kann. Durch Anklicken einer solchen Farbe lässt sich die Auswahl bestätigen und die Farbe als Hexadezimalwert in das entsprechende Eingabefeld eintragen. Hierbei wird dieses zusätzlich der Auswahl entsprechend eingefärbt, um auf einfache Art und Weise die aktuelle Farbe erkennen zu können. Über die Auswahlbox innerhalb des neuen Browserfensters besteht ebenso die Möglichkeit zwischen verschiedenen Farbpaletten zu wechseln, was somit ein breites Spektrum an unterschiedlichen Farben zur Verfügung stellt. Abbildung 3.2 illustriert dies anhand der beiden Farben für den Titelbildschirm.

Abbildung 3.2.: Titelbildschirm mit geänderten Farbeinstellungen

Die nächste Einstellung betrifft die Animation des Titelbildschirms. Unabhängig von Titeltyp ist dessen Hintergrundebene bereits bei Beginn des Präsentationsvideos zu sehen. Dies bedeutet, dass ein hochgeladenes Bild, welches dieser Ebene zugewiesen wurde, sofort sichtbar ist. Dabei ist es irrelevant, ob es sich bei einem solchen Bild um den gesamten Titel oder nur um eine Hintergrundgrafik handelt. Auf die zweite Option soll im weiteren Verlauf dieses Kapitels noch einmal genauer eingegangen werden. Mit der Auswahlbox **Titelanimation** lässt sich nun im Gegensatz hierzu die Animation der einzelnen Texte eines generierten Titelbildschirms kontrollieren. Hierfür stehen zwei Optionen zur Verfügung, welche im Folgenden erläutert werden. Die Option ***Alles einblenden*** bewirkt ein gleichmäßiges Einblenden aller

[3] http://www.softcomplex.com/products/tigra_color_picker/

angegebenen Texte beginnend beim ersten Bild der Animation. Auf der anderen Seite sorgt ***Zeilen einblenden*** dafür, dass die einzelnen Texte nacheinander, also zeilenweise eingeblendet werden. Das Ausblenden des Titelbildschirms ist jedoch bei jeder dieser Optionen gleich. Hier werden sowohl der Hintergrund als auch alle Texte gemeinsam ausgeblendet.
Die Auswahlbox mit der Bezeichnung **Titellayout** liefert Möglichkeiten zur Anordnung der Texte auf dem Titelbildschirm. Hier werden zwei unterschiedliche Optionen angeboten. Die erste Einstellung wurde bereits weiter oben beschrieben und positioniert Titel, Untertitel und Autor in der Mitte des Bildschirms. Datum und Fußzeile erscheinen am unteren Rand. Die zweite Option trägt den Titel ***Große Fußzeile***, welcher bereits dessen Funktion beschreibt. Hierbei wird das Datum aus der Fußzeile entfernt und mittig unter den Autor gesetzt, womit nun wesentlich mehr Platz für die tatsächliche Fußzeile vorhanden ist. Diese kann somit die gesamte Bildschirmbreite ausfüllen. Besonders empfehlenswert ist die Option somit bei übermäßig langen Texten für das Eingabefeld **Fußzeile**. Abbildung 3.3 stellt die beiden Optionen gegenüber.

Abbildung 3.3.: verschiedene Titellayouts mit Hintergrundgrafik

Die letzte Option bietet die Möglichkeit, zusätzlich zu einem generierten Titelbildschirm eine Bilddatei als Hintergrundgrafik zu verwenden. Hiermit kann beispielsweise ein Firmenlogo oder eine ähnliche Grafik unterstützend hinzugefügt werden. Dies wurde auch in Abbildung 3.3 mit dem Logo der Universität Trier getan. Abschließend ist noch erwähnen, dass sowohl Bilddateien, die als Hintergrundgrafik dienen als auch solche, die den gesamten Titelbildschirm darstellen, erst mit allen anderen Dateien im entsprechenden Upload-Formular hochgeladen werden.

3.2.2. Schritt 2 - Das Aussehen der Diagramme

Das zweiten Formular stellt verschiedene Einstellungen bezüglich des Aussehens der einzelnen Diagramme zur Verfügung. Hier lässt sich beispielsweise der Diagrammtyp oder das verwendete Farbschema wählen. Im Folgenden sollen die unterschiedlichen Optionen genauer erläutert und anhand von diversen Beispielen illustriert werden.

Mit der Auswahlbox **Diagrammtyp** lässt sich aus einer Menge von vier unterschiedlichen Arten von Diagrammen die bevorzugte wählen. Im Einzelnen sind folgende Diagrammtypen verfügbar: Kreisdiagramme, Ringdiagramme, Balkendiagramme und Liniendiagramme. Diese haben, je nach Verwendungszweck, Vor- oder Nachteile. So werden Kreisdiagramme übereinander gestaffelt dargestellt, wobei es zu Überdeckungen kommen kann. Ringdiagramme haben diesen Nachteil nicht, allerdings bieten sie im Gegenzug weniger Platz für Beschriftungen. Nachfolgend sollen diese Eigenschaften für alle Diagrammtypen detaillierter beschrieben werden. Abbildung 3.4 gibt zudem einen Überblick über die vier verschiedenen Diagrammtypen.

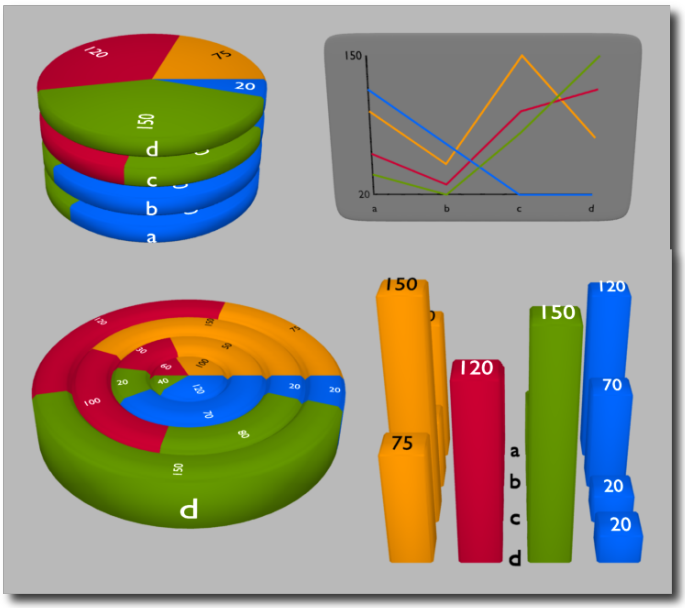

Abbildung 3.4.: verfügbare Diagrammtypen

Kreisdiagramme geben die entsprechenden Daten prozentual anhand von Kreissegmenten wieder. Um mehrere solcher Datensätze am gleichen Ort darstellen zu können, werden die einzelnen Diagramme übereinander gestaffelt. Somit bleibt mehr Platz für die Angabe von Werten und anderen Beschriftungen. Allerdings können aufgrund dessen die gesamten Daten nicht auf einem einzigen Bild präsentiert werden, sondern es bedarf in diesem Fall einer Bildsequenz oder eines Videos.

Ringdiagramme arbeiten auf eine ähnliche Art und Weise. Auch hier werden die Daten prozentual anhand von Ringsegmenten dargestellt. Im Gegensatz zu den Kreisdiagrammen lassen sich am Ende der jeweiligen Animation sämtliche visualisierten Daten überblicken. Dies bedeutet, dass bei den Ringsegmenten keine Überdeckungen auftreten. Nachteil hierbei ist der fehlende Platz, wodurch Beschriftungen verkleinert oder eventuell wieder ausgeblendet werden müssen. Somit ist es für diesen Diagrammtyp besonders wichtig, Werte auf ein

Minimum an Stellen zu reduzieren und Beschriftungen möglichst kurz zu halten.
Balkendiagramme verwenden, anders als die beiden vorangegangenen Diagrammtypen, keine prozentuale Darstellung. Hierbei wird der größte Wert eines einzelnen Datensatzes als Referenz herangezogen und in den sichtbaren Bildschirmbereich eingepasst. Somit lassen sich alle anderen Werte darauf abstimmen, was ermöglicht, auch Werte verschiedener Teildiagramme untereinander zu vergleichen. Diese Information geht bei der prozentualen Darstellung verloren. So können beispielsweise zwei Kreisdiagramme gleich große Kreissegmente derselben Farbe besitzen, obwohl die absoluten Werte völlig unterschiedlich sind. Abbildung 3.5 stellt diesen Unterschied zwischen Balken- und Kreisdiagrammen an einem abstrakten Beispiel dar. Zusätzlich werden hier jeweils die absoluten Werte des Datensatzes angezeigt, was den oben beschriebenen Nachteil ein wenig abschwächt. Um die Kreisdiagramme besser vergleichen zu können, zeigt Abbildung 3.5 diese, entgegen der normalen Darstellung, nicht übereinander, sondern nebeneinander.

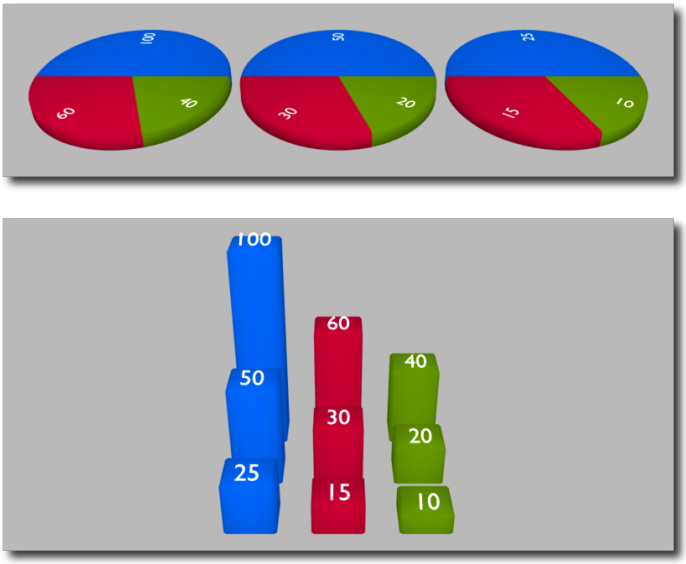

Abbildung 3.5.: Vergleich von Kreis- und Balkendiagrammen

Nachteilig bei der Verwendung von Balkendiagrammen ist allerdings, dass auch hier Überdeckungen entstehen können. Dies ist aber vom jeweiligen Datensatz abhängig und muss nicht zwingend alle Informationen verbergen. So ist beispielsweise in Abbildung 3.5 der Wert jedes Balkens zu erkennen, während in anderen Fällen einzelne Werte verdeckt sein könnten.
Liniendiagramme sind der vierte und letzte Diagrammtyp. Sie eignen sich besonders für Datensätze mit mehr als zwei Spalten, da mit ihnen ein Verlauf über die Werte dieser Spalten recht gut dargestellt werden kann. Weiterhin ist dies der einzige Diagrammtyp, für welchen auch negative Werte erlaubt sind. Aufgrund des geringen Platzes gilt es auch hier, ähnlich wie bei den Ringdiagrammen, Zahlenwerte auf ein Minimum an Stellen zu reduzieren und Beschriftungen möglichst kurz zu halten.

Die nächste Option betrifft die Farbeinstellungen. Unter dem Punkt **Diagrammfarben** können diese auf mehrere verschiedene Arten gewählt werden. Mit der Einstellung ***Zufällig*** lässt sich für jeden neuen Eintrag im gesamten Datensatz zufällig eine Farbe wählen. Hierbei hat man allerdings nur wenige Freiheiten, da die generierten Farben erst nach dem Erzeugen der Vorschaubilder zu begutachten sind. Aus diesem Grund sollten zufallsgenerierte Farben hauptsächlich zu Testzwecken verwendet werden.
Die Option mit der Bezeichnung ***Berechnen*** bietet im Gegensatz dazu zusätzliche Einstellungsmöglichkeiten. Diese werden beim Anwählen obiger Option in Form von zwei neuen Eingabefeldern für Farben eingeblendet. Hierbei ist es wichtig, möglichst unterschiedliche Farben zu wählen, da diese die Endpunkte eines Farbverlaufs darstellen. Genauer gesagt, wird zwischen den beiden angegebenen Werten die für den jeweiligen Auftrag benötigte Anzahl an Farben linear interpoliert. Je nach Datensatz kann es aber vorkommen, dass nur zwei Farben zur Verfügung stehen müssen. In diesem Fall verwendet der Service lediglich die beiden angegebenen. Bei drei Farben wird eine weitere zwischen den vorgegebenen berechnet, bei vier zwei, ... usw.
Noch mehr Kontrolle über die zu verwendenden Farben bietet die Option ***Vorgeben***. Wurde sie angewählt, so werden acht Eingabefelder für Farben eingeblendet. Hierfür sind acht vorzugsweise unterschiedliche Farbe anzugeben. Wird allerdings nur eine geringere Anzahl an Farben benötigt, so lassen sich die verbleibenden mit Hilfe des Links *»leere Farbfelder zufällig füllen«* entsprechend auffüllen. Auf diese Art und Weise ist auch eine komplett zufällige Wahl der Farben möglich. Sind nämlich alle Felder leer, so wird für jedes eine zufällige Farbe gewählt und angezeigt.
In Gegensatz dazu müssen bei der letzten Option mit dem Namen ***Auswählen*** keine Farben von Hand angegeben werden. Hier lässt sich der Unterpunkt ***Vordefiniertes Schema*** dazu verwenden, ein solches auszuwählen. Das selektierte Farbschema wird daraufhin visuell dargestellt. Abschließend illustriert Abbildung 3.6 zum besseren Verständnis die oben aufgezeigten Möglichkeiten. Von links nach rechts sind dies: zufällig gewählte Farben, berechnete Farben, vorgegebene Farben und gewähltes Farbschema.

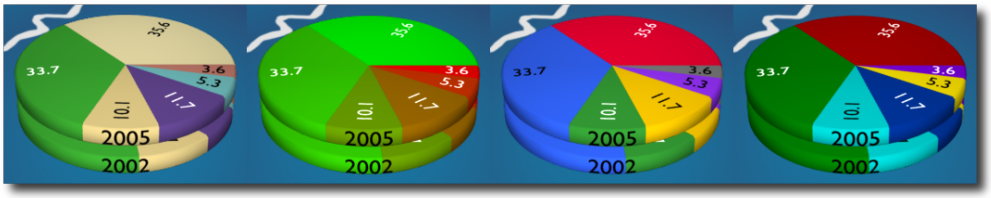

Abbildung 3.6.: Gegenüberstellung der verschiedenen Farbmodi

Mit der Option **Numerische Werte anzeigen** lässt sich angeben, ob die zugrundeliegenden Werte eines Datensatzes gezeichnet werden sollen oder nicht. Hierbei handelt es sich um die tatsächlich im Datensatz enthaltenen Werte ohne jegliche Umrechnung. Wie bereits erwähnt, kann diese Option beispielsweise im Zusammenhang mit Kreisdiagrammen von Vorteil sein.

Weitere wichtige Einstellungen betreffen die Text- und Hintergrundfarbe. Mit Hilfe der hierfür vorgesehenen Eingabefelder lassen sich beide auf die bereits vorgestellte Art und Weise ändern. Die Option **Textfarbe** bestimmt die für diverse Beschriftungen verwendete Farbe. Diese sollte so gewählt werden, dass sie sich sowohl von der hochgeladenen Bilddatei als auch von den einzelnen Diagrammfarben abhebt. Wichtig zu erwähnen ist allerdings, dass sich hiermit nur die Farbe der Beschriftungen und nicht die der dargestellten numerischen Werte wählen lässt. Auch die Textfarbe der Legende kann mit dieser Option nicht beeinflusst werden. Im Gegensatz zur **Textfarbe** ist die **Hintergundfarbe** nur dann zu sehen, wenn ein Rand der dargestellten Bilddatei erreicht wird. Kurz gesagt, mit dieser Farbe lässt sich die Umgebung des Präsentationsvideos auffüllen. Zur Verdeutlichung wurde im nachfolgenden Beispiel (Abb. 3.7) eine rote Text- und eine graue Hintergrundfarbe ausgewählt.

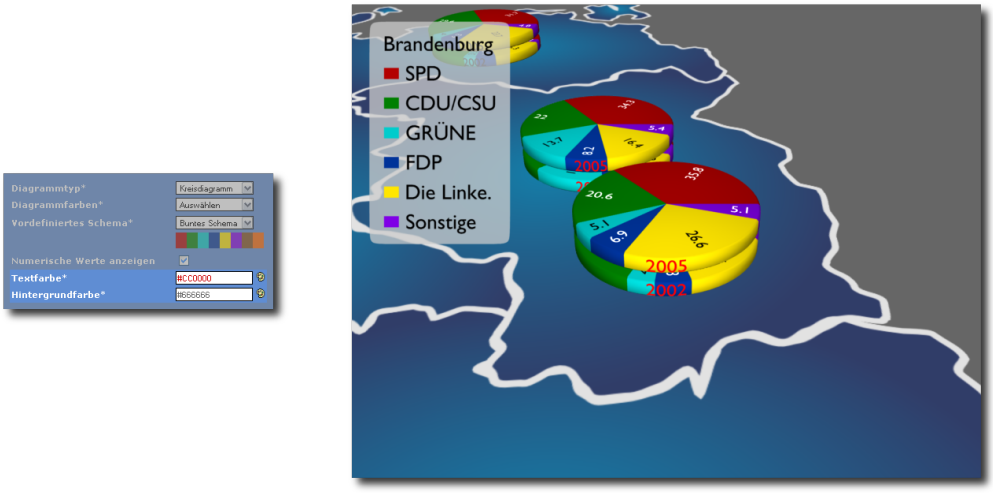

Abbildung 3.7.: Text- und Hintergrundfarbe

Bei Liniendiagrammen erscheint zudem die Einstellung **Hintergrundfarbe der Zeichenfläche**, für welche standardmäßig eine graue Farbe voreingestellt ist. Hiermit lässt sich die Farbe des Hintergrundobjekts bei Liniendiagrammen wählen, wobei diese nicht mit der oben vorgestellten Hintergrundfarbe zu verwechseln ist. Damit die dargestellten Informationen leserlich bleiben, sollte sich der angegebene Wert von den verwendeten Linienfarben und der eingestellten Textfarbe abheben.
Auch im zweiten Schritt können eine Reihe von zusätzlichen Optionen mit Hilfe der Schaltfläche **Erweiterte Einstellungen** eingeblendet und detaillierte Änderungen durchgeführt werden. Zunächst steht hierbei die Option **Material** zu Verfügung, mit deren Hilfe sich das Aussehen von Diagrammen näher bestimmen lässt. Zu Wahl stehen die Einstellungen ***Normal*** und ***Transparent***. Während die vorherigen Abbildungen allesamt ein normales Material zeigen, wird beim nachfolgenden Vergleich der **Diagrammgrößen** ein transparentes verwendet. Diese können die Werte von 1 bis 6 annehmen, wobei 1 für das kleinste und 6 für das größte Diagramm steht. Allerdings ist hierbei zu beachten, dass die angegebene Größe nur

dann zum Tragen kommt, wenn tatsächlich auch der entsprechende Platz zur Verfügung steht. So kann es beispielsweise geschehen, dass Diagramme, die sehr nahe beieinander liegen, skaliert werden müssen, um eine gegenseitige Beeinträchtigung zu verhindern. In diesem Fall lässt sich die ausgewählte Größe somit nicht einhalten. Generell sind Werte zwischen 2 und 5 zu empfehlen, wovon nur in Ausnahmefällen abgewichen werden sollte. Während 5 als Standardwert für Liniendiagramme gilt, lassen sich bei den übrigen mit der Größe 4 bessere Ergebnisse erzielen. Dies ist allerdings abhängig von der jeweiligen Karte, sowie der Verteilung der Diagramme. Abbildung 3.8 stellt die vier wichtigsten Diagrammgrößen gegenüber.

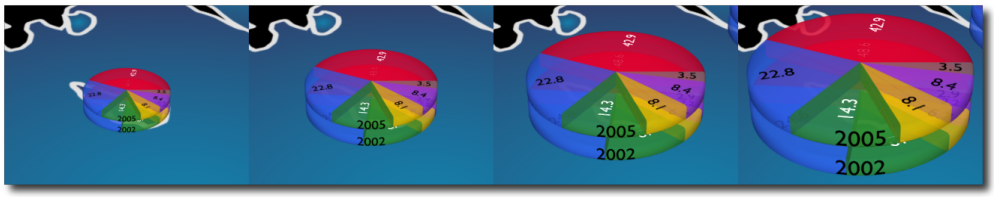

Abbildung 3.8.: Diagrammgrößen für die Werte 2 bis 5

Über **Textmodus** kann die Position der Beschreibungstexte, welche den Spaltenköpfen im jeweiligen Datensatz entsprechen, festgelegt werden. Die in dieser Auswahlbox verfügbaren Einträge ist dabei stark abhängig vom gewählten Diagrammtyp. Die Option **Standard** ist jedoch bei allen Diagrammen vorhanden und sollte in den meisten Fällen vollkommen ausreichend sein. Bei Kreisdiagrammen bieten sich zusätzlich die folgenden Möglichkeiten an: ***Zentriert***, ***Kante***, ***Rand*** und ***3D Text***. Die erste Option stellt den Text liegend und ohne 3D-Effekte in der Mitte des Kreisdiagramms dar. Hierbei ist zu beachten, dass es eventuell zu Überschneidungen mit angezeigten Wertangaben kommen kann. Gleiches gilt für die Darstellung des Textes entlang der Kante des Diagramms, da dieser ebenfalls auf der oberen Fläche des Zylinders platziert wird. Auch durch einen 3D-Text, welcher sich wiederum mittig, allerdings stehend auf dem Kreisdiagramm präsentiert, können Wertangaben verdeckt werden. Die beste Wahl in Kombination mit der Einstellung **Numerische Werte anzeigen** ist somit die Option ***Rand***. Hier wird der Beschreibungstext auf den Rand des jeweiligen Kreisdiagramms gelegt, wozu es einer entsprechenden Wölbung bedarf.
Die beiden Einstellungen ***Kante*** und ***Rand*** sind ebenfalls für Ringdiagramme verfügbar und haben auch dort ähnliche Vor- und Nachteile. Allerdings ist bei der Darstellung des Beschreibungstextes am Rand zu beachten, dass dieser von dem jeweils nächsten Ring überdeckt wird und somit am Ende der Animation nicht mehr zu sehen ist. Speziell für die Verwendung mit transparenten Materialien ist die Option ***Rand (Ausbl.)*** gedacht, welche prinzipiell das gleiche Erscheinungsbild wie die Einstellung ***Rand*** hat. Im Gegensatz dazu wird ein Beschreibungtext bei der Option ***Rand (Ausbl.)*** nach Beenden der Animation des jeweiligen Teildiagramms wieder ausgeblendet.
Für die Verwendung von Balkendiagrammen stehen, neben der Standardeinstellung, zwei weitere Optionen zur Verfügung. Mit ***Stehend*** bzw. ***Stehend (Ausbl.)*** wird ein dreidimen-

sionaler Beschreibungstext vor jeder Reihe eines Balkendiagramms positioniert. Dieser Text lässt sich, analog zu den Ringdiagrammen, auch wahlweise nach Erzeugen einer solchen Reihe von Balken ausblenden. Als Alternative hierzu existiert zudem der Eintrag ***Liegend***, bei welchem der Text zweidimensional auf der darunter liegenden Karte platziert wird. Tauchen allerdings auf dieser selbst bereits diverse Beschreibungstexte auf, so sollte man sicherheitshalber auf eine der stehenden Möglichkeiten zurückgreifen. Bei Liniendiagrammen stehen, bis auf die Option ***Standard***, keine weiteren Einstellungen zur Verfügung.
Die letzte Zeile der erweiterten Einstellungen betrifft das sogenannte **Alpha-Mapping**. Mit diesem Verfahren lässt sich eine rechteckige Form der Karte verhindern, wie in Abbildung 3.9 gezeigt. Voraussetzung hierfür ist allerdings eine Karte im Dateiformat `PNG` oder `GIF` mit einem transparenten Hintergrund.

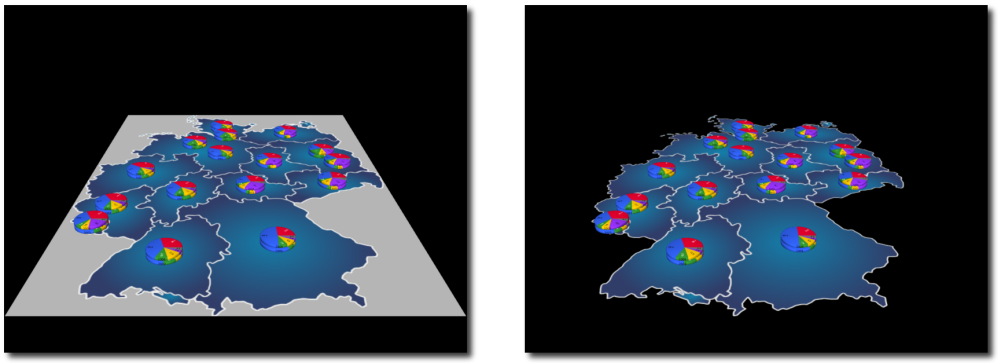

Abbildung 3.9.: Karte ohne und mit Alpha-Mapping

3.2.3. Schritt 3 - Animationen

Das dritte Teilformular bietet Einstellungen bezüglich aller verwendeten Animationen. Hier kann beispielsweise die Art des Einblendens der Diagramme oder die Dauer diverser Animationen festgelegt werden. Speziell die erweiterten Einstellungen stellen eine ganze Reihe von Optionen zur Manipulation solcher Animationszeiten zur Verfügung, was über ein besonderes Eingabefeld getan wird. Jedes dieser Eingabefelder besitzt zusätzlich zwei Schaltflächen auf der linken und zwei auf der rechten Seite. Das Eingabefeld selbst gibt die Dauer der jeweiligen Animation in Bildern pro Sekunde an. Während die Schaltflächen mit den Bezeichnungen - und - - diese Anzahl um 1 bzw. 25 verringern, erhöhen die Schaltflächen + und + + die Zahl der Bilder entsprechend. Rechts der Schaltflächen erscheint bei jedem Eingabefeld eine Umrechnung des Zahlenwerts in Sekunden, wobei eine Sekunde genau 25 Bildern entspricht. Zudem wurde für jede dieser Einstellungen ein zusätzlicher Informationstext entworfen, welcher sich durch Bewegen der Maus über das Symbol **(?)** einblenden lässt. Weiterhin wurden sämtliche Eingabefelder, abhängig von der jeweiligen Funktion, auf einen bestimmten Wertebereich eingeschränkt. Auch durch eine manuelle Eingabe bleibt es somit

unmöglich ungültige Werte zu erzwingen. Die Bedeutung der einzelnen Animationseinstellungen wird allerdings erst im zweiten Teil dieses Abschnitts genauer behandelt.
Zunächst betrachten wir aber die beiden Optionen außerhalb der erweiterten Einstellungen. Hier wird über die Auswahlbox **Animationstyp** die Art des Auftretens einzelner Diagrammteile bestimmt. Wählt man für diese den Eintrag ***Einblenden***, so verringert sich langsam die Transparenz des jeweiligen Objektes über einen gewissen Zeitraum, um einen solchen Effekt zu erzielen. Im Gegensatz dazu wird mit der Option ***Wachsend*** das entsprechende Objekt vom unteren Ende aus bis zur finalen Form aufgebaut. Diese Variante ist besonders bei Balkendiagrammen zu empfehlen, aber auch bei den anderen Diagrammtypen eine sinnvoll Alternative. Bei Liniendiagrammen ist allerdings zu beachten, dass nicht die Animation die Linien, sondern die der Hintergrundflächen gesteuert wird. Mit einer zusätzlichen Auswahlbox lässt sich im Bereich der erweiterten Einstellungen der **Animationstyp der Linien** festlegen. Hier ist zum momentanen Zeitpunkt lediglich ein einziger Eintrag mit dem Namen ***Einblenden*** verfügbar.
Die zweite Einstellung mit der Bezeichnung **Diagramme ausblenden** ist genau dann besonders sinnvoll, wenn viele Diagramme auf kleinem Raum dargestellt werden. Hier kann es durchaus vorkommen, dass sich diese gegenseitig verdecken. Ein solches Problem tritt beispielsweise dann auf, wenn zuerst ein Diagramm im vorderen Bereich und anschließend eines sehr dicht dahinter erzeugt wird. Generell sollte man deshalb darauf achten, die Diagramme im hinteren bzw. oberen Bereich der Karte als Erstes zu erzeugen. Ist dies nicht möglich, so sollte **Diagramme ausblenden** sicherheitshalber angewählt werden.
Im Nachfolgenden betrachten wir die erweiterten Einstellungen ein wenig detaillierter. Da über diese ein bestimmter Zeitraum definiert wird, lassen sich hierfür die oben vorstellten Eingabefelder verwenden. Einen Überblick hierüber gibt Abbildung 3.10, welche alle Einstellungen in Bezug auf den Diagrammtyp ***Liniendiagramm*** zeigt. Eventuelle Unterschiede werden jeweils bei den entsprechenden Optionen genauer erläutert.

Die erste Option bezieht sich auf die **Anzeigedauer des Titels**. Hier ist ein Standardwert von fünf Sekunden oder 125 Bildern pro Sekunde vorgegeben, womit die Zeit vom Beginn des Präsentationsvideos bis zum vollständigen Verschwinden des Titelbildschirms gemeint ist. Bei längeren Texten wird empfohlen diesen Zeitraum ein wenig zu erhöhen, um dem Zuschauer die Möglichkeit zu geben, alle Informationen aufnehmen zu können. Daraufhin folgt das sogenannte Intro, welches einen Überblick über die gesamte Karte ohne Diagramme bereitstellt. Mit der gleichnamigen Einstellung lässt sich nun dessen Anzeigedauer verkürzen oder verlängern. Ist die angegebene Zeit für das Intro verstrichen, so folgt die Bewegung der Kamera zum ersten Diagramm. Auch diese wird über die entsprechende Option mit dem Namen **Flugzeit zur Karte** manipuliert. Nachdem nun alle Diagramme erzeugt wurden, erfolgt erneut eine ähnliche Animation. Diesmal bewegt sich die Kamera allerdings vom letzten Diagramm zur Gesamtübersicht. Auch in diesem Fall lässt sich Konfiguration der Flugzeit über das entsprechende Eingabefeld vornehmen. Anschließend befindet sich die Kamera wieder

Abbildung 3.10.: Erweiterte Einstellungen mit eingeblendetem Hilfetext

in der Gesamtübersicht. Als Gegenstück zum Intro existiert hier das sogenannte Outro, mit welchem die Anzeigedauer der gesamten Karte mit allen generierten Diagrammen gemeint ist. Unter **Anzeigedauer des Outros** kann diese entsprechend den eigenen Vorstellungen verändert werden.

Die weiteren Einstellungen beziehen sich auf die Animationen eines einzelnen Diagramms und dessen Komponenten. Eine Besonderheit stellt allerdings die Option **Bewegungszeit der Kamera (frames/BU)** dar. Hiermit lässt sich nicht die gesamte Bewegungszeit der Kamera zwischen zwei Diagrammen einstellen, sondern lediglich ein Grundwert angeben. Dieser definiert, wie viele Bilder verwendet werden sollen, um die Kamera genau eine Einheit weiter zu bewegen. Es handelt sich hierbei um die Grundeinheit von Blender, welche auch unter der Bezeichnung *Blender Unit* bekannt ist. Zudem sollte zur besseren Vorstellung erwähnt werden, dass die kürzeste Seite einer Karte immer die Länge 10 solcher Einheiten besitzt. Je nach Entfernung zweier Punkte, kann der angegebene Wert somit mehrere Male multipliziert werden. Deshalb ist es ratsam, nicht zu große Änderungen an diesem Wert vorzunehmen.

Trifft die Kamera an einer neuen Diagrammposition ein, so kann es sinnvoll sein, die Animation dieses Diagramms nicht sofort zu starten. Dadurch kann sich der Betrachter zunächst orientieren und wird nicht mit der startenden Animation überfordert. Hierfür steht das Eingabefeld **Wartezeit vor Diagrammanimation** zur Verfügung und lässt sich auf die oben beschriebene Weise manipulieren. Eine weitere Option ist die **Animationsdauer eines Elements**, welche Abbildung 3.10 allerdings nicht zeigt. Es handelt sich hierbei um die Zeit, die zum Einblenden eines einzelnen Kreis-/Ringsegments, eines Balkens oder aber eines Beschreibungstextes benötigt wird. Für den Fall, dass Liniendiagramme gewählt wurden, erscheint an dieser Stelle ein Eingabefeld für die **Animationsdauer der Zeichenfläche**. Speziell für

Liniendiagramme stehen zusätzlich im unteren Bereich der erweiterten Einstellungen die Optionen **Animationsdauer der Achsen** und **Animationdauer einer Linie** zur Verfügung. Ersteres beschreibt die Zeit, die zum Einblenden der Achsen des Koordinatensystems samt aller Beschriftungen benötigt wird. Das zweite Eingabefeld arbeitet analog zu der Einstellung **Animationsdauer eines Elements**.
Mit den letzten drei Einträgen können verschiedene Wartezeiten konfiguriert werden. Hier ist allerdings erneut zwischen Liniendiagrammen und den übrigen Diagrammtypen zu unterscheiden. So lassen sich im Falle der Liniendiagramme die Wartezeiten nach einer Linienanimation, nach der Animation der Zeichenfläche und nach dem gesamten Diagramm verändern (s. Abb. 3.10). Während ersteres angibt, wie viel Zeit zwischen den Animationen zweier unterschiedlicher Linien verstreicht, definiert die zweite Einstellung, wie lange bis zum Zeichnen der ersten Linie des Diagramms gewartet wird. Über das dritte Eingabefeld lässt sich der Zeitraum nach der Animation des Diagramms bis zum Bewegen der Kamera regeln. Genauer gesagt also der Zeitabschnitt, welcher zum Betrachten des gesamten Diagramms zur Verfügung steht. Ein solche Option ist auch für alle anderen Diagrammtypen vorhanden. Dort sollte allerdings beachtet werden, dass diese Wartezeit erst nach dem Erzeugen sämtlicher Teildiagramme eintrifft. Die Einstellungen **Wartezeit nach Elementanimation** und **Wartezeit nach Teildiagramm** sind nur für Kreis-, Ring und Balkendiagramme verfügbar. Erstere definiert den Zeitraum zwischen den Animationen zweier Kreis-/Ringsegmente oder Balken. Mit **Wartezeit nach Teildiagramm** lässt sich hingegen angeben, wie lange nach der Animation des letzten Elements eines Teildiagramms bis zur Animation des ersten Elements des darauffolgenden gewartet wird.

3.2.4. Schritt 4 - Das Schlussdiagramm

Mit dem Schlussdiagramm wird die Möglichkeit gegeben, Erkenntnisse oder Gesamtergebnisse verschiedenster Art zu präsentieren. Hierbei handelt es sich um ein zweidimensionales Diagramm, das ähnlich dem Titel am Ende der Präsentation eingeblendet werden kann. Die Verwendung dieses vierten Teilformulars ist allerdings optional. So lässt sich durch einfaches Betätigen der Schaltfläche **Weiter** das Erzeugen eines Schlussdiagramms verhindern. Soll dieses aber verwendet werden, so muss man entweder einen zusätzlichen Datensatz zur Verfügung stellen oder eine weitere Bilddatei hochladen. Ein solcher Datensatz ist an das Ende der im nächsten Schritt hochzuladenen CSV-Datei anzufügen. Weitere Informationen zu deren Aufbau können Abschnitt 3.4 entnommen werden.
Um ein Schlussdiagramm zu verwenden, muss die Option **Schlussdiagramm aktivieren** angewählt werden. Daraufhin erscheint eine Auswahlbox, welche den **Typ** des Schlussdiagramms erwartet. Diese bietet die Möglichkeiten ***Generieren*** und ***Bilddatei hochladen*** an. Erstere benötigt einen zusätzlichen Datensatz und blendet beim Anwählen weitere Optionen ein. Hier wird zunächst der Diagrammtyp festgelegt. Wie auch bei der gleichnamigen

Einstellung aus Abschnitt 3.2.2, lässt sich zwischen den Typen Kreis-, Ring-, Balken- oder Liniendiagramm wählen. Diese werden allerdings, im Gegensatz zu allen anderen Diagrammen, zweidimensional dargestellt. Die nächste Option sollte ebenfalls bekannt sein, da sich mit ihr die numerischen Werte einblenden lassen. Auch hier handelt es sich um die tatsächlich im Datensatz vorhandenen Werte ohne jegliche Umrechnung. Die letzte Einstellung betrifft den **Titel des Schlussdiagramms**, welcher am oberen Rand des Präsentationsvideos zentriert dargestellt wird. Dessen Angabe ist allerdings optional. Für die zweite Möglichkeit mit der Bezeichnung ***Bilddatei hochladen*** werden keine weiteren Einstellungen benötigt. Hier erscheint lediglich der Hinweis, dass die entsprechende Grafik zu einem späteren Zeitpunkt hochgeladen wird. Bei dieser handelt es sich um eine gewöhnliche Bilddatei mit einem bestimmten Seitenverhältnis. Zum Vermeiden von Verzerrungen sollte es entsprechend eingehalten werden. Genauere Informationen hierüber liefert Abschnitt 3.4. Abbildung 3.11 zeigt ein generiertes Schlussdiagramm und dessen Einstellungen.

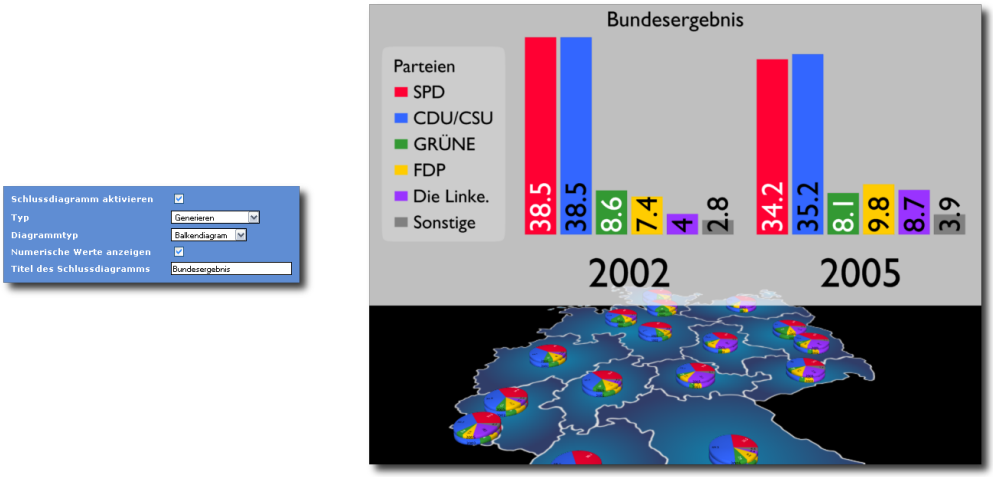

Abbildung 3.11.: Generiertes Schlussdiagramm

Auch dieses Teilformular bietet ein Reihe erweiterter Einstellungen, welche sich mit der entsprechenden Schaltfläche öffnen lassen. Einige der hier angebotenen Optionen sind allerdings abhängig vom Typ des Schlussdiagramms und werden erst später vorgestellt. Das Animationsverhalten lässt sich allerdings in beiden Fällen konfigurieren und stellt die folgenden Eingabefelder zur Verfügung: ***Flugzeit zum Schlussdiagramm***, ***Animationsdauer des Schlussdiagramms*** und ***Anzeigedauer des Schlussdiagramms***.
Ersteres beschreibt den Zeitraum, welcher benötigt wird, um von der Gesamtübersicht mit allen Diagrammen zu der Position zu wechseln, in der das Schlussdiagramm eingeblendet wird. Auch hierfür stehen die im vorherigen Abschnitt vorgestellten Eingabefelder bereit. Mit diesen lässt sich die Anzahl der Bilder pro Sekunde für die jeweilige Animation präzise einstellen. Die zweite Option gibt an, wie lange das Einblenden des Schlussdiagramms dauern soll. Generell geschieht dies analog zur Einstellung ***Einblenden*** des Animationstyps aus

Schritt 3, wobei durch ein langsames Verringern der Transparenz alle vorhandenen Objekte dargestellt werden. Eine Ausnahme sind hierbei aber die Balkendiagramme, deren Animation entsprechend des Typs ***Wachsend*** verläuft. Dies hat keine spezielle Funktion, wirkt allerdings optisch ansprechender. Nachdem nun das Schlussdiagramm vollständig eingeblendet wurde, lässt sich mit Hilfe der dritten und letzten Option festlegen, wie groß der Zeitraum bis zum Ende des Präsentationsvideos sein soll. Genauer gesagt wird hiermit also angegeben, wie lange das Schlussdiagramm zu sehen ist.
Soll ein Schlussdiagramm *generiert* werden, so steht eine weitere Einstellungsmöglichkeit zur Verfügung. Mit Hilfe dieser lassen sich die verwendeten Farben bestimmen. Analog zur gleichnamigen Option in Schritt 2 stehen auch hier die Einträge ***Zufällig***, ***Berechnen***, ***Vorgeben*** und ***Auswählen*** bereit. Diese werden auf dieselbe Arte und Weise verwendet, wie bereits in Abschnitt 3.2.2 erklärt. Des Weiteren steht ein fünfter Eintrag mit der Bezeichnung ***Analog*** zur Auswahl. Wird er selektiert und die Änderungen durch Betätigen der Schaltfläche **Weiter** bestätigt, so übernimmt das Formular die in Schritt 2 gewählten Farbeinstellungen auch für das Schlussdiagramm. Auf diese Art und Weise ist es nicht nötig, die zuvor angegebenen Farben erneut einzustellen.
Abschließend soll das Verwenden einer hochgeladenen Grafik als Schlussdiagramm illustriert werden. Abbildung 3.12 zeigt deshalb sowohl die Grafik selbst als auch das daraus entstandene Schlussdiagramm. Dies verdeutlicht zudem die benötigten Seitenverhältnisse. Entgegen der hier gezeigten Variante ist es aber auch möglich, jede beliebige Grafik am Ende des Präsentationsvideos einzublenden, was bedeutet, dass nicht zwingend ein statistisches Endergebnis präsentiert werden muss.

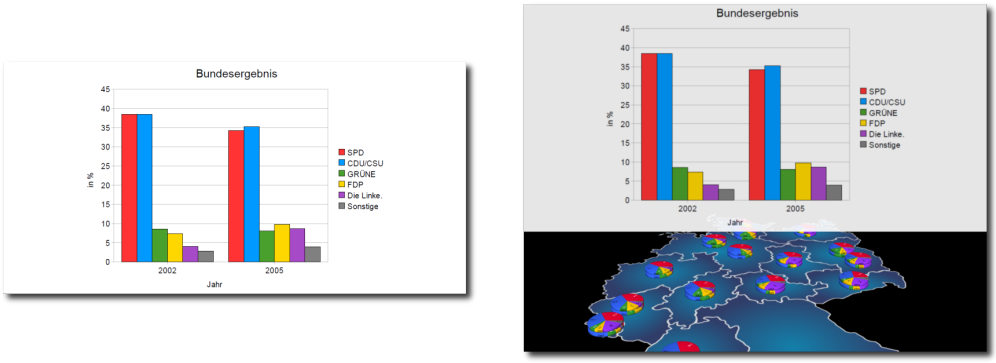

Abbildung 3.12.: Schlussdiagramm mit hochgeladener Grafik

3.2.5. Schritt 5 - Hochladen von Daten und Grafiken

Das fünfte Teilformular beschäftigt sich mit dem Hochladen aller benötigten Dateien. Hiermit sind sowohl die Daten zum Erzeugen der Diagramme als auch sämtliche Bilddateien gemeint. Für jede einzelne Datei wird im oberen Bereich des Formulars ein entsprechendes

Eingabefeld bereitgestellt. Diese bestehen aus einem normalen Textfeld und einer Schaltfläche mit der Bezeichnung ***Durchsuchen*** rechts davon. Dort lässt sich nun entweder der Pfad zur jeweiligen Datei direkt eingeben oder mit Hilfe der Schaltfläche ein Standarddialog zum Auswählen von Dateien öffnen. Wurden auf diese Weise die Diagrammdaten und alle benötigen Grafiken angegeben, so lässt sich mit der Schaltfläche **Dateien hochladen** der Uploadprozess starten. Hierbei werden sämtliche Dateien zum einen auf deren Dateiformat und zum anderen auf die Größe überprüft. Während für Grafiken Formate wie `PNG`, `GIF` oder `JPG` in verschiedenen Ausprägungen erlaubt sind, können für die Diagrammdaten jegliche Art von Textdateien wie `TXT` oder `CSV` verwendet werden. Das Limit für die Größe der Grafiken beträgt 2 bzw. im Falle der Karte 5 Megabyte und die Diagrammdaten dürfen 128 KB nicht überschreiten.
Wurden alle diese Vorgaben eingehalten, so bestätigt die Webseite einen erfolgreichen Upload auf zwei Arten. Als Erstes erscheint die Meldung **Upload erfolgreich** direkt unter dem jeweiligen Eingabefeld. Zudem werden hier Informationen über die Datei und deren Eigenschaften angegeben. Auch im Falle eines fehlgeschlagenen Uploads werden an dieser Stelle Informationen angezeigt, welche zur Fehlerbehebung beitragen sollen. Ein Beispiel für einen solchen Fehler wird in Abbildung 3.13 dargestellt. Dort wurde im ersten Eingabefeld, entgegen der verlangten Bilddatei, eine XML-Datei hochgeladen. Da diese aber nicht den Vorgaben entspricht, lehnt das Formular den Upload ab und weist nachfolgend darauf hin. Solche Informationstexte sind sofort nach dem Hochladen einer Datei vorhanden und verschwinden sobald das Teilformular auf irgendeine Art und Weise verlassen wird. Um aber stets zu wissen, welche Dateien bereits erfolgreich hochgeladen wurden, befindet sich hierfür im unteren Bereich der Seite eine Liste aller Dateien. Dieser kann der aktuelle Status jedes einzelnen Uploads über die Angaben nicht hochgeladen, fehlerhaft und hochgeladen entnommen werden. Auch dieses Verhalten stellt Abbildung 3.13 dar.

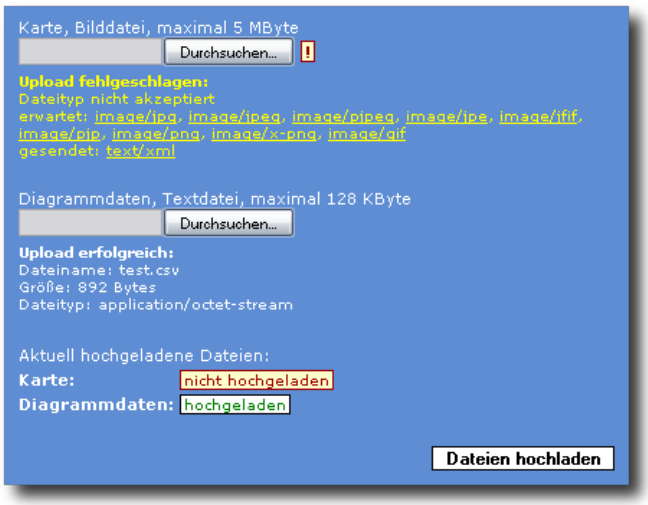

Abbildung 3.13.: Teilfomular für den Datei-Upload

Erst wenn alle Dateien fehlerfrei hochgeladen wurden, lässt sich das Formular über die Schaltfläche **Weiter** erfolgreich bestätigen. Zuvor sind vor allem die Diagrammdaten zu beachten, da diese nicht nur das entsprechende Dateiformat besitzen, sondern zudem einer vorgegebenen Struktur folgen müssen. Wie eine solche CSV-Datei auszusehen hat, beschreibt jedoch Abschnitt 3.4. Treten nun Fehler bei der Struktur der Diagrammdaten auf, so werden diese im unteren Bereich, noch unter der Schaltfläche zum Hochladen der Dateien, mit der jeweiligen Zeilennummer angegeben. Hierbei kann es sich beispielsweise um einen ungültigen Wert oder eine unterschiedliche Anzahl von Spalten handeln. Diese Fehler lassen sich nun Stück für Stück abarbeiten, wobei zur Kontrolle die entsprechende Datei erneut hochgeladen werden muss. Sind alle Fehlermeldungen verschwunden, so lässt sich zum nächsten Teilformular übergehen.

3.2.6. Schritt 6 - Positionierung der Diagramme

Dieses Teilformular beschäftigt sich der Positionierung der einzelnen Diagramme auf der Karte bzw. der zugrundeliegenden Grafik. Hiebei kann auf zwei unterschiedliche Weisen verfahren werden. Zum einen besteht die Möglichkeit Positionen mit Hilfe eines Java-Applets anzugeben und zum anderen kann dies manuell über entsprechende Eingabefelder erfolgen. Da bei der zweiten Variante keine visuelle Ausgabe vorhanden ist, sollte sie lediglich dann verwendet werden, wenn es dem Browser nicht gelingt, das Java-Applet darzustellen. Aus diesem Grund beschäftigt sich der vorliegende Abschnitt hauptsächlich mit der Steuerung des Java-Applets.
Über die Auswahlbox mit der Bezeichnung **Eingabetyp** wird zunächst das Java-Applet mit dem gleichnamigen Eintrag angewählt. Daraufhin erscheint der Link ***Applet starten***, welcher den Browser dazu veranlasst ein neues Fenster zu öffnen und darin das Java-Applet auszuführen. Nach einer kurzen Wartezeit startet die Anwendung und lädt die verwendete Karte, was je nach Verbindung und Kartengröße eine einige Augenblicke dauern kann. Um Fehler zu vermeiden, sollte aber solange gewartet werden, bis das Applet die Grafik tatsächlich anzeigt. Ist dies geschehen, so lassen sich die Diagramme auf der Karte positionieren. Die Oberfläche des geladenen Applets wird am Beispiel einer Deutschlandkarte in Abbildung 3.14 gezeigt. Dort ist rechts die hochgeladene Karte zu sehen, während am linken Rand Informationen und Einstellungsmöglichkeiten bezüglich der Diagrammpositionen zu finden sind.
Durch Betätigen der rechten Maustaste im Bereich der Karte kann nun das erste Diagramm positioniert werden. Zur Visualisierung wird ein Kreis an der aktuellen Mausposition gezeichnet und entsprechend nummeriert. Auf diese Weise lassen sich nun alle übrigen Punkte setzen, wobei die gezeichnete Zahl dem jeweiligen Datensatz in der hochgeladenen CSV-Datei entspricht. So würde beispielsweise an dem Punkt mit der Beschriftung 3 auch der dritte Datensatz erzeugt werden. Deshalb kann es hilfreich sein, die CSV-Datei im Hintergrund zu öffnen.

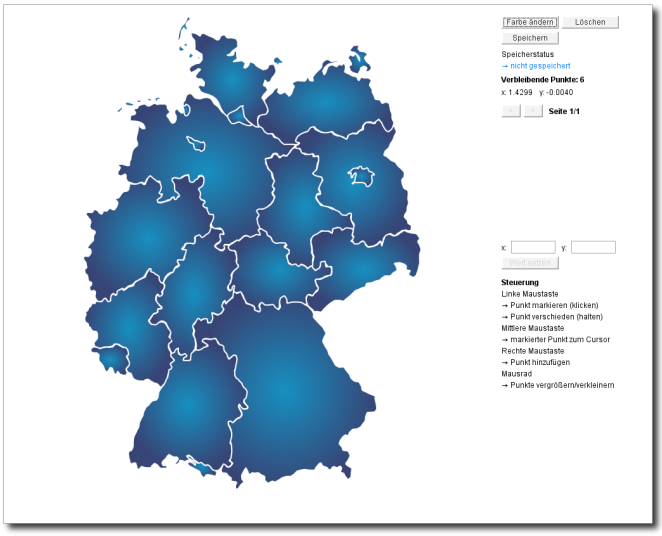

Abbildung 3.14.: Oberfläche des Java-Applets nach dem Start

Allerdings kann es durchaus vorkommen, dass ein gezeichneter Punkt nicht am gewünschten Ort erzeugt wurde oder nachträglich dessen Position verändert werden soll. Aus diesem Grund ist es möglich, einen Punkt im Nachhinein zu verschieben. Durch Betätigen und Halten der linken Maustaste lässt sich die Position des darunterliegenden Punktes beliebig abändern, solange sie sich innerhalb der Karte befindet. Auf diese Weise wurde der verschobene Punkt ebenfalls markiert. Hierbei wird dessen Farbe verändert, sodass er sich von allen anderen Punkten abhebt. Ein solches Markieren kann auch durch einfaches Anklicken erreicht werden. Sind die gezeichneten Punkte zu klein, um sie verschieben zu können, so lässt sich deren Größe mit Hilfe des Mausrads manipulieren. Diese kann sowohl erhöht als auch verringert werden. So sind kleinere Punkte zur genaueren Positionierung zu empfehlen, da sie eine geringere Fläche der Karte verdecken. Deshalb kann es auch während des Bewegens eines Punktes sinnvoll sein, dessen Größe zu regulieren. Eine weitere Möglichkeit die Position eines Punkte zu verändern bietet die mittlere Maustaste. Durch Betätigen dieser wird der momentan markierte Punkt an die aktuelle Position des Mauszeigers verschoben. Hiermit ist es möglich noch feinere Positionierungen durchzuführen. Für alle oben aufgezeigten Mausinteraktionen befindet sich zudem eine kurze Beschreibung im unteren Teil des Informations- und Einstellungsbereichs, welche im Nachfolgenden ein wenig genauer betrachtet werden soll.
Im oberen Bereich befinden sich zunächst drei Schaltflächen mit unterschiedlichen Funktionen. Die wichtigste davon lässt sich zum Speichern der aktuellen Diagrammpositionen verwenden. Durch Betätigen dieser Schaltfläche überprüft das Java-Applet zunächst, ob tatsächlich alle benötigten Punkte gesetzt wurden. Ist dies der Fall, so wird im Informationsbereich direkt darunter der Speicherstatus von ***nicht gespeichert*** in ***erfolgreich gespeichert*** umgeändert. Zudem sollte die Anzeige für die noch verbleibenden Punkte auf Null stehen.

Diese befindet sich ebenfalls im oberen Bereich, direkt unter dem Speicherstatus. Hierüber ist zudem nachvollziehbar, wie viele Punkte noch platziert werden müssen. Sind beim Betätigen der Schaltfläche ***Speichern*** allerdings nicht alle Punkte gesetzt oder andere Fehler aufgetreten, so erscheint eine Fehlermeldung mit kurzer Erklärung an gleicher Stelle. Unterstützend zu der Information über den Speicherstatus, wird in allen genannten Fällen eine detaillierte Meldung des aktuellen Ereignisses mittig über der Karte eingeblendet.
Die Schaltfläche mit der Bezeichnung ***Farbe ändern*** ist genau dann besonders hilfreich, wenn sich die Färbung der Punkte nicht oder nur schlecht von der Karte abhebt. Hiermit lässt sich zwischen fünf unterschiedlichen Farben wechseln, wobei sich auch die Farbe für markierte Punkte ändert. Über die Schaltfläche ***Löschen*** können alle gesetzten Punkte entfernt werden. Das Löschen einzelner Punkte ist allerdings nicht möglich und aufgrund der Funktion zum Verschieben der Punkte auch nicht zwingend notwendig.
Unter der Angabe über die verbleibenden Punkte kann die aktuelle Position des Mauszeigers abgelesen werden. Befindet sich er im Bereich der Karte, so liegen sowohl der x- als auch der y-Wert zwischen 0.0 und 1.0. Da sich sämtliche Punkte in diesem Bereich befinden, stammen auch deren Koordinaten aus dem genannten Intervall. Deren genaue Position kann dem darunterliegenden Informationsbereich entnommen werden. Dieser kann sich bei einer hohen Anzahl von Punkten auch über mehrere Seiten erstrecken. Mit Hilfe der entsprechenden Schaltflächen lässt sich deshalb zwischen den einzelnen Seiten umschalten.
Eine andere Art Diagramme zu positionieren befindet sich direkt unter der Liste aller gesetzten Punkte. Dort lassen sich mit Hilfe von zwei Eingabefeldern Koordinaten auf der Karte direkt über die Tastatur eingeben. Hierbei müssen sich beide Werte selbstverständlich im geschlossenen Intervall zwischen 0.0 und 1.0 befinden. Wurden die Eingabefelder korrekt ausgefüllt, so wird die Schaltfläche ***Wert setzen*** aktiviert. Durch Betätigen dieser lässt sich daraufhin der aktuell markierte Punkt an die definierte Stelle verschieben. Zu beachten ist hierbei allerdings, dass je nach Kartengröße und den angegebenen Werten Rundungsfehler auftreten können.
Sind alle Diagrammpositionen erfolgreich gespeichert, so kann das Java-Applet geschlossen und mit der Schaltfläche **Weiter** zum nächsten Teilformular gewechselt werden. Anderenfalls folgt eine Fehlermeldung. Lässt sich ein solcher Fehler nicht durch erneutes Ausführen des Applets beheben oder das Applet überhaupt gar nicht erst starten, so muss auf die manuelle Angabe der Punkte ohne visuelle Unterstützung zurückgegriffen werden.
Diese wird über das Anwählen des Eintrags ***manuelle Angabe*** aktiviert. Daraufhin erscheinen, neben einer weiteren Auswahlbox, für jeden in der CSV-Datei enthaltenen Datensatz zwei Eingabefelder für die Koordinaten der einzelnen Diagrammpositionen. Über die Auswahlbox lässt sich zunächst einstellen, ob mit absoluten oder relativen Werten gearbeitet werden soll. In beiden Fällen liegt der Ursprung in der oberen linken Ecke der Grafik. Der Unterschied hierbei ist lediglich der zur Verfügung stehende Wertebereich. Bei der ersten Methode können ganzzahlige Werte zwischen Null und der Breite der Karte in Pixeln auf

der x-Achse und Werte zwischen Null und der Höhe der Karte in Pixeln auf der y-Achse angegeben werden. Somit lässt sich, unter Verwendung eines gängigen Grafikprogramms, die jeweilige Diagrammposition bestimmen und in die Textfelder eingeben. Im Normalfall ist dies die einfachere Methode.
Die zweite Möglichkeit erlaubt lediglich relative Werte zwischen 0.0 und 1.0, was sowohl für die x- als auch die y-Achse gilt. Diese lassen sich nun folgendermaßen berechnen. Zunächst sollte der jeweilige Punkt auf der Karte gefunden und dessen Koordinaten identifiziert werden. Hierdurch erhält man, wie bereits bei der ersten Methode, sowohl für den x- als auch für den y-Wert eine Zahl zwischen Null und der Breite bzw. der Höhe der Karte. Diese Werte müssen nun lediglich auf das Intervall von 0.0 bis 1.0 umgerechnet werden, was über eine einfache Division des abgelesenen x-Wertes durch die Breite der Karte erfolgt. Gleiches lässt sich für den y-Wert unter Verwendung der Höhe durchführen. Auf diese Weise wird für alle weiteren Punkte verfahren.
Bei beiden Methoden ist allerdings zu beachten, dass das Betätigen der Schaltfläche **Weiter** jegliche zuvor gesetzten Diagrammpositionen überschreibt. Ist dies nicht erwünscht, so lassen sich durch ein erneutes Wechseln zum Eintrag ***per Java-Applet*** alle bisher angegebenen Diagrammpositionen ohne Änderungen bestätigen.

3.2.7. Schritt 7 - Sonstige Einstellungen

Dieses Teilformular enthält alle verbleibenden Einstellungen, die noch durchgeführt werden müssen, aber nicht einer der anderen Kategorien zuzuordnen sind. So lässt sich dort beispielsweise die Auflösung des Präsentationsvideos einstellen. Es sollte allerdings beachtet werden, dass dessen Qualität abhängig von den hochgeladenen Bilddateien ist. Eine besondere Rolle spielt hierbei die Karte. Besitzt sie eine zu niedrige Auflösung, so kann sie im fertigen Präsentationsvideo verwaschen wirken. Dieser Effekt tritt bei einer zu starken Skalierung auf. Deshalb sollte die Karte eine bestimmte Größe nicht unterschreiten. Welche Abmessungen eine Karte für die jeweilige Videoauflösung besitzen sollte, wird in Abschnitt 3.4 erläutert. Zudem werden im oberen Bereich des Formulars die Auflösungen aller hochgeladenen Bilddateien noch einmal aufgelistet. Aus sämtlichen Werten berechnet das Formular bzw. das PHP-Skript eine empfohlene Auflösung für das Präsentationsvideo. Diese wird sowohl direkt unter den gelisteten Auflösungen der Bilddateien als auch hinter der entsprechenden Auswahlbox angezeigt. Sind die Abmessungen der Grafiken allerdings zu gering, so wird darauf hingewiesen die aktuelle Bilddatei durch eine größere zu ersetzen. Hinter der Auswahlbox erscheint in diesem Fall der Hinweis keine. Abbildung 3.15 zeigt diese Situation im linken Bild, während rechts eine Karte mit ausreichender Größe hochgeladen wurde. Wie zu sehen, empfiehlt sich bei dieser eine Auflösung von 800 x 600.

Über die Auswahlbox mit der Bezeichnung **Videoformat** lässt sich der Dateityp des Präsentationsvideos bestimmen. Zum momentanen Zeitpunkt ist nur die Option ***Mpeg*** verfügbar,

Abbildung 3.15.: Empfohlene Auflösung bei unterschiedlichen Kartengrößen

da hierbei ein guter Kompromiss zwischen Qualität und Dateigröße erzielt werden kann.
Die darauffolgenden Eingabefelder sind speziell für die Angabe der E-Mail-Adresse des Benutzers vorgesehen. Diese wird gleich zweimal verlangt, um die Gefahr eines eventuellen Tippfehlers zu minimieren. Zudem ist hier die Angabe einer korrekten E-Mail-Adresse nötig, was beim Versenden des Teilformulars überprüft wird. Treten dabei Probleme auf, so bricht der Vorgang mit einer Fehlermeldung ab. Daraufhin sollte die E-Mail-Adresse entsprechend korrigiert und erneut die Schaltfläche **Weiter** betätigt werden.
Die letzte Einstellung betrifft das Publizieren des generierten Präsentationsvideos. Dort lässt sich bestätigen, dass ein solches auf der Webseite des ChartFlight Services zu Demonstrationszwecken veröffentlicht werden kann. Dies bedarf allerdings einer weiteren Zustimmung mit Hilfe eines Bestätigungsdialogs.

3.2.8. Schritt 8 - Die Gesamtübersicht

Der letzte Schritt bei der Erstellung eines Auftrags ist die sogenannte Gesamtübersicht. Hier werden alle Teilformulare noch einmal aufgelistet und lassen sich über den entsprechenden Link erneut besuchen. Dies ist besonders vorteilhaft, wenn verschiedene Werte eines bestimmten Teilformulars zu ändern sind. So lässt sich recht unkompliziert mit Hilfe der Gesamtübersicht dorthin wechseln.
Weiterhin wird für jedes einzelne Teilformular ein Statushinweis angegeben. Dieser zeigt die Zeichenfolge abgeschlossen, wenn das jeweilige Formular korrekt ausgefüllt und abgeschickt wurde. Anderenfalls erscheint hier nicht abgeschlossen in roter Schrift. Ist dies der Fall, so muss zum entsprechenden Formular zurückgekehrt, dieses korrigiert und erneut abgeschickt werden. Erst wenn alle Formulare den Status abgeschlossen besitzen, lässt sich der Auftrag erfolgreich absenden.
Ist das Absenden eines Auftrags aus irgendeinem Grund nicht erwünscht, so lässt sich dieser mit Hilfe der Schaltfläche **Auftrag löschen** abbrechen. Hierdurch werden alle Teilformulare auf ihre Standardwerte zurückgesetzt und somit die Eingaben des Benutzers gelöscht.
Abbildung 3.16 zeigt die Gesamtübersicht eines Auftrags, bei welchem die Formulare ***Ani-***

mation und ***Sonstiges*** bisher nur fehlerhaft oder überhaupt nicht ausgefüllt wurden. Somit ist es unmöglich diesen bereits abzusenden.

Abbildung 3.16.: Gesamtübersicht mit nicht bestätigten Formularen

Konnte der Benutzer den Auftrag erfolgreich absenden, so wird dieser in eine Datenbank aufgenommen. Zudem erscheint eine Bestätigungsseite, welche zusätzlich Informationen über den weiteren Verlauf des Auftrags enthält. Hier wird beispielsweise erklärt, welche Schritte als nächstes durchgeführt werden oder was den Benutzer nach dem Erstellen der Vorschaubilder erwartet. Diesen Vorgang erläutert der nachfolgende Abschnitt 3.3.
Ebenso wird auf der Bestätigungsseite die jeweilige Auftragsnummer angegeben, welche aus dem aktuellen Datum, der Uhrzeit beim Starten des Auftrags und einer Zufallszahl besteht. Auch die momentane Position in der Warteschlange lässt sich dort ablesen. Diese errechnet sich lediglich aus jenen Aufträgen, die auf das Erstellen der Vorschaubilder warten. Andere werden hierbei nicht berücksichtigt.

3.3. Weitere Funktionen der Benutzeroberfläche

Der vorliegende Abschnitt beschäftigt sich mit dem Bestätigen, Ablehnen und Löschen eines Auftrags. Diese Funktionen sind erst nach dessen Absenden und dem darauffolgenden Rendern der Vorschaubilder verfügbar. Denn erst dann erhält der Benutzer eine E-Mail mit dem entsprechenden Link zu einer personalisierten Webseite. Hier können zunächst alle Vorschaubilder betrachtet werden, was für das weitere Vorgehen entscheidend ist. Je nach Einstellung erscheint im unteren Bereich der Seite eine Tabelle mit vier bzw. fünf Einträgen zu den gerenderten Bildern. Diese lassen sich durch Anklicken des jeweiligen Links in einem neuen Fenster öffnen. Der fünfte Eintrag ist dabei von der Wahl eines Schlussdiagramms abhängig. Wurde dieses aktiviert, so ist auch dafür eine entsprechende Vorschau verfügbar. Abbildung 3.17 zeigt die Auflistung der Vorschaubilder samt Auftragsnummer.

Im unteren Bereich der Vorschauseite stehen zudem drei Schaltflächen zur Verfügung. Von links nach rechts sind dies **Auftrag löschen**, **Auftrag ändern** und **Auftrag ausführen**. Hierüber lässt sich das weitere Vorgehen steuern.
Mit der ersten Schaltfläche kann, wie der Name schon sagt, der aktuelle Auftrag gelöscht

Abbildung 3.17.: Auflistung der Vorschaubilder

werden. Dies bedarf keiner weiteren Bestätigung und entfernt den entsprechenden Auftrag sofort aus der Datenbank. Die hochgeladenen bzw. erzeugten Benutzerdaten werden kurz darauf ebenfalls gelöscht. Zur Bestätigung erscheint auch hier, ähnlich wie nach dem Absenden eines Auftrags, eine weitere Webseite mit dem entsprechenden Hinweis. Allerdings besteht ein Unterschied zur der gleichnamigen Schaltfläche im unteren Bereich der Gesamtübersicht. Dort werden keine Einträge aus der Datenbank gelöscht, sondern lediglich der aktuelle Auftrag aus den Formularen entfernt. Um also einen bereits in die Datenbank aufgenommenen Auftrag tatsächlich zu löschen, lässt sich ausschließlich die Schaltfläche auf der personalisierten Webseite verwenden.
Waren die Vorschaubilder nicht zufriedenstellend, so besteht ebenso die Möglichkeit, Änderungen am aktuellen Auftrag durchzuführen, wofür sich die Schaltfläche **Auftrag ändern** verwenden lässt. Nach Betätigen dieser werden die gesamten Benutzerdaten geladen und das erste Teilformular geöffnet. Hier lassen sich alle verfügbaren Einstellungen erneut einsehen und gegebenenfalls abändern. Dies funktioniert auf die gleiche Art und Weise, wie es in Abschnitt 3.2 bereits beschrieben wurde. Ein solcher Auftrag reiht sich nach dem Abschicken erneut in die Warteschlange ein, erhält allerdings eine höhere Priorität als ein vollständig neu erzeugter. Ist der geänderte Auftrag wieder an erster Position in der Warteschlange, so wird die veränderte Szene generiert, neue Vorschaubilder gerendert und diese dem Benutzer präsentiert. Sind die gerenderten Bilder immer noch nicht zufriedenstellend, so besteht selbstverständlich die Möglichkeit den Auftrag ein weiteres Mal zu ändern.
Entsprechen die gerenderten Bilder aber den Vorstellungen des Benutzers, so lässt sich der Auftrag über die dritte Schaltfläche nun endgültig für die Erzeugung des Präsentationsvideos freigeben. Auch hiernach erscheint eine Bestätigungsseite, welche erneut über den weiteren Verlauf aufklärt. Zudem werden, wie beim Absenden eines Auftrags, wiederum dessen Nummer und die Position in der Warteschlange angezeigt. Diese ist allerdings unabhängig von der zuvor erwähnten Warteschlange und beinhaltet lediglich Aufträge, welche für das finale Rendern des Präsentationsvideos freigegeben wurden.

3.4. Erzeugung der benötigten Daten

Dieser Abschnitt enthält Informationen über die Erzeugung der für einen Auftrag benötigten Daten. Hierbei werden sowohl die darzustellenden Datensätze der einzelnen Diagramme als auch die Bilddaten für Karte, Titelbildschirm oder Schlussdiagramm betrachtet. Zudem beschriebt der vorliegende Abschnitt, wie sich solche Datensätze erzeugen bzw. aufbereiten lassen und welche Regeln einzuhalten sind, um gute Präsentationsvideos zu erstellen.

3.4.1. Erzeugung eines Datensatzes

In diesem Abschnitt wird exemplarisch dargestellt, wie sich mit Hilfe des Moduls *Calc* der *OpenOffice.org Suite* Daten so aufbereiten lassen, dass sie vom ChartFlight Service erfolgreich erkannt und verarbeitet werden können. Dies betrifft sowohl die Daten der dreidimensionalen Diagramme als auch die des Schlussdiagramms. Hierzu verwenden wir den konstruierten Datensatz aus Tabelle 3.1, welcher so gewählt ist, dass möglichst viele Sonderfälle abgedeckt werden. Wie zu erkennen, ist der Datensatz in einen gelben und einen grünen Bereich aufgeteilt. Während die gelb hinterlegten Teiltabellen die Daten für alle dreidimensionalen Diagramme liefern, enthalten die grün hinterlegten Zeilen Daten für ein Schlussdiagramm.

1	A	B	C	D
2	Diagramm 1	Spalte 1	Spalte 2	
3	Zeile 1	50	80	
4	Zeile 2	50	20	
5				
6	Diagramm 2	Spalte 1	Spalte 2	Spalte 3
7	Zeile 1	21	70	50
8	Zeile 2	66	9	40
9				
1	Diagramm 3	Spalte 1	Spalte 2	
11	Zeile 1	120	80	
12	Zeile 2	20	15	
13	Zeile 3	5	40	
14				
15	Diagramm 4	Spalte 1	Spalte 2	Spalte 3
16	Zeile 1	100	10	1
17	Zeile 2	50	5	0,5
18	Zeile 3	25	2,5	0,25
19				
20	Schlussdiagramm	Spalte 1	Spalte 2	
21	Zeile 1	70	24	
22	Zeile 2	54	56	
23	Zeile 3	26	124	
24	Zeile 4	99	12	

Tabelle 3.1.: Konstruierter Datensatz

Aufbau der Daten

Wie bereits erwähnt, besteht ein Datensatz aus mehreren Teiltabellen, welche jeweils von genau einer, im Beispiel rot hinterlegten Zeile getrennt sind. Diese Zeilen sind notwendig, um die Tabellen voneinander abgrenzen zu können. Sollten sie nicht vorhanden sein, erscheint nach dem Hochladen der CSV-Datei eine Fehlermeldung. Das Upload-Formular erkennt jedoch nicht, dass dort eine Leerzeile fehlt, sondern würde darüber informieren, dass eine Zeile mit nicht-numerischen Werten aufgetaucht ist. Der Grund hierfür ist, dass die beiden Teiltabellen nicht separat, sondern als eine einzige Tabelle interpretiert werden. Tritt ein solches Problem auf, so sollte man deshalb zunächst überprüfen, ob alle Tabellen mit einer Leerzeile voneinander getrennt sind. Im Falle eines Schlussdiagramms wird auch dieses auf die gleiche Art und Weise vom vorherigen Diagramm separiert. Generell ist es auch möglich, mehr als eine Zeile zum Abtrennen der verschiedenen Diagramme untereinander zu verwenden. Allerdings empfiehlt es sich aus Gründen der Übersicht bei dieser Notation zu bleiben.
Betrachtet man nun eine solche Teiltabelle ein wenig genauer, so wird recht schnell deutlich, dass diese sowohl einen Zeilen- als auch einen Spaltenkopf besitzt. Über den jeweiligen Spaltenkopf lässt sich die Beschriftung eines Teildiagramms steuern. Mit Hilfe des Zeilenkopfes kann die Beschriftung des jeweiligen Elements innerhalb eines Diagramms angegeben werden. Die Zeilenköpfe steuern zudem die Farbe eines Elements, sodass gleiche Zeilenköpfe auch gleiche Farben bedeuten. Dies gilt ebenfalls diagrammübergreifend. Definiert man beispielsweise für die Beschriftung *Zeile 1* eine rote Farbe, so wird der jeweilige Wert in jeder Zeile mit dieser Beschriftung auch in Rot dargestellt.
Insgesamt dürfen allerdings nur 8 unterschiedliche Zeilenköpfe in der gesamten Datei auftreten. Eine Ausnahme hierfür ist aber das Schlussdiagramm, welches unabhängig von der oben genannten Regel zu betrachten ist. Dies bedeutet, dass sich 8 weitere Zeilenköpfe verwenden lassen, sofern ein solches Vorgehen benötigt wird. Ein einzelnes Diagramm ist somit also auch auf die maximale Anzahl von 8 Zeilen plus einer Zeile für die Spaltenköpfe beschränkt. Ähnlich wird mit den Spalten verfahren. Hier sind maximal 8 Spalten plus einer Spalte für die Zeilenköpfe möglich. In der linken oberen Ecke jeder Tabelle bleibt somit Platz für eine kurze Beschreibung. Ein Beispiel für eine solche Beschriftung in Tabelle 3.1 wäre also *Diagramm 1*, für einen Spaltenkopf *Spalte 1* und für einen Zeilenkopf *Zeile 1*.
Aus dem oben genannten resultiert nun, dass jeweils eine Spalte einer Teiltabelle die Daten für ein einzelnes Teildiagramm zur Verfügung stellt. Bei mehreren Spalten werden somit auch mehrere Teildiagramme erzeugt. Soll beispielsweise nur ein einziges Kreisdiagramm pro Teiltabelle dargestellt werden, genügt hierfür jeweils eine Spalte. Über die Zeilen wird daraufhin die Farbe und Größe des jeweiligen Sektors angegeben. Eine gesamte Teiltabelle stellt infolgedessen also die Daten für ein vollständiges Diagramm an einer gegebenen Position bereit. Abbildung 3.18 soll dies anhand der ersten Teiltabelle des oben aufgeführten Datensatzes verdeutlichen.

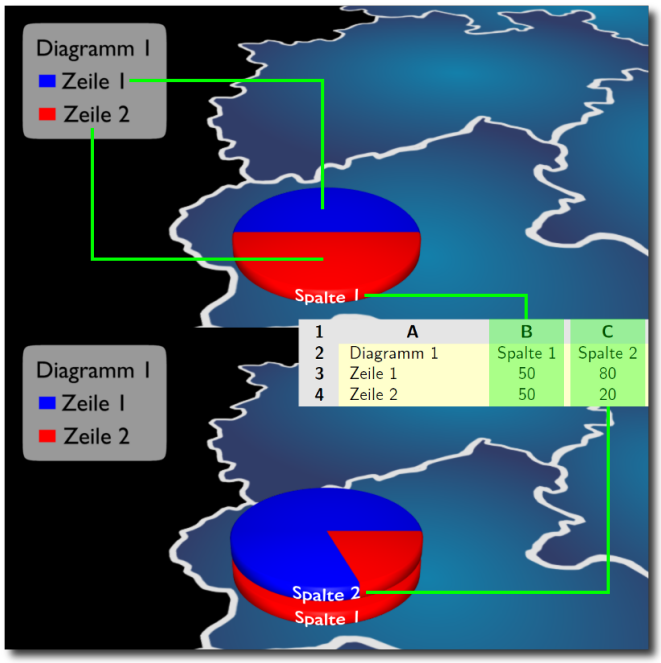

Abbildung 3.18.: Kreisdiagramme zur Darstellung des ersten Datensatzes

Wie an Teildiagramm 4 zu sehen ist, werden natürlich auch Werte mit Nachkommastellen verarbeitet. Diese lassen sich auf zwei unterschiedliche Arten angeben: entweder wird hierfür ein Dezimalkomma oder, wie im Englischen, ein Dezimalpunkt verwendet - beide Schreibweisen sing möglich. Negative Werte lassen sich allerdings nur durch Liniendiagrammen darstellen. Bei allen anderen Diagrammtypen werden diese abgewiesen und eine entsprechende Fehlermeldung beim Hochladen der CSV-Datei angezeigt. Zudem gilt auch jegliche Art von Zeichenketten außerhalb der Zeilen- und Spaltenköpfe als ungültige Eingabe.
Auf die gleiche Art und Weise wird das Schlussdiagramm erzeugt. Auch hier generiert der ChartFlight Service für jede Spalte ein separates Teildiagramm und stellt die Zeilen mit den vorgegebenen Farben dar. Abbildung 3.19 zeigt das Schlussdiagramm, welches auf Basis der Beispieldaten erstellt wurde.

Umwandeln der Daten in eine CSV-Datei

Wurden alle oben genannten Regeln beachtet bzw. die Daten entsprechend dieser aufbereitet, lassen sie sich als CSV-Datei speichern. Die Vorgehensweise ist allerdings von Programm zu Programm verschieden. Im Nachfolgenden wird dies für *OpenOffice.org* ab Version 2.0 detaillierter erläutert.
Das Erzeugen einer CSV-Datei ist im Prinzip recht einfach, allerdings sind dabei einige wichtige Einstellungen vorzunehmen. Zunächst muss im Menüpunkt ***Datei*** der Eintrag ***Speichern unter ...*** gewählt werden, woraufhin der normale Dateiauswahldialog erscheint. Dort befindet

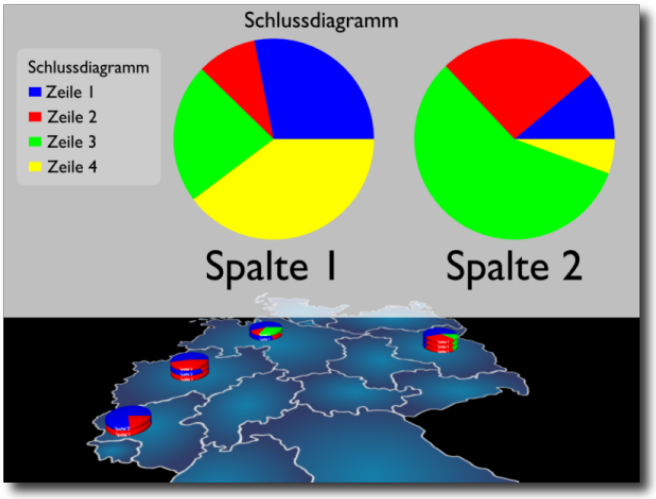

Abbildung 3.19.: Schlussdiagramm des Beispieldatensatzes

sich im unteren Bereich eine Auswahlbox mit dem Namen **Dateityp**, mit welcher sich nun der Eintrag ***Text CSV (.csv)*** wählen lässt. Zudem sollte man sicherstellen, dass entweder die Automatische Dateinamenserweiterung aktiv ist oder man beim Eingeben des Dateinamens die korrekte Dateiendung manuell anfügt. Daraufhin können mit einem Klick auf ***Speichern*** die getätigten Einstellungen bestätigt werden.
Ist dies getan, so erscheint ein weiterer Dialog mit der Nachfrage, ob es tatsächlich beabsichtigt war die Datei im CSV-Format zu speichern. Um den Exportvorgang nicht abzubrechen, sollte der Dialog entsprechend bestätigt werden. Das Erscheinen dieses Fensters ist allerdings abhängig von den jeweiligen Programmeinstellungen.
Im letzten Fenster, mit dem Titel **Textexport**, ist es äußerst wichtig, dass die nachfolgenden Werte exakt einstellt werden. Hierdurch lassen sich viele der möglichen Fehler beim Hochladen der CSV-Datei bereits ausgrenzen und zudem treten keine oder kaum Probleme mit Sonderzeichen auf:

Zeichensatz	Unicode (UTF-8) (wichtig zur korrekten Darstellung von Sonderzeichen)
Feldtrenner	, (Komma)
Texttrenner	"(Doppeltes Anführungszeichen)
Zellinhalt wie angezeigt	aktiv
Feste Spaltenbreite	inaktiv

Tabelle 3.2.: Einstellungen für den CSV-Export

Wurden alle Einstellungen korrekt vorgenommen, so sollte die erzeugte CSV-Datei dem nachfolgenden Listing entsprechen. Dabei ist zu erkennen, dass alle Texteingaben in doppelten Anführungszeichen stehen, während die Zahlenwerte ohne diese auskommen. Als Feldtrenner wurde, wie zuvor eingestellt, das Komma verwendet. Allerdings kommt es in der vierten Teiltabelle zu Konflikten zwischen dem Komma, welches bei nicht ganzzahligen Werten verwendet

wird und dem, das als Feldtrenner dient. Aus diesem Grund werden auch Fließkommazahlen in Anführungszeichen gesetzt. Weiterhin ist zu erkennen, dass eine Leerzeile im Beispiellisting mit mehreren Kommas dargestellt wird. Dies entsteht bei der Umwandlung einer Tabelle ins CSV-Format, ist aber nicht unbedingt für ein korrektes Erkennen der Datei beim Hochladen notwendig. Dort kann deshalb auch eine vollständig leere Zeile auftauchen.
Mit Hilfe dieser Informationen und eines geeigneten Texteditors sollte es somit auch möglich sein, einen solchen Datensatz von Hand zu erzeugen bzw. aufzubereiten. Dabei ist allerdings zu beachten, dass für die Kodierung der Datei der UTF-8 Zeichensatz verwendet wird.

```
1  "Diagramm 1","Spalte 1","Spalte 2",
2  "Zeile 1",50,80,
3  "Zeile 2",50,20,
4  ,,,
5  "Diagramm 2","Spalte 1","Spalte 2","Spalte 3"
6  "Zeile 1",21,70,50
7  "Zeile 2",66,9,40
8  ,,,
9  "Diagramm 3","Spalte 1","Spalte 2",
10 "Zeile 1",120,80,
11 "Zeile 2",20,15,
12 "Zeile 3",5,40,
13 ,,,
14 "Diagramm 4","Spalte 1","Spalte 2","Spalte 3"
15 "Zeile 1",100,10,1
16 "Zeile 2",50,5,"0,5"
17 "Zeile 3",25,"2,5","0,25"
18 ,,,
19 "Schlussdiagramm","Spalte 1","Spalte 2",
20 "Zeile 1",70,24,
21 "Zeile 2",54,56,
22 "Zeile 3",26,124,
23 "Zeile 4",99,12,
```

Um aber auch ein gutes Präsentationsvideo zu erhalten, gilt es zudem, die nachfolgenden Richtlinien zu beachten. Diese sollen für eine bessere Lesbarkeit aller dargestellten Texte oder Beschreibungen sorgen.

1. Beschriftungen für Spalten- und Zeilenköpfe sollten so kurz wie möglich sein. Denn je länger diese sind, desto größer ist die Wahrscheinlichkeit, dass sie skaliert werden müssen und die Schriftgröße sich somit verringert.

2. Gleiches gilt für den Diagrammtitel sowie für alle anderen Texteingaben.

3. Ebenso sollten die numerischen Werte nach Möglichkeit nicht zu viele Stellen aufweisen und falls doch, entsprechend reduziert werden. Dies ist hauptsächlich dann wichtig, wenn die Einstellung ***Numerische Werte anzeigen*** in den jeweiligen Teilformularen aktiviert ist.

3.4.2. Auswahl einer Karte

Dieser etwas kürzere Abschnitt zeigt, wie die richtige Karte, oder besser gesagt eine Karte, in der richtigen Auflösung für die Präsentation gefunden werden kann. Mit Karte muss nicht zwingend eine Landkarte gemeint sein, es lässt sich jede beliebige Grafik als Basis des Präsentationsvideos verwenden. Hierbei werden die gängigsten Bildformate wie JPEG, PNG oder GIF unterstützt. Die Qualität des Präsentationsvideos hängt allerdings maßgeblich von der Qualität der Karte ab. Denn wenn diese eine zu niedrige Auflösung bzw. Größe aufweist, können die Zeichen oder Illustrationen auf ihr recht schnell verschwommen wirken.
Um die optimale Größe für eine Karte herauszufinden, sollte zunächst bekannt sein, welche der vier verfügbaren Videoauflösungen verwendet wird. Denn abhängig davon ist auch die benötigte Mindestgröße einer Karte, um oben genannte Probleme zu vermeiden. Die nachfolgende Auflistung (3.3) gibt Aufschluss über das Verhältnis zwischen Videoauflösung und minimaler Kartengröße. Zu beachten ist allerdings, dass unter dem Punkt *minimale Kartengröße* immer die Länge der kürzesten Seite einer Karte betrachtet wird. Ist dort beispielsweise ein Wert von 1000 aufgelistet, so heißt dies, dass die Karte an ihrer kürzesten Seite 1000 Pixel nicht unterschreiten sollte. Unterstützend hierfür lässt sich auch die in Abschnitt 3.2.7 gezeigte Funktion zur Bestimmung der empfohlenen Videoauflösung verwenden. Dort wird ebenfalls mit der unten aufgeführten Tabelle gearbeitet, wobei in den beiden letzten Zeilen jeweils die gemittelte minimale Kartengröße in diesen Vorgang mit einfließt.

Videoauflösung (in Pixel)	minimale Kartengröße (in Pixel)
512 x 384	800
640 x 480	1000
800 x 600	1500 bis 2000
1024 x 768	2000 bis 2500

Tabelle 3.3.: Einstellungen für den CSV-Export

Nun gibt es mehrere Möglichkeiten, an eine Karte in der entsprechenden Größe zu gelangen. So lässt sich beispielsweise im Internet danach suchen, wobei es die jeweiligen Urheberrechte zu beachten gilt. Hier gibt es allerdings auch eine ganze Reihe freier Karten, die für beliebige Projekte verwendet und teilweise sogar abgeändert werden dürfen. Ebenso kann die benötigte Grafik bereits in einem Vektorformat wie SVG vorliegen. Dies hätte den Vorteil, dass eine solche Karte in jeder beliebigen Auflösung exportiert werden können. Eine dritte Möglichkeit wäre das Einscannen einer analogen Karte. Auch hierbei lässt sich die benötigte Auflösung entsprechend voreinstellen. Die oben genannten Methoden sind die drei gängigsten, wobei

sicherlich noch andere existieren.
Abschließend ist noch zu sagen, dass natürlich auch Karten von kleinerem Format als den oben vorgeschlagenen verwendet werden können. Dies funktioniert dann allerdings nur auf Kosten einer verminderten Qualität. Von Karten mit Seitenverhältnissen kleiner als 500 x 500 Pixeln ist im Allgemeinen aber abzuraten. Ähnliches gilt auch für zu große Karten. So ist es ebenfalls nicht zu empfehlen, Karten mit zu großen Seitenverhältnissen, wie beispielsweise 20000 x 20000 Pixeln zu verwenden. Dies erhöht zum einen die Dauer für das Hochladen der Karte, zum anderen kann aber auch die Zeit zur Generierung des Präsentationsvideos hierdurch verlängert werden.

3.4.3. Bilddateien für Titel und Schlussdiagramm

Da neben der Karte auch weitere Grafiken verwendet werden können, enthält dieser Abschnitt Informationen darüber, in welcher Größe solche Bilddateien vorliegen sollten. Weiterhin spielt auch das Seitenverhältnis eine wichtige Rolle. Hierbei werden sowohl Grafiken für den Titelbildschirm als auch solche für das Schlussdiagramm betrachtet. Während der Titelbildschirm die gesamte Bildfläche benötigt, was den Seitenverhältnissen des zugrundeliegenden Videos entspricht, hat das Schlussdiagramm eine reduzierte Höhe bei gleichbleibender Breite. Dies verdeutlichen die Abbildungen 3.3 und 3.12 aus Abschnitt 3.2.
Die genauen Verhältnisse lassen sich in der nachfolgenden Tabelle ablesen. Richtet man sich nach diesen Vorgaben, so werden weder Titelbildschirm noch Schlussdiagramm irgendeiner Art von Verzerrung unterliegen. Verwendet man für die jeweilige Grafik auch die entsprechende Größe, unter Beibehaltung des Seitenverhältnisses, so lassen sich zudem Qualitätsverluste vermeiden. Diese Größenangaben sind ebenfalls in Tabelle 3.4 aufgeführt und richten sich nach den Einstellungen für die Auflösung des Präsentationsvideos.

Bildatei	Seitenverhältnis	Beispielgrößen
Titelbildschirm	4:3	512 x 384
		640 x 480
		800 x 600
		1024 x 768
Schlussdiagramm	2:1	512 x 256
		640 x 320
		800 x 400
		1024 x 512

Tabelle 3.4.: Seitenverhältnisse der Grafiken für Titel und Schlussdiagramm

3.5. Das Ergebnis

Wurden nun alle Einstellungen getätigt, die benötigten Daten hochgeladen und die Vorschaubilder betrachtet, so lässt sich der Auftrag für den endgültigen Rendervorgang freige-

ben. Detaillierte Informationen hierzu enthält Abschnitt 3.3. Daraufhin wird dieser erneut in die Warteschlange eingereiht. Sobald dort die erste Position erreicht ist, startet der Rendervorgang und kann mehrere Stunden in Anspruch nehmen. Nach der Fertigstellung meldet sich der Service ein zweites Mal per E-Mail, welche weitere Hinweise und den Link zu der abschließenden Webseite enthält. Dort kann der Benutzer das erzeugte Präsentationsvideo nun herunterladen, das in der Regel zwischen 20 und 100 Megabyte groß ist. Dies sind allerdings lediglich Richtwerte, weshalb die tatsächliche Dateigröße, je nach Länge des Videos und Größe aller hochgeladenen Bilddateien, hiervon abweichen kann.
Nach dem Erhalt der E-Mail hat der Benutzer rund einen Monat Zeit, um das Präsentationsvideo herunterzuladen. Danach werden alle Benutzerdaten vom Server gelöscht. Dieser Zeitraum lässt sich aber beliebig kürzen oder verlängern, was im nächsten Kapitel über die Architektur des ChartFlight Services detaillierter betrachtet wird. Auch in diesem Fall erhält der Benutzer, kurz bevor der oben genannte Zeitraum abläuft, eine E-Mail mit der entsprechenden Warnung und der Angabe, wie lange die jeweiligen Daten noch auf dem Server verbleiben.
Die nachfolgende Abbildung (3.20) zeigt noch einmal in einer kurzen Sequenz das Ergebnis des gerenderten Präsentationsvideos. Hierbei wurden die Bilder bewusst klein gehalten, da ein Großteil dieser bereits in Abschnitt 3.2 zu betrachten ist.

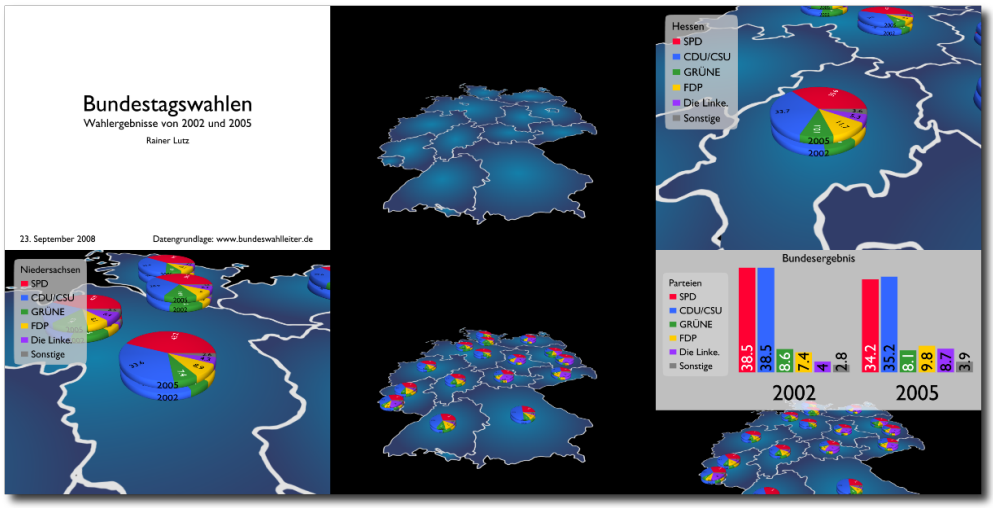

Abbildung 3.20.: Beispielsequenz des Präsentationsvideos

3.6. Ein weiteres Beispiel

Zum Abschluss des vorliegenden Kapitels betrachten wir nun ein weiteres Beispiel. Hierbei zeigen sich verschiedene Aspekte und Funktionen des ChartFlight Services, welche bisher lediglich erläutert, allerdings nicht dargestellt wurden. Der hier vorgestellte Auftrag soll unter

anderem zeigen, dass sich auch unterschiedliche Sachverhalte darstellen lassen und die Visualisierung nicht zwingend auf einer Karte stattfinden muss. So wird nachfolgend ein solches Präsentationsvideo betrachtet. Dieses verwendet als zugrundeliegende Grafik ein Fußballfeld und visualisiert passend dazu die Ballkontakte der deutschen Nationalmannschaft im Spiel gegen Holland bei der WM 1990. Hierzu wird jeder Spieler einzeln angeflogen und dessen Statistiken für die erste Halbzeit dargestellt. Das entsprechende Diagramm ist in mehrere Zeitintervalle aufgeteilt, sodass die Ballkontakte in den ersten 15 Minuten, dem mittleren Teil der Halbzeit und den letzten 15 Minuten sich einzeln ablesen lassen. Das Schlussdiagramm visualisiert diese Zeiträume jeweils für den gesamten Mannschaftsteil.

Abbildung 3.21.: Bildsequenz des Fußballbeispiels

Der zugrundeliegende Datensatz wurde von Micheal Burch zur Verfügung gestellt und die benötigten Daten zunächst herausgefiltert. Im Gegensatz dazu ist die Grafik des Fußballfeldes unter dem nachfolgenden Link frei verfügbar und ließ sich entsprechend anpassen. Hierbei wurden lediglich, unter Verwendung eines Programms zur Manipulation von Rastergrafiken, die unterschiedlichen Beschreibungstexte entfernt.

`http://commons.wikimedia.org/wiki/Image:Fußballfeld_mit_Maßen_und_Beschreibung.png`

Kapitel 4.

4 Architektur

Nachdem nun der ChartFlight Service aus Sicht eines Benutzers betrachtet wurde, gibt das vorliegende Kapitel Aufschluss über die Architektur der einzelnen Komponenten. Im ersten Abschnitt werden diese zunächst kurz vorgestellt und dessen Funktionen erläutert. Hierbei soll lediglich deren Aufgabe und Stellenwert innerhalb des Services sowie deren Zusammenarbeit und Kommunikation verdeutlicht werden. Aufschluss über die entsprechenden Implementierungsdetails geben jedoch die separaten Abschnitte 4.3, 4.4 und 4.5. Hier lässt sich anhand verschiedener Auszüge des Quellcodes ein tieferer Einblick in die Implementierung der jeweiligen Komponente vornehmen. Abschließend werden Probleme behandelt, die während der Entwicklung auftraten und die Installation des Services beschrieben.

4.1. Vorstellung der Komponenten

Dieser Abschnitt stellt die einzelnen Komponenten des ChartFlight Services ein wenig genauer vor, beschreibt deren Funktionen und die Art und Weise, wie diese untereinander kommunizieren bzw. zusammenarbeiten. Weiterhin wird hier gezeigt, was bei der Bearbeitung eines Auftrags im Hintergrund abläuft. Dies geschieht in zwei Schritten: Zunächst soll anhand eines vereinfachten Ablaufdiagramms die Grundfunktionalität des ChartFlight Services erläutert und daraufhin detaillierter auf die einzelnen Komponenten und komplexere Aufgaben eingegangen werden. Da die Datenbank ein wichtiger Bestandteil des Services ist, stellt ein eigener Unterabschnitt deren Funktion vor. Hier werden die darin enthaltenen Tabellen und die Informationen, welche zum Hinzufügen eines neuen Auftrags nötig sind, detailliert beschrieben. Auf diese Weise lassen sich alle wichtigen Voraussetzungen für die nachfolgenden Kapitel zu schaffen.

4.1.1. Ablauf eines Auftrags

Der ChartFlight Service besteht aus drei Komponenten, welchen unterschiedliche Aufgaben zugeteilt sind. Hierbei muss zunächst zwischen dem sogenannten *Frontend* und dem *Backend* unterschieden werden. Ersteres bezeichnet den Teil des Services, welcher für den Benutzer sichtbar ist und über diesen er seine Daten hochladen und Einstellungen tätigen kann. Die Möglichkeiten des Frontends wurden bereits in Kapitel 3 vorgestellt und somit bedarf es an dieser Stelle keiner detaillierteren Informationen. Kurz gesagt handelt es sich hierbei um den Teil des Services, mit dem der Benutzer in Kontakt tritt.
Während das Frontend also lediglich Funktionen zur Erfassung der Benutzerdaten besitzt, verrichtet das Backend den Hauptteil der Arbeit. Erst dort kommen Blender und die speziell dafür entworfenen Python-Skripte ins Spiel. Hierbei wird mit Hilfe dieser Skripte und auf Basis der Benutzerdaten eine dreidimensionale Szene generiert, entsprechend des aktuellen Status gerendert und danach der jeweilige Benutzer benachrichtigt.
Da aus leistungstechnischen Gründen sich nicht für jeden einzelnen Benutzer auch ein eigene Instanz von Blender starten lässt, existiert zudem eine Warteschlange, in welche die abgeschickten Aufträge eingereiht werden. Die Abarbeitung übernimmt ein Java-Programm, das speziell für diesen Zweck implementiert wurde. Zudem ist der sogenannte JobListener für weitere Aufgaben, wie beispielsweise für das Löschen von abgebrochenen oder abgeschlossenen Aufträgen sowie für das Benachrichtigen des Benutzers in letzterem Fall verantwortlich. Hierzu lässt sich das Programm im Hintergrund starten und überprüft darauf in vorgegebenen Abständen, ob zu bearbeitende Aufträge in der Datenbank vorliegen oder ob diese eventuell bereinigt werden muss.
Nachfolgend soll anhand eines vereinfachten Ablaufdiagramms das Zusammenarbeiten der gerade vorgestellten Komponenten und deren Grundfunktionen ein wenig genauer erläutert werden, bevor wir danach ein detaillierteren Ablauf betrachten. Abbildung 4.1 zeigt zunächst das vereinfachte Ablaufdiagramm, welches die Grundkomponenten des ChartFlight Services beinhaltet.

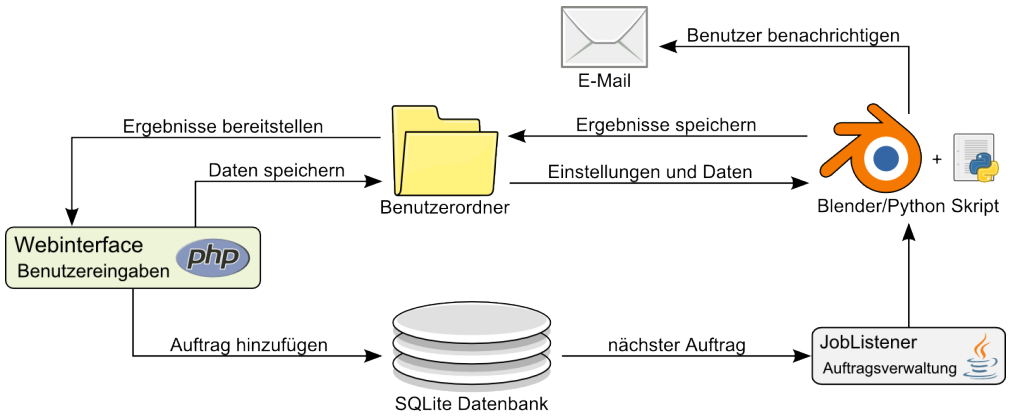

Abbildung 4.1.: vereinfachtes Ablaufdiagramm

Bei genauerer Betrachtung dieses Ablaufdiagramms lassen sich nun recht einfach die oben beschriebenen Komponenten wiederfinden. Zudem kann man dort drei weitere wichtige Bestandteile des Services erkennen: den Benutzerordner, die Datenbank und das Versenden von E-Mails. Zusammengefasst sind dies die sechs Grundkomponenten, wobei Blender und die diversen Python-Skripte als eine einzige gelten. Durch Kommunikation der einzelnen Komponenten untereinander lässt sich nun ein Auftrag abarbeiten. Zu beachten ist allerdings, dass im vereinfachten Diagramm von einem fehlerfreien Ablauf dieses Auftrags ausgegangen wird. Hierbei handelt es sich lediglich um den Prozess eines einzigen Auftrags und nicht um die Bearbeitung mehrerer solcher Aufträge oder allgemeiner Aufgaben.
Im ersten Schritt werden die vom Benutzer getätigten Eingaben über das Webinterface entgegengenommen. Hierbei handelt es sich sowohl um solche Einstellungen, die mit Hilfe der Formulare gemacht wurden, als auch um alle hochgeladenen Daten. Diese speichert das Frontend daraufhin im speziell dafür angelegten Benutzerordner zur weiteren Verarbeitung. Zudem wird der abgesendete Auftrag der SQLite-Datenbank hinzugefügt und kann nun, bei der nächsten Überprüfung durch den JobListener, gefunden werden. Hierzu muss dieser allerdings an erster Stelle in der Warteschlange stehen, was heißt, dass alle zuvor eingefügten Aufträge bereits bearbeitet wurden. Ist dies der Fall, so startet der JobListener Blender, die benötigten Skripte und reicht somit den Auftrag an die 3D-Modellierungssoftware weiter. Hier erzeugen die Python-Skripte zunächst die Szene auf Basis der zuvor gespeicherten Daten und rendern daraufhin die entsprechenden Ergebnisse. Auch diese werden im Benutzerordner gesichert und anschließend über das Webinterface zur Verfügung gestellt. Weiterhin erhält der Benutzer per E-Mail eine Benachrichtigung über die Bereitstellung der Ergebnisse seines Auftrags. Sowohl das Rendern der diversen Vorschaubilder als auch das des finalen Präsentationsvideos folgt prinzipiell dem vorgestellten Ablauf, wobei sich diese lediglich in der Zahl der zu rendernden Bilder unterscheiden.
Grundsätzlich wurden mit Hilfe dieses Ablaufdiagramms bereits alle wichtigen Komponenten für die Bearbeitung eines Auftrags vorgestellt. Allerdings lassen sich hier noch weitere Aufgaben durchführen, damit der ChartFlight Service möglichst fehlerfrei ablaufen kann. Selbst im Falle eines Fehlers müssen die entsprechenden Sicherheitsvorkehrungen getroffen werden, um beispielsweise einen vollständigen Ausfall des Services zu vermeiden oder den betroffenen Benutzer gegebenenfalls zu benachrichtigen. Ein weiterer wichtiger Punkt wäre die Behandlung abgebrochener oder abgeschlossener Aufträge und der jeweiligen Benutzerdaten. Aus diesem Grund lässt sich die Arbeitsweise ebenso anhand eines komplexeren Ablaufdiagramms erläutern, welches das Zusammenarbeiten der einzelnen Komponenten wesentlich detaillierter darstellt. Zudem wurden mögliche Fehler in das Diagramm aufgenommen und visualisiert, wie diese im Ernstfall zu behandeln sind. Analog zum vereinfachten Ablaufdiagramm werden auch hier die drei Hauptkomponenten - Frontend, JobListener und Backend - dargestellt, wobei die komplexere Variante deren Aufgabenliste nun vollständig angibt. Abbildung 4.2 zeigt dies und alle weiteren Vorgänge des komplexen Ablaufdiagramms im Querformat.

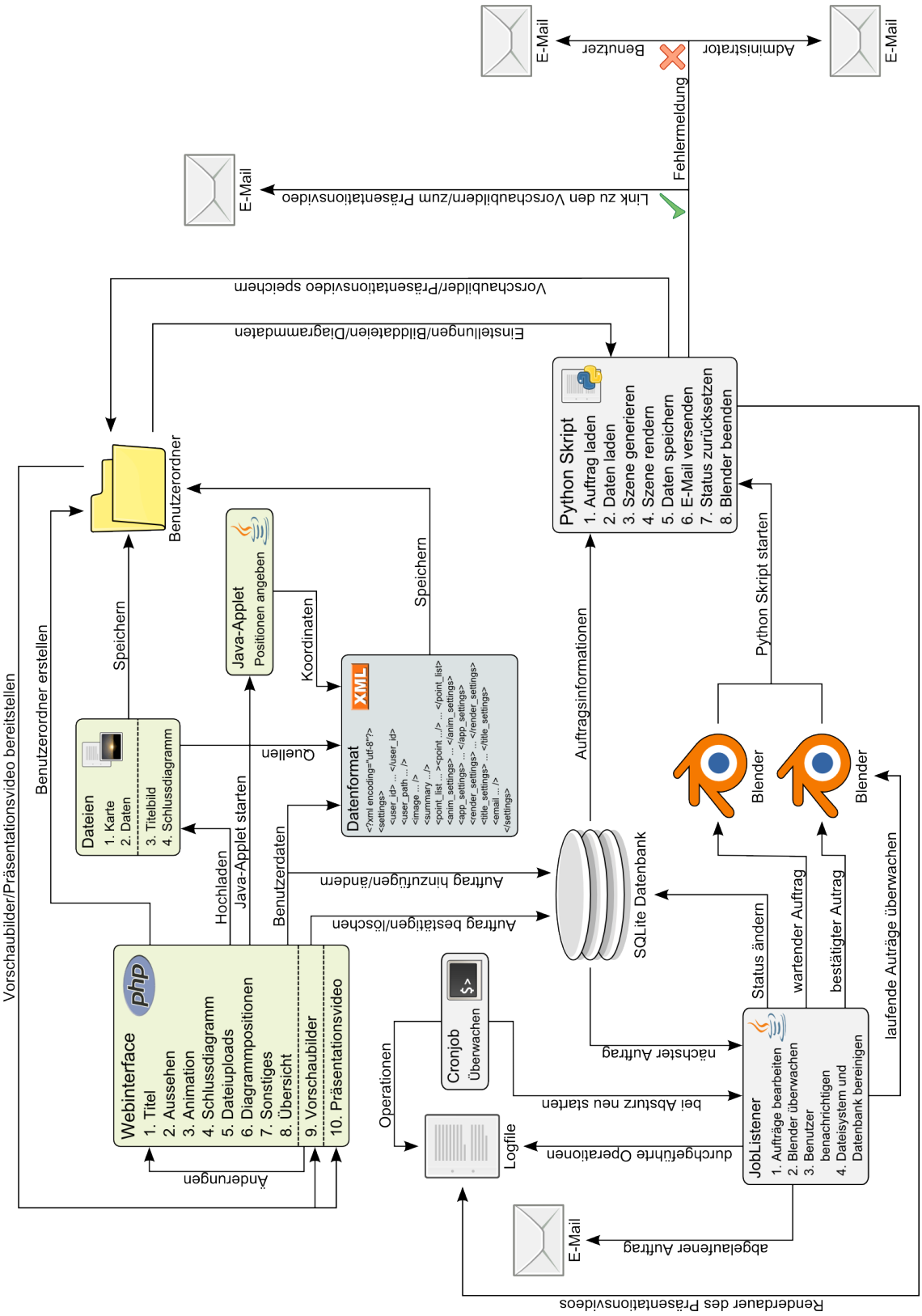

Abbildung 4.2.: komplexes Ablaufdiagramm

Auch im komplexen Ablaufdiagramm beginnt ein Auftrag mit dem Webinterface. Dort werden nacheinander die ersten acht Schritte in Form der verschiedenen Teilformulare, wie in Abschnitt 3.2 beschrieben, abgearbeitet. Bereits im ersten Teilformular wird für jeden Benutzer eine Auftragsnummer generiert und auf deren Basis ein Ordner für Daten, Einstellungen und Ergebnisse erzeugt. Dieser Benutzerordner speichert daraufhin in Schritt 5 die entsprechenden Daten. Hierbei ist das Hochladen der Karte und der Diagrammdaten zwingend erforderlich. Zudem können optional weitere Dateien, wie Titelgrafik oder Schlussdiagramm verwendet werden. Im nachfolgenden Schritt 6 lassen sich nun die Diagrammpositionen angeben, welche zunächst zwischengespeichert und erst nach Absenden des Auftrags zur Einstellungsdatei hinzugefügt werden. Dies geschieht mit Hilfe des dafür vorgesehenen Java-Applets. Nach dem Absenden des Auftrags erzeugt ein speziell hierfür entwickeltes PHP-Skript eine XML-Datei, welche jegliche getätigten Einstellungen, die Pfade zu allen hochgeladenen Dateien und die zwischengespeicherten Diagrammpositionen enthält. Auch diese wird in den Benutzerordner verschoben. Wie eine solche XML-Datei im Detail aussieht, lässt sich Abschnitt 4.5 entnehmen. Weiterhin fügt das PHP-Skript anhand der zuvor generierten Nummer den entsprechenden Auftrag in die Datenbank ein, wobei es zusätzlich den Pfad zum jeweiligen Benutzerverzeichnis angibt. Wurden die gerenderten Vorschaubilder im Benutzerordner gespeichert, so kommt erneut das Webinterface ins Spiel. Dieses hält neben einer Schaltfläche für das Ändern, auch die Option zum Bestätigen oder Löschen eines Auftrags bereit. Während das Ändern auf die bereits zuvor beschriebene Art und Weise durchgeführt wird, lassen sich die beiden anderen Möglichkeiten durch einen einfachen Zugriff auf die Datenbank realisieren. Im ersten Fall verändert das PHP-Skript lediglich den Status, um den Auftrag als bestätigt zu markieren. Soll dieser aber gelöscht werden, so entfernt das PHP-Skript den aktuellen Eintrag aus der Datenbank.
Das nächste Element des Ablaufdiagramms ist der bereits vorgestellte JobListener, dessen vier Hauptaufgaben nachfolgend erläutert werden. Die angegebene Reihenfolge stellt dabei deren Prioritäten dar. So ist die Bearbeitung jeglicher Art von Aufträgen die wichtigste aller Aufgaben. Hierbei durchsucht der JobListener die Datenbank nach neuen bzw. geänderten oder bestätigten Aufträgen. Konnte er ersteres finden, so startet er eine Instanz von Blender und teilt dem entsprechenden Python-Skript über die Datenbank mit, welcher Auftrag zu bearbeiten ist. Dieses ändert zunächst den Status des entsprechenden Auftrags, um dessen Bearbeitung zu signalisieren und somit zu verhindern, dass weitere Aufträge der gleichen Art betrachtet werden. Daraufhin behandelt es den jeweiligen Auftrag. Während dieses Vorgangs prüft der JobListener in vorgegebenen Abständen, ob Blender gegebenenfalls beendet wurde oder abgestürzt ist, ohne den Status des Auftrags zu aktualisieren. Ist dies der Fall, so setzt der JobListener den Status des entsprechenden Datenbankeintrags auf den vorherigen Wert zurück. Hierdurch ist es zwar nötig, den betroffenen Auftrag erneut zu bearbeiten, allerdings wird der Service auf diese Art und Weise nicht blockiert, falls Blender den Rendervorgang abbricht. Konnte der Joblistener einen bestätigten Auftrag finden, so wird auch hier zunächst

dessen Status geändert und daraufhin eine Instanz von Blender gestartet. Diese übernimmt den kompletten Prozess des Renderns und kann gegebenenfalls parallel zu einer zweiten Instanz mit der oben erwähnten Aufgabe ausgeführt werden. Zwei Instanzen des gleichen Typs sind allerdings zu keiner Zeit möglich. Eine Überwachung von Blender wird selbstverständlich auch hier durchgeführt und ist gerade beim Rendern eines Videos besonders wichtig.
Die dritte Aufgabe des JobListeners hängt eng mit der Bereinigung des Dateisystems und der Datenbank zusammen. Hierbei handelt es sich um eine regelmäßig durchgeführte Überprüfung des aktuellen Status aller Aufträge, wobei drei Situationen besonders zu beachten sind. Zunächst kann es vorkommen, dass ein Benutzer zwar einen Auftrag startet und eventuell Dateien hochlädt, aber diesen überhaupt nicht absendet. In einem solchen Fall werden nach einem vorgegebenen Zeitraum alle erzeugten Daten samt Benutzerordner gelöscht. Da hier kein Eintrag in der Datenbank vorliegt, ist es auch nicht nötig, diese weiter zu betrachten. Der zweite Fall beschreibt die Situation in der bereits alle Vorschaubilder gerendert und eine E-Mail an den Benutzer versandt wurde. Wie bereits in Abschnitt 3.3 beschrieben muss dieser nun den Auftrag entweder für die weitere Bearbeitung freigeben oder entsprechend abändern. Wird über einen längeren Zeitraum keine dieser Aktionen durchgeführt, so benachrichtigt der JobListener zunächst den Benutzer mit dem Hinweis dies zu tun. Erfolgt auch hierauf keine Reaktion, so löscht das Java-Programm den entsprechenden Ordner samt Inhalt und entfernt den abgelaufenen Auftrag aus der Datenbank. Nach einem ähnlichen Prinzip lässt sich mit fertiggestellten Aufträgen verfahren. Auch hier wird nach einem zuvor festgelegten Zeitraum der Benutzer per E-Mail über einen abgelaufenen Auftrag benachrichtigt. Diesem steht daraufhin eine gewisse Zeit zur Verfügung, um das fertige Präsentationsvideo herunterzuladen bevor dieses und alle anderen Daten endgültig gelöscht werden. Führt der JobListener eine der oben erwähnten Aufgaben durch, so protokolliert er diese, unter Angabe des aktuellen Datums und der Uhrzeit, in einem speziell hierfür vorgesehenen Logfile.
Allerdings kann es nun vorkommen, dass auch der JobListener abstürzt oder durch ein Herunterfahren des Systems beendet und bei erneuten Hochfahren nicht wieder gestartet wird. Genau aus diesem Grund lassen sich unter Linux sogenannte Cronjobs erstellen, welche in einem vorgegebenen Intervall beliebige Aufgaben ausführen können. Beim Starten des Systems werden alle gelisteten Cronjobs wieder aufgenommen und verrichten ihre gewohnte Arbeit. Speziell für den Fall des JobListeners überprüft nun ein solcher Cronjob in einem vorgegebenen Intervall, ob das Java-Programm ständig im Hintergrund läuft und somit auch die Verfügbarkeit des gesamten Services sichergestellt ist. Dies geschieht über ein Shell-Skript, welches den entsprechenden Prozess unter allen laufenden ausfindig macht. Ließ sich keine Instanz des JobListeners finden, so wird er erneut gestartet und dieser Vorgang in einer Protokolldatei eingetragen.
Um nun wieder zum tatsächlichen Verlauf eines Auftrags zurückzukehren, wird nachfolgend das weitere Vorgehen nach dem Starten einer der beiden Blender-Instanzen erläutert. Der Ablauf dieser Vorgänge ist prinzipiell der gleiche, wobei sich lediglich anhand des Ergebnis-

ses ein Unterschied feststellen lässt. Sofort nach dem Start von Blender wird mit Hilfe von Scriptlinks das benötigte Python-Skript zum Erzeugen der dreidimensionalen Szene geöffnet und die dort aufgelisteten Operationen durchgeführt. Dieses Vorgehen lässt sich nun in sieben Schritte einteilen. Zunächst greift Blender auf die SQLite-Datenbank zurück, um von dort alle benötigten Informationen über den aktuellen Auftrag zu erhalten. Hierzu zählen sowohl die aktuelle Auftragsnummer, der Pfad zum entsprechenden Verzeichnis und die Information darüber, ob das finale Präsentationsvideo oder lediglich die Vorschaubilder gerendert werden sollen. Wie eine solche Verbindung mit der Datenbank funktioniert, wurde bereits in Kapital 2 vorgestellt. Anhand dieser Informationen lassen sich nun die benötigten Daten aus dem Benutzerorder laden. Dies beschränkt sich zunächst nur auf die in der XML-Datei gespeicherten Einstellungen. Erst im dritten Schritt, der Generierung der gesamten Szene und aller Animationen, werden auch die verfügbaren Bilddateien in Blender eingebunden. Nach dem erfolgreichen Erzeugen der Szene, startet das Python-Skript anschließend den Renderprozess. Auf welche Art dies nun geschieht, ist selbstverständlich vom derzeitigen Status des aktuellen Auftrags abhängig. Diese Information erhält Blender aber bereits im ersten Schritt bei dem Zugriff auf die Datenbank. Ist der Rendervorgang abgeschlossen, so speichert Blender zunächst die Ergebnisse im Benutzerverzeichnis ab und versendet daraufhin eine E-Mail, welche über den erfolgreichen Renderprozess berichtet und die Links zur personalisierten Webseite bereitstellt. Dort lassen sich nun die jeweiligen Ergebnisse betrachten bzw. herunterladen. Abschließend wird der Status des aktuellen Auftrags auf den jeweils nachfolgenden gesetzt und Blender beendet. Traten allerdings während der Ausführung des Python-Skripts Fehler auf, so wird eine E-Mail mit der entsprechenden Benachrichtigung an den Benutzer gesendet. Zudem erhält auch der ChartFlight Service selbst ein E-Mail mit einer Auflistung aller aufgetretenen Fehler. Weiterhin sollte noch erwähnt werden, dass auch Blender Informationen in einer Protokolldatei aufzeichnet. Dies bezieht sich allerdings nur auf die Dauer des Rendervorgangs der jeweiligen Präsentationsvideos.

4.2. Die Datenbank

Dieser Abschnitt stellt die im ChartFlight Service verwendete Datenbank vor. Hierbei handelt es sich um eine SQLite-Datenbank, welche bereits in Kapitel 2 betrachtet wurden. Die dort erwähnten Vorteile gelten auch in diesem Fall. So präsentiert sie sich als eine einzelne Datei, die aber die volle Funktionalität einer Datenbank mitbringt. Zudem lässt sich mit allen verwendeten Programmiersprachen komfortabel auf eine solche Datenbank zugreifen. Der letzte Grund warum SQLite verwendet wurde, bezieht sich darauf, dass kein zusätzlicher Datenbankserver zu starten ist und somit, bei Entwicklung einer entsprechenden Benutzerschnittstelle, auch eine lokale Installation des Services möglich wäre.
Die für den Service verwendete Datenbank besitzt nun zwei unterschiedliche Tabellen. Während sich die erste zum Speichern der verschiedenen Aufträge verwenden lässt, nimmt die

zweite den jeweils nächsten zu bearbeitenden Auftrag entgegen. So muss lediglich dort nachgeschaut werden, welcher Auftrag als nächstes zu rendern und auf welche Art und Weise dies zu tun ist. Die beiden nachfolgenden Tabellen stellen den Inhalt der Datenbank anhand zufällig gewählter Einträge dar und decken dabei möglichst viele unterschiedliche Fälle ab. Hierbei ist allerdings zu erwähnen, dass aus Platzgründen der Pfad zum jeweiligen Benutzerordner gekürzt wurde und nur die letzten beiden Schritte angezeigt werden.

user_id	directory	email	state	date	notified
0809291523162497	.../users/0809291523162497/	u1@domain.de	done	080930101645	0
0810021929061134	.../users/0810021929061134/	u2@domain.de	rendering	081002195312	0
0810121012234492	.../users/0810121012234492/	u3@domain.de	confirmed	081012103433	0
0810122334279645	.../users/0810122334279645/	u4@domain.de	rejected	081013004521	0
0810152233159283	.../users/0810152233159283/	u5@domain.de	preview	081015230332	0
0810161316347648	.../users/0810161316347648/	u6@domain.de	waiting	081016134604	0

Tabelle 4.1.: Tabelle zum Speichern der verschiedenen Aufträge

Zunächst besitzt jeder Auftrag eine eindeutige Auftragsnummer. Aus diesem Grund wurde die erste Spalte der Tabelle als Primärschlüssel definiert und dient im Weiteren somit zur Identifizierung jedes einzelnen Auftrags. Wie in der zweiten Spalte zu sehen ist, ließ sich auch auf Basis dieser Auftragsnummer das Benutzerverzeichnis erzeugen. Daraufhin folgt die E-Mail-Adresse des Benutzers. Diese wurde ebenfalls in die Datenbank aufgenommen, damit der JobListener beim Verschicken von E-Mails nicht auf die im entsprechenden Verzeichnis platzierte XML-Datei zugreifen muss. Die vierte Spalte ist für den ChartFlight Service besonders wichtig, da diese den aktuellen Status eines jeden Auftrags angibt. Je nach Eintrag wird hier unterschiedlich verfahren, worauf wir später noch einmal zurückkommen. In **date** soll das aktuelle Datum samt Uhrzeit gespeichert werden. Während der Zeitpunkt, an dem der Auftrag über das Webinterface gestartet wurde, in der jeweiligen Auftragsnummer kodiert ist, kann die Spalte **date**, je nach aktuellem Status, einen unterschiedlichen Wert enthalten. Abschließend gibt **notified** an, ob ein Benutzer bereits über einen demnächst ablaufenden Auftrag informiert wurde.
Die zweite Tabelle in der SQLite-Datenbank wird lediglich dafür verwendet, den nächsten zu bearbeitenden Auftrag zwischenzuspeichern. So muss dieser nur ein einziges Mal aus der Menge aller unterschiedlichen Aufträge herausgefiltert werden und lässt sich somit über einen simplen Zugriff auf die zweite Tabelle der Datenbank identifizieren. Dies vereinfacht vor allem die Arbeitsweise des Python-Skripts. So erhält es beispielsweise über eine zusätzliche Spalte die Information, ob für den aktuellen Auftrag Vorschaubilder oder das finale Präsentationsvideo gerendert werden sollen. Die nachfolgende Tabelle verdeutlicht dies anhand zweier Einträge.

user_id	render_type
0810041222093897	preview_images
0810162129567511	video

Tabelle 4.2.: Tabelle der zu bearbeitenden Aufträge

Nachdem nun die einzelnen Tabellen der Datenbank vorgestellt wurden, sollen die nachfolgenden Beschreibungen erläutern, welche Zustände ein einzelner Auftrag im Laufe seiner Bearbeitung annehmen kann. Hierzu stehen acht solcher Zustände zur Verfügung, wobei der letzte nur im Falle eines Fehlers angenommen wird. Im Einzelnen sind dies:

- waiting
- preview
- confirmed
- done
- rendering_preview
- rejected
- rendering
- error

Wird nun ein neuer Auftrag nach dessen Absenden in die Haupttabelle der Datenbank eingetragen, so erhält dieser zunächst den Zustand ***waiting***. Wie in Tabelle 4.1 zu sehen ist, trägt das PHP-Skript ebenso das Datum und die aktuelle Uhrzeit zum Zeitpunkt des Absendens in die dafür vorgesehene Spalte ein. Vergleicht man diese mit der Auftragsnummer, welche den Zeitpunkt zu Beginn des Auftrags enthält, so lässt sich erkennen, dass der Benutzer eine gewisse Zeit für das Ausfüllen der Formulare benötigt hat. Auf eine ähnliche Weise wird auch nach jedem einzelnen Rendervorgang die Spalte **date** des entsprechenden Auftrags aktualisiert. Dies ist vor allem in Bezug auf abgelaufene Aufträge wichtig, da auf Basis des in **date** vorhandenen Wertes die jeweiligen Berechnungen hierfür durchgeführt werden.
Wie die Bezeichnung ***waiting*** bereits erahnen lässt, muss ein Auftrag in einem solchen Zustand zunächst auf die Bearbeitung warten. Dies bedeutet, dass alle vor ihm in die Datenbank eingetragenen Zeilen bevorzugt behandelt werden, wenn bei ihnen das Feld **state** ebenfalls den Wert ***waiting*** aufweist. Noch bevor aber ein solcher Auftrag bearbeitet wird, sucht der JobListener nach Einträgen mit dem Zustand ***rejected***. Hierbei handelt es sich um abgelehnte und daraufhin geändert Aufträge. Um den Benutzer nach einem solchen Vorgang nicht zu lange warten zu lassen, wird deshalb auch zwischen diesen beiden Zuständen unterschieden. Wurde nun ein Auftrag im Zustand ***waiting*** oder ***rejected*** gefunden, so ist die Vorgehensweise in beiden Fällen gleich. Durch Eintragen der jeweiligen Auftragsnummer und des entsprechenden Rendertyps in die zweite Tabelle wird Blender daraufhin signalisiert, dass für den nachfolgenden Auftrag Vorschaubilder zu rendern sind. Bevor das Python-Skript aber diesen Vorgang startet, löscht es zunächst den gelesenen Eintrag aus der zweiten Tabelle und ändert den Zustand des zu bearbeitenden Auftrags in ***rendering_preview***. Hierdurch wird dem JobListener mitgeteilt, dass eine Instanz von Blender gerade Vorschaubilder rendert und es somit nicht möglich ist, einen weiteren Auftrag dieser Art zu bearbeiten. Wurde alles erfolgreich durchgeführt, so ändert das Python-Skript den Zustand des bearbeiteten Auftrags in ***preview*** und beendet Blender.
An diesem Punkt muss der Benutzer erneut eingreifen und das weitere Vorgehen bestimmen. Ist er mit den gerenderten Vorschaubildern nicht zufrieden, so ändert er den Auftrag entsprechend ab. Intern bedeutet dies, dass ein weiteres Rendern von Vorschaubildern erforderlich und der Zustand des Auftrags somit in ***rejected*** zu ändern ist. In diesem Fall wird daraufhin, wie oben beschreiben, verfahren. Bestätigt der Benutzer allerdings den Auftrag, so wird

dessen Zustand auf ***confirmed*** gesetzt. Der JobListener erhält hierdurch die Information, dass ein Auftrag zum abschließenden Rendervorgang bereitsteht. Auch hier werden die Aufträge abhängig von der Position der jeweiligen Zeile in der Datenbank behandelt, wobei der JobListener lediglich Einträge mit dem Zustand ***confirmed*** betrachtet. Der erste dieser Einträge wird unter Angabe seiner eindeutigen Auftragsnummer und des Rendertyps ***video*** zur zweiten Tabelle hinzugefügt. Hierauf kann die gestartete Blender-Instanz nun zugreifen und analog zum Rendern der Vorschaubilder dessen Zustand ändern. Durch den Wert ***rendering*** wird auch hier dem JobListener signalisiert, dass sich gerade ein Auftrag in Bearbeitung befindet und das Starten eines weiteren somit blockiert ist. Nach Beenden des Rendervorgangs setzt das Python-Skript den entsprechenden Zustand auf den Wert ***done***. Die Bearbeitung von Aufträgen mit einem solchen Zustand ist somit abgeschlossen und aufgrund dessen werden sie bei den oben genannten Datenbankzugriffen nicht weiter berücksichtigt. Nach einem vorgegebenen Zeitraum entfernt der JobListener diese dann aus der Datenbank. Zuvor wird allerdings der jeweilige Benutzer per E-Mail benachrichtigt.
Tritt während der Ausführung des Python-Skripts ein Fehler auf, so wird der Zustand des entsprechenden Auftrags auf den Wert ***error*** gesetzt. Dieser lässt sich nur durch ein manuelles Eingreifen wieder zurücksetzen. Somit wird verhindert, dass der fehlerhafte Auftrag den gesamten Service unnötigerweise blockiert. Ähnlich wie fertiggestellte Aufträge werden deshalb auch fehlerhafte vom JobListener ignoriert. Abschließend visualisiert Abbildung 4.3 den oben beschriebenen Prozess anhand eines Flussdiagramms. Hierbei wurde davon ausgegangen, dass dieser Vorgang fehlerfrei abläuft.

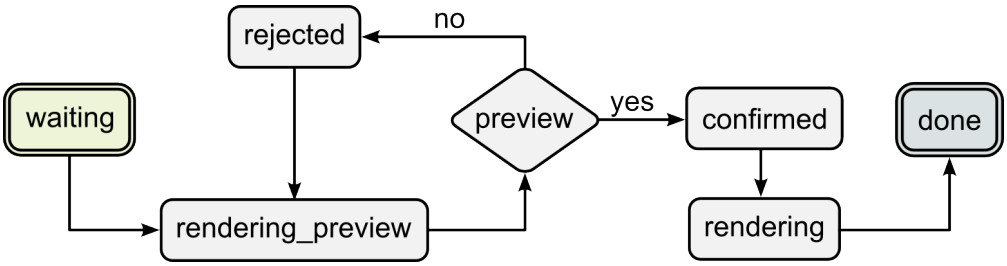

Abbildung 4.3.: Ablauf eines Auftrags innerhalb der Datenbank

4.3. Das Backend

Dieser Abschnitt beschäftigt sich mit dem Backend des ChartFlight Services und dessen Implementierung. Wie schon erwähnt, ist hierzu die Programmiersprache Python notwendig, welche in Kapitel 2 im Zusammenhang mit Blender bereits vorgestellt wurde. Zunächst betrachten wir allerdings die eigens hierfür entwickelten Klassen und Funktionen. Dabei wird aber nicht auf spezielle Implementierungsdetails eingegangen, sondern nur auf die Struktur und das Zusammenarbeiten dieser. Im Gegensatz hierzu lässt sich im darauffolgenden Unter-

abschnitt ein detaillierterer Blick in die einzelnen Methoden der jeweiligen Klassen werfen. Dort werden anhand diverser Beispiele des Quellcodes die unterschiedlichen Vorgehensweisen präsentiert. Die hierbei aufgetretenen Probleme und deren Behandlung diskutiert ein eigener Abschnitt am Ende des Kapitels.

4.3.1. Überblick

Das Backend besteht aus drei selbst entwickelten Modulen und einem bereits verfügbaren. Bei letzterem handelt es sich um das in Kapitel 2 vorgestellte SimpleMail, auf welches hier nicht näher eingegangen werden soll. Die ersten drei Module verfolgen unterschiedliche Aufgaben, worunter sich die Generierung der Szene und der zugehörigen Animationen sowie das Einlesen und Speichern sämtlicher Einstellungen befindet. Im Einzelnen sind dies:

`diagram_factory.py`	Hauptmodul zum Erzeugen der Szene und der Animationen
`user_data.py`	Zwischenspeichern jeglicher Einstellungen
`data_parsing.py`	Einlesen jeglicher Einstellungen

Tabelle 4.3.: Auflistung der verwendeten Module

Das Hauptmodul

Das Hauptmodul beinhaltet eine ganze Reihe von Klassen zur Erzeugung der verschiedenen Diagrammtypen und weiteren wichtigen Bestandteilen der Szene. Ebenso ist dies der Ausgangspunkt für die Bearbeitung eines Auftrags, da von dort aus die main-Funktion aufgerufen und somit der gesamte Hintergrundprozess gestartet wird. Diese befindet sich am Ende des Hauptmoduls, welches sich zusätzlich in fünf weitere Abschnitte gliedern lässt. Die darin enthaltenen Klassen sollen im Nachfolgenden vorgestellt und deren Funktion näher erläutert werden. Tabelle 4.4 zeigt eine Auflistung der übrigen fünf Teilbereiche.

Hauptklasse	**Hilfsklassen**	**sonstige Klassen**	**3D Diagramme**	**Schlussdiagramm**
DiagramFactory	Materials	CameraAnimation	BasicChart	Basic2DChart
	Ipos	Map	PieChart	Pie2DChart
	Utils	Lighting	RingChart	Ring2DChart
		Title	BarChart	Bar2DChart
			LineChart	Line2DChart
			Key	Key2D
				SummaryChart

Tabelle 4.4.: Auflistung der Klassen des Hauptmoduls

Wie zu sehen, ist ***DiagramFactory*** die Hauptklasse des Moduls und somit auch des gesamten Backends. Hierin wird auf Basis der übrigen Klassen die Szene samt aller benötigten Animationen generiert und anschließend gerendert. Zudem hat diese Klasse die Aufgabe unter Zuhilfenahme der Module `user_data.py` und `data_parsing.py` sowohl die Einstellungen

der Konfigurationsdatei als auch sämtliche Benutzerangaben des aktuellen Auftrags zu laden und über die jeweiligen Objekte bereitzustellen. Auch zur Benachrichtigung eines Benutzers per E-Mail und zum Hinzufügen eines Eintrags zur Protokolldatei stehen entsprechende Methoden bereit. Für ersteres wird zusätzlich das Modul *SimpleMail* verwendet.
In der zweiten Spalte von Tabelle 4.4 sind drei wichtige Hilfsklassen aufgeführt, welche verschiedene Aufgaben während des Generierens der Szene und ihrer Animationen erfüllen. Hauptsächlich fassen deren Methoden eine Reihe von einzelnen Befehlen der Blender API so zusammen, dass sie an der entsprechenden Stelle komfortabel eingesetzt werden können. So stellt die Klasse ***Materials*** diverse statische Methoden zur Erzeugung der unterschiedlichsten Materialien, Farben und Texturen zur Verfügung. Hierzu zählen unter anderem solche zur Berechnung eines Farbverlaufs oder zufälligen Auswahl von einzelner Farben. Weiterhin kann hiermit die Texturierung der Untergrundebene unter Verwendung der hochgeladenen Karte vorgenommen werden.
Die zweite Klasse bietet Möglichkeiten zur Erzeugung verschiedener Animationen. Auch hierbei handelt es sich um statische Methoden. Diese nehmen zunächst ein zu animierendes Objekt entgegen und mit Hilfe zusätzlicher Parameter wird daraufhin der genaue Verlauf eine solchen Animation gesteuert. Zudem lassen sich verschiedene Typen von Animationen wählen. So kann ein Objekt entweder durch Einblenden oder durch ein langsames Aufbauen aus der zugrundeliegenden Karte dargestellt werden.
Die dritte Hilfsklasse mit der Bezeichnung ***Utils*** beinhaltet eine ganze Reihe an mathematischen Methoden. Hier sind sowohl Funktion der Linearen Algebra, als auch solche zur Berechnung minimaler bzw. maximaler Werte eines eingelesenen Datensatzes implementiert.
Im Gegensatz zu den gerade beschriebenen Hilfsklassen enthalten die nachfolgenden Klassen keine statische Methoden. Hierbei betrachten wir zunächst solche, die unter dem Begriff **sonstige Klassen** aufgeführt sind. Aus dem genannten Grund müssen zur Verwendung dieser zuerst die entsprechenden Objekte erzeugt werden, was in der Hauptklasse ***DiagramFactory*** und dort in der Methode ***createScene()*** geschieht. Die Klasse ***CameraAnimation*** beinhaltet, wie der Name schon sagt, Methoden zur Erzeugung der Kamera, benötigter Hilfsobjekte und den entsprechenden Animationen. Zudem speichert sie ebenfalls Referenzen auf die zuvor erzeugten Blender-Objekte. Wichtig für die nachfolgende Betrachtung des Python-Skripts ist die Unterscheidung von Objekten als Instanz einer Klasse und solchen, die Blender intern verwendet und in der 3D-Ansicht darstellt. Diese werden künftig als Blender-Objekt bezeichnet. Nach ähnlichem Prinzip ist die Klasse ***Map*** aufgebaut. Diese beinhaltet Methoden zum Erzeugen einer Ebene und dem Zuweisen der hochgeladenen Karte unter Zuhilfenahme der Klasse ***Materials***. Auch hier wird eine Referenz auf das zurückgelieferte Blender-Objekt gespeichert. Für die Beleuchtung der Szene ist die Klasse ***Lighting*** zuständig. Hierbei wird für ein einheitliches Licht unter Verwendung sogenannter Sun-Objekte gesorgt. Zu guter Letzt befindet sich am Ende der dritten Spalte von Tabelle 4.4 eine Klasse zur Erzeugung und Verwaltung des Titelbildschirms. Diese übernimmt Aufgaben, wie das Laden eventuell

verfügbarer Grafiken oder das Generieren der entsprechenden Textobjekte. Somit verwendet auch sie Methoden der Klasse ***Materials***.
Die beiden letzten Spalten beinhalten Klassen zur Generierung drei- oder zweidimensionaler Diagramme. Letzteres wird allerdings ausschließlich beim Erzeugen eines Schlussdiagramms benötigt. Hierfür existiert zusätzlich die Klasse ***SummaryChart***, welche bei dessen Generieren auf die entsprechenden Diagramme zurückgreift. Sowohl für drei- als auch zweidimensionale Diagramme steht zunächst eine Basisklasse an oberer Position. Diese implementiert die benötigte Grundfunktionalität und dient somit als Ausgangspunkt der spezialisierten Diagrammklassen. Erst dort finden sich die entsprechenden Methoden zur Erzeugung und Animation der jeweiligen Diagramme. Wie hierbei zu erkennen ist, übernimmt eine solche Klasse nicht das Generieren eines einzelnen Diagramms, sondern aller in der Szene bzw. im Schlussdiagramm benötigten. Abhängig vom gewählten Typ mussten allerdings verschiedene Member-Variablen und Hilfsmethoden hinzugefügt werden. Eine detailliertere Beschreibung über den Inhalt und die Funktionsweise dieser Klassen wird in den nachfolgenden Unterabschnitten gegeben. Zu guter Letzt sind noch die Aufgaben der Klassen ***Key*** und ***Key2D*** zu betrachten. Diese sind für die Generierung von diagrammspezifischen Legenden verantwortlich und enthalten somit Methoden zum Erzeugen zweidimensionaler Formen und Texte.

Das Modul user_data.py

Dieses Modul beinhaltet Klassen zur Speicherung der eingelesenen Daten zur Laufzeit. Hierbei sind sowohl die Grundeinstellungen aus der Konfigurationsdatei als auch die durch den Benutzer angegebenen bzw. hochgeladenen Daten betroffen. Folgenden Klassen sind im Modul `user_data.py` verfügbar:

Benutzerdaten		**Grundeinstellungen**
UserData	Image	ConfigINI
Summary	AnimSettings	
AppSettings	RenderSettings	
TitleSettings	PointList	
Point	Diagram	
Row		

Tabelle 4.5.: Auflistung der Klassen des Moduls user_data.py

Wie in Tabelle 4.5 zu erkennen, stehen eine ganze Reihe von Klassen für das Zwischenspeichern der Benutzerdaten zur Verfügung. Hierbei gilt ***UserData*** als Hauptklasse der Benutzerdaten, in welcher die übrigen entweder direkt oder indirekt in einer weiteren Klasse auftauchen. Dies bedeutet also, dass darüber auf die gesamten vom Benutzer angegebenen Daten zugegriffen werden kann. Die Klasse ***UserData*** enthält unter anderem Member-Variablen für die Auftragsnummer, das Benutzerverzeichnis und die angegebene E-Mail-Adresse sowie für Instanzen aller oben aufgeführten Klassen mit Ausnahme der letzten drei. Diese finden

sich allerdings in der Klasse ***PointList*** und den darin verwendeten Objekten wieder.
Die nächsten beiden Klassen speichern Informationen über die zugrundeliegende Grafik und das Schlussdiagramm. Erstere beinhaltet Angaben, wie die Größe der Karte oder der Pfad zur entsprechenden Bilddatei. Die Klasse ***Summary*** hingegen ist ein wenig komplexer und kapselt Werte wie den Diagrammtyp, die Überschrift der Schlussdiagramms oder, im Falle einer hochgeladenen Grafik, den jeweiligen Pfad zu dieser Datei. Soll das Schlussdiagramm allerdings *generiert* werden, so beinhaltet sie anstatt des oben genannten Pfades die zur Erzeugung benötigten Daten in Form eines Objekts der Klasse ***Diagram***.
Weiterhin existieren vier Klassen mit dem Postfix »Settings«, welche diverse Benutzereingaben entgegennehmen. Darunter befinden sich solche zur Speicherung aller Einstellungen bezüglich der Animationen und des Aussehens. Ebenso existiert mit der Klasse ***RenderSettings*** eine Möglichkeit zum Speichern diverser Optionen hinsichtlich des Rendervorgangs. Die Klasse ***TitleSettings*** dient zur Sicherung der unterschiedlichen Angaben in Bezug auf den Titelbildschirm. Auch hier kann wahlweise eine Bilddatei verwendet und die übrigen Einstellungen vernachlässigt werden.
Die darauffolgenden Klassen werden allesamt zum Speichern der Datensätze aus der CSV-Datei verwendet, damit nach deren Einlesen die Daten in einer leicht zugreifbaren Form bereitstehen. Die Klasse ***PointList*** kapselt hierbei sämtliche verfügbaren Daten unter Zuhilfenahme der Klassen ***Point***, ***Diagram*** und ***Row***. Im Prinzip ist sie nichts anderes als eine Liste von ***Point*** Objekten und einigen Zusatzinformationen, wie beispielsweise dem Diagrammtyp. Ein solcher Punkt enthält nun die Informationen darüber, an welcher Stelle auf der Karte ein Diagramm erzeugt und welcher Datensatz hierfür verwendet werden soll. Dessen einzelne Zeilen werden durch je ein Objekt der Klasse ***Row*** repräsentiert und mit Hilfe der Klasse ***Diagram*** gekapselt. Während ersteres eine Member-Variable zum Speichern der jeweiligen Zeilenköpfe bietet, enthält ein Objekt vom Typ ***Diagram*** eine zusätzliche Liste mit allen Spaltenköpfen. Weiterhin lässt sich dort auch der entsprechende Titel speichern.
Zu guter Letzt soll noch die Klasse ***ConfigINI*** erwähnt werden, welche diverse Werte der Konfigurationsdatei nach deren Einlesen kapselt. Die über eine Instanz dieser Klasse verfügbaren Einstellungen werden für jeden Benutzer gleichermaßen verwendet und lassen sich nicht von diesem verändern.

Das Modul data_parser.py

Das letzte Modul ist für das Einlesen sämtlicher Daten verantwortlich und steht in direkter Verbindung mit dem vorherigen. Dieses enthält lediglich drei unterschiedliche Klassen, welche in Tabelle 4.6 aufgelistet sind und die dort dargestellten Aufgaben erfüllen. So existiert beispielsweise eine spezielle Klasse zum Einlesen der XML-Datei und den darin enthaltenen Benutzerdaten. Zu deren Realisierung wurde auf ein bereits in Python integriertes Modul zum Bearbeiten von XML-Dateien zurückgegriffen. Dieses bietet einen komfortablen Zugriff

auf eine geöffnete XML-Datei, sodass lediglich die benötigten Werte herausgefiltert und mit Hilfe eines Objekts der Klasse ***UserData*** gespeichert werden müssen. Weiterhin verwendet ***UserData*** intern ein Objekt des Typs ***CSVParser***, welches das Laden von Datensätzen der einzelnen Diagramme ermöglicht. Zusätzlich werden hierfür die oben vorgestellten Klassen zur Datenspeicherung benötigt.

Benutzerdaten	Diagrammdaten	Konfigurationsdaten
UDParser	CSVParser	CFGParser

Tabelle 4.6.: Auflistung der Klassen des Moduls data_parser.py

Die Klasse ***CSVParser*** basiert ebenfalls auf einem bereits verfügbaren Modul. Auch dieses wird bei einer standardmäßigen Installation von Python mitgeliefert und bietet die grundlegende Funktionalität zum Bearbeiten von CSV-Dateien. Da die angebotenen Funktionen allerdings zum Einlesen von Diagrammdaten für den ChartFlight Service nicht ausreichend sind, mussten verschiedene Methoden zusätzlich implementiert werden. Die daraufhin erhaltenen Daten lassen sich mit Hilfe der speziell hierfür vorgesehenen Klassen zur Datenspeicherung aus dem vorherigen Abschnitt verwalten.
Unabhängig davon arbeitet die Klasse ***CFGParser***, welche die Daten aus der Konfigurationsdatei liest und diese in einem Objekt des Typs ***ConfigINI*** speichert. Auch hierzu wurde ein vorgefertigtes Modul zum Laden solcher Konfigurationsdateien verwendet. Dieses liefert nach Angabe des Namens eines bestimmten Eintrags dessen Wert zurück.

4.3.2. Erzeugen und Rendern einer Szene

Nach dem Vorstellen der einzelnen Klassen und deren Aufgaben betrachtet der vorliegende Abschnitt nun, wie diese zusammenarbeiten, um die jeweilige Szene zu erzeugen, zu rendern und schlussendlich den Benutzer zu benachrichtigen. Da es allerdings zu viel Platz in Anspruch nehmen würde, den Prozess für jeden Diagrammtyp einzeln vorzustellen, beschränkt sich dieser Abschnitt auf die Behandlung von Kreisdiagrammen. Die übrigen lassen sich prinzipiell auf die gleiche Art und Weise verwenden, wobei lediglich die Methoden zum Erzeugen der einzelnen Blender-Objekte unterschiedlich implementiert wurden. Hierunter fällt beispielsweise auch die Positionierung der Beschriftungstexte. Weiterhin ist noch zu erwähnen, dass sich nicht alle vorgestellten Auszüge des Quellcodes vollständig sind. Die für die Erklärung des jeweiligen Sachverhalts benötigten Zeilen werden selbstverständlich im entsprechenden Listing belassen oder durch ein aussagekräftiges Kommentar ersetzt und daraufhin genauer beschrieben.
Der Startpunkt eines jeden Auftrags ist die sich am Ende des Hauptmoduls befindende main-Funktion. Wie im nachfolgenden Auszug zu sehen, wird dort ein neues Objekt der Klasse ***DiagramFactory*** erzeugt, welches für nahezu alle anderen Aufgaben verantwortlich ist.

```
1  def main():
2      df = DiagramFactory();
3
4      config_path = Blender.Get('homedir') + '/scripts/config.ini';
5
6      if (df.loadConfig(config_path)):
7          df.processJob();
8
9          if (len(df.getErrors()) > 0):
10             # send notification emails
11         else:
12             print "job successful"
13     else:
14         if (len(df.getErrors()) > 0):
15             # print errors
16
17     Blender.Quit();
```

Hiernach wird mit Hilfe der Methode ***loadConfig()*** die Konfigurationsdatei geladen. Wie in Zeile 4 zu sehen, befindet sich diese in dem gleichen Verzeichnis, in welchem auch das Hauptmodul liegt und wird mit dem Dateinamen `config.ini` angesprochen. Für den Ladevorgang selbst werden die zuvor beschriebenen Klassen ***ConfigINI*** und ***CFGParser*** verwendet. Wie bereits erwähnt, dient erstere hierbei zur Speicherung der gelesenen Werte. War das Laden der Konfigurationsdatei erfolgreich, so lässt sich durch einen Aufruf der Methode ***processJob()*** der entsprechende Auftrag durchführen. Konnte die Konfigurationsdatei allerdings nicht gefunden werden oder traten bei deren Laden Fehler auf, so gibt die Methode ***loadConfig()*** »False« zurück. Somit wird der else-Zweig in Zeile 13 ausgeführt und die zuvor in einer Liste gespeicherten Fehler über die Konsole ausgegeben. Auch in der Methode ***processJob()*** kann es durchaus zu einem Fehler kommen (Zeile 9). In diesem Fall wird allerdings ein wenig anders verfahren, da durch das erfolgreiche Laden der Konfigurationsdatei nun die Möglichkeit besteht eine E-Mail zu versenden. Je nach aufgetretenen Fehlern wird hierbei sowohl der ChartFlight Service selbst als auch der Benutzer benachrichtigt (Zeile 10). Letzteres setzt aber voraus, dass dessen E-Mail-Adresse zu dem Zeitpunkt, an welchem ein Fehler auftrat, auch zur Verfügung steht. Wurde der Auftrag aber erfolgreich durchgeführt, so gibt die main-Funktion die entsprechende Nachricht über die Konsole aus. Dies ist allerdings lediglich für Debug-Zwecke interessant. Unabhängig vom Erfolg eines Auftrags wird Blender in Zeile 17 geschlossen.

Als Nächstes erfolgt nun ein Sprung in die Methode ***processJob()*** aus Zeile 7. Diese beinhaltet die gesamte Funktionalität des Backends, was bedeutet, dass von dort aus auf die Datenbank zugegriffen, die Szene erzeugt, das Ergebnis gerendert und zudem der Benutzer benachrichtigt wird. Hierfür stehen selbstverständlich diverse Methoden zur Verfügung.

Speziell beim Erzeugen der Szene werden ebenfalls viele der oben vorgestellten Klassen verwendet. Wie bereits beim Vorstellen der Datenbank erwähnt, greift das Python-Skript zunächst nicht auf deren Haupttabelle zu, sondern erhält die Information über den nächsten zu bearbeitenden Auftrag aus der Tabelle **NextJob**. Auf Basis der sich dort befindenden Auftragsnummer kann nun der tatsächliche Eintrag in der Haupttabelle ausfindig gemacht und dessen Daten in den Member-Variablen **`__user_id`**, **`__user_dir`** und **`__user_email`** zwischengespeichert werden. Neben der Auftragsnummer enthält die Tabelle **NextJob** ebenso die Information über die Art des zu bearbeitenden Auftrags. Zu deren Speicherung wird allerdings eine lokale Variable mit dem Namen **`render_type`** verwendet. Mit Hilfe all dieser Daten lassen sich nun die Einstellungen in Form der XML-Datei laden. Hierzu greift das Python-Skript auf die eigens dafür entwickelten Module ***data_parser.py*** und ***user_data.py*** zurück. Ließen sich die Einstellungen erfolgreich einlesen, so kann daraufhin die Szene über die Methode ***createScene()*** erzeugt werden. Dies geschieht in den Zeilen 3 und 4 des nachfolgenden Listings, welches sich lediglich auf die wichtigen Bestandteile beim Bearbeiten eines Auftrags beschränkt. Hierbei werden weder die Datenbankzugriffe noch das Versenden von E-Mails detaillierter dargestellt, da deren Grundlagen bereits in Kapitel 2 behandelt wurden. Zudem enthält dieses Listing keinen Code zur Fehlerbehandlung.

```
1   # load current job from database
2
3   self.loadUserData(self.__user_dir + "/" + self.__user_id + "_settings.xml");
4   self.createScene();
5
6   if (render_type == "video"):
7       # set state to 'rendering'
8
9       self.writeLogEntry("Started rendering job - ID: " + self.__user_id);
10      self.renderAnimation();
11      self.writeLogEntry("Finished rendering job - ID: " + self.__user_id);
12
13      # send email to current user
14      # set state to 'done' and update date field
15
16  elif (render_type == "preview_images"):
17      # set state to 'rendering_preview'
18
19      self.renderPreview();
20
21      # send email to current user
22      # set state to 'preview' and update date field
```

Nach dem Erzeugen der Szene wird anschließend auf Basis des zuvor gespeicherten Rendertyps entweder das Präsentationsvideo oder die Vorschaubilder gerendert (Zeile 6, 16). Hierfür ist zunächst ein Ändern des Status in ***rendering*** bzw. ***rendering_preview*** innerhalb der Datenbank erforderlich und erst daraufhin kann der entsprechende Auftrag tatsächlich gerendert werden (Zeile 10, 19). Im Falle des finalen Renderns erfolgen zudem die jeweiligen Einträge in die Protokolldatei (Zeile 9, 11). Wurde dieser Vorgang abgeschlossen, so benachrichtigt das Python-Skript den Benutzer per E-Mail, welche neben einem kurzen Erklärungstext auch den entsprechenden Link und weitere Informationen über den Auftrag enthält. Hierzu zählen unter anderem dessen Nummer oder die zu erwartende Länge des Präsentationsvideos. Anschließend wird der Status des Auftrags innerhalb der Datenbank in ***done*** bzw. ***preview*** geändert und zudem im Feld **date** das aktuelle Datum samt Uhrzeit eingetragen (Zeile 14, 22). Dieses ist über das bereits in Python integrierte Modul *datetime* zugreifbar.
Im Nachfolgenden sollen nun einige Methoden und deren Funktionalitäten genauer betrachtet werden. Hierbei gilt ***createScene()*** als eine der wichtigsten, da sie für das Erzeugen der gesamten Szene und aller benötigten Animationen zuständig ist. Dies geschieht selbstverständlich unter Zuhilfenahme weiterer Methoden und Klassen. Aufgrund ihrer Größe ist es allerdings nicht möglich den Quellcode an dieser Stelle anzugeben, weswegen deren Aufgaben lediglich in Textform aufgezeigt werden. Das entsprechende Listing befindet sich allerdings im Anhang ab Seite 125.
Zum Erzeugen von Szenen benötigt die Methode ***createScene()*** zunächst die vom Benutzer vorgegebenen Einstellungen sowie dessen hochgeladene Dateien. Diese sind über die Member-Variable `__user_data` erreichbar und werden daraufhin zum komfortableren Zugriff in diversen lokalen Variablen zwischengespeichert. Anhand dieser lässt sich hiernach eine Ebene für die Karte und die dazu passende Beleuchtung generieren, was mit Hilfe der Klassen ***Map*** und ***Lighting*** geschieht. Eine solche Ebene wird auf Basis der zugrundeliegenden Bilddatei erzeugt. Hierzu bestimmt die entsprechende Methode zunächst die kürzeste Seite der hochgeladenen Grafik und setzt diese mit einem Standardwert von 10 Blendereinheiten gleich. Daraufhin lässt sich über einen einfachen Dreisatz die Länge der anderen Seite in Blendereinheiten berechnen. Somit kann garantiert werden, dass die Ebene das gleiche Seitenverhältnis hat, wie die Karte des Benutzers. Anschließend wird der gerade erzeugten Ebene die entsprechende Grafik zugewiesen. Dies erfolgt auf Basis diverser Methoden der Blender API und der Klasse ***Materials*** aus dem Hauptmodul. Die Beleuchtung wird ebenfalls unter Verwendung der Blender API generiert, wobei zusätzlich die Seitenlängen der zuvor erzeugten Karte nötig sind. Somit ist es möglich je eine Lichtquelle an den Eckpunkten der Ebene zu platzieren und diese auf deren Mittelpunkt auszurichten. Weiterhin bestrahlt eine fünfte Lichtquelle die Szene aus Sicht der Kamera. Hierbei wurden in allen fünf Fällen sogenannte *Sun Lamps* verwendet, bei welchen das ausgestrahlte Licht keine Intensität in Bezug auf die zurückgelegte Distanz verliert und somit die Szene gleichmäßig beleuchtet. Zudem steht eine weitere Beleuchtungsmethode zur Verfügung, welche mit anderen Lichtquellen ar-

beitet. Dessen Verwendung kann über die Konfigurationsdatei angegeben werden. Der Status dieser Beleuchtung ist allerdings experimentell und somit lassen sich nicht für jede beliebige Karte optimale Ergebnisse garantieren.
Die nächste Aufgabe der Methode ***createScene()*** ist das Generieren der einzelnen Diagramme und deren Animationen. Hierbei wird zunächst zwischen den diversen Diagrammtypen unterschieden und daraufhin ein Objekt der entsprechenden Klasse erzeugt und ebenfalls einer Member-Variablen zugewiesen. Zu diesem Zeitpunkt existiert allerdings noch keines der Diagramme in Form eines Blender-Objekts, sondern es wurde lediglich ein Objekt der entsprechenden Diagrammklasse für den Generierungsprozess vorbereitet. Um die Generierung nun tatsächlich durchzuführen, muss zunächst der Zeitpunkt bestimmt werden, an dem die Animation starten soll. Dieser basiert auf den vom Benutzer angegebenen Werten für die Länge des Titels und der ersten Gesamtübersicht. Mit Hilfe der übrigen Einstellungen bezüglich der Animationszeiten lassen sich nun alle vorgegeben Diagramme generieren. Hierzu wird die Methode ***createDiagrams()*** des entsprechenden Diagrammtyps unter Angabe des zuvor bestimmten Startframes aufgerufen. Diese berechnet zunächst für alle darzustellenden Diagramme deren maximale Größe, um Überschneidungen mit nahe liegenden Blender-Objekten zu vermeiden. So kann es beispielsweise vorkommen, dass nicht alle Diagramme die gleiche Größe besitzen. Dieser Vorgang kann dem nachfolgenden Listing entnommen werden.

```
1  points = self._point_list.getPoints();
2
3  for p in points:
4      p.setMaxRadius(self._max_width);
5
6      for q in points:
7          if not (p == q):
8              radius = Utils.distance(p, q, self._map);
9
10             if (radius < p.getMaxRadius()):
11                 p.setMaxRadius(radius);
```

Hier wird zunächst in der ersten Zeile die Liste aller Diagrammpositionen in der Variablen `points` gespeichert. Diese ist nicht zu verwechseln mit einem Objekt der Klasse ***PointList***, welches nicht nur eine solche Liste, sondern auch zusätzliche Informationen kapselt. Daraufhin wird der maximale Radius jedes einzelnen Punkts auf einen bei der Erzeugung des Diagramm-Objekts übergebenen Wert gesetzt (Zeile 4). Auf Basis dieses Startwerts lässt sich nun überprüfen, ob es zu Konflikten mit anderen Diagrammen kommen kann. Ist die Distanz zweier Diagramme kleiner als der momentan gespeicherte maximale Radius, so wird er entsprechend aktualisiert (Zeile 10, 11). Auf diese Weise lassen sich nun alle Paare von Punkten abarbeiten.

Wurden alle Berechnungen erfolgreich durchgeführt, so lassen sich über eine weitere for-Schleife und die Methode ***createChart()*** die Diagramme an den einzelnen Positionen erzeugen. Da es sich hierbei um einen zeitlichen Ablauf aufgrund der Animationen handelt, wird zudem für jedes einzelne Diagramm dessen Start- und Endframe in einer Liste gespeichert. Auf Basis des letzteren und der Distanz zur nächsten Diagrammposition kann nun das Startframe des nachfolgenden Diagramms berechnet und der Methode ***createChart()*** übergeben werden. Die Liste aller Frames hingegen ist später beim Erzeugen der Kameraanimation äußerst wichtig, da sie Informationen enthält, zu welchen Zeitpunkten sich die Kamera bewegen soll bzw. wann sie still steht.
Die Methode ***createChart()*** erhält neben dem Startframe auch ein Objekt des Typs ***Point***. Dieses liefert alle Informationen, die benötigt werden, um das Diagramm zu erzeugen. Das bedeutet, dass hiermit sowohl dessen Position als auch der darzustellende Datensatz in Form eines ***Diagram***-Objektes bekannt sind. Wie oben beschrieben, lässt sich ebenfalls auf die maximale Größe eines Diagramms zugreifen. Anhand dieser Informationen können nun die einzelnen Teildiagramme samt Beschriftungen und Animationen erzeugt werden. Das nachfolgende Listing stellt dies in einer gekürzten Form dar. Hier sollen lediglich die Aufrufe der diversen Methoden aufgezeigt und erklärt werden.

```
1  for i in range( 0, len( diagram.getRow(0).getValues() ) ):
2      cur_value = 0.0;
3      cur_offset = 0.0;
4
5      descr_obj = self.columnDescr(diagram.getColumnDescr(i), cur_frame, ...);
6      cur_frame += self._anim_frames + self._frames_per_element;
7
8      for j in range( 0, len( diagram.getRows() ) ):
9          row = diagram.getRow(j);
10         cur_value = row.getValue(i)/sum_per_column[i];
11
12         if (cur_value > 0.0):
13             segment_obj = self.segment(cur_value, cur_offset, cur_frame, ...);
14
15             if (self._show_values):
16                 value_obj = self.valueText(row.getValue(i), cur_frame, ...);
17
18             cur_frame += self._anim_frames + self._frames_per_element;
19             cur_offset += cur_value;
20
21     cur_frame += self._frames_per_chart;
22     posZ += pie_distance_modifier * self._pie_height;
23
24 eFrame = cur_frame + self._frames_per_diagram;
```

Wie bereits in Abschnitt 3.4 erwähnt, enthält jeweils eine Spalte des zugrundeliegenden Datensatzes die Informationen für ein Teildiagramm. Aus diesem Grund läuft die Hauptschleife in Zeile 1 die einzelnen Spalten des in der Variablen `diagram` gespeicherten Datensatzes ab. Hierzu wird zunächst für jedes Teildiagramm dessen Beschriftung erzeugt, was mit Hilfe der Methode ***columnDescr()*** geschieht. Diese erhält unter anderem den jeweiligen Spaltenkopf als Parameter, für welchen daraufhin ein entsprechendes Blender-Objekt erzeugt und animiert wird. Zudem lässt sich hier ebenso ein Startframe für die jeweilige Animation übergeben. Die übrigen Parameter sollen hier nicht weiter betrachtet werden, dienen aber hauptsächlich zu Positionierung des Beschreibungstextes. Die zurückgelieferte Referenz auf das erzeugte Blender-Objekt wird zunächst zwischengespeichert und in der ungekürzten Version dieser Methode einer Liste aller Beschreibungstexte hinzugefügt. In Zeile 6 wird daraufhin der Zähler für Frames auf Basis der Benutzereinstellungen aktualisiert, da hierüber der Startzeitpunkt für die nachfolgende Animation definiert ist. Anfangs wurde dieser mit dem übergebenen Startframe initialisiert. Anschließend können nun mit Hilfe einer weiteren for-Schleife die einzelnen Zeilen des Datensatzes für die jeweils fest vorgegebene Spalte durchlaufen werden. Hierzu ergibt sich der aktuell darzustellende Wert aus dem Quotienten des gelesenen und der Summe aller Werte einer Spalte (Zeile 10). Diese Summen wurden bereits im Vorfeld in der Liste `sum_per_column` gespeichert. Ist der erhaltene Wert größer als Null, so wird ein entsprechendes Kreissegment unter Betrachtung der vorherigen erzeugt (Zeile 13) und, falls vom Benutzer erwünscht, mit dem im Datensatz enthaltenen Wert beschriftet (Zeile 15, 16). Hierfür steht zudem die Variable `cur_offset` bereit, in welcher der jeweilige Versatz innerhalb des Kreises gespeichert werden soll. Auch in diesem Fall handelt es sich um animierte Blender-Objekte, weswegen jeweils ein Startframe anzugeben ist. Analog zur ersten Methode erhalten auch ***segment()*** und ***valueText()*** weitere Parameter für Position und Größe der zu erzeugenden Blender-Objekte. Zum Abschluss der inneren for-Schleife erhöht die Methode sowohl die Variable `cur_frame` als auch der Versatz für das nächste Kreissegment (Zeile 18, 19). Gleiches gilt für das Ende der äußeren Schleife, wobei dort statt Erhöhen des Versatzes der entsprechende Z-Wert aktualisiert wird (Zeile 22). Dieser lässt sich für die Positionierung der einzelnen Teildiagramme übereinander verwenden und berechnet sich aus dem Produkt der Höhe eines solchen und einem speziellen Modifikator, welcher einen Abstand zwischen jeweils zwei Teildiagrammen erzeugt. Nach Beenden der Hauptschleife wird abschließend das letzte Frame der aktuellen Animation berechnet und am Schluss der Methode zurückgegeben. Dieses lässt sich daraufhin für die nächste Animation verwenden.
Wichtig ist noch zu erwähnen, dass sowohl speziell Methoden zur Erzeugung der jeweiligen Blender-Objekte als auch solche zum Generieren der Animationen existieren. Während erstere einen Postfix mit der Bezeichnung *Geometry* besitzen, erhalten letztere den Anhang *Ipo*. Auf diese Methoden soll allerdings nicht näher eingegangen werden, da sie abhängig vom Diagrammtyp unterschiedlich implementiert sind und teilweise Hilfsmethoden verwenden.
Konnten sämtliche Diagramme generiert werden, so kehrt das Python-Skript in die Methode

createScene() der Hauptklasse zurück. Auf Basis der Liste aller Start- und Endframes, welche beim vorherigen Schritt gespeichert wurde, lässt sich nun die Kamera samt Animation erzeugen. Zudem wird ein sogenanntes Tracking-Objekt erstellt, welches die Ausrichtung der Kamera steuert. Das heißt, dass die Rotation der Kamera vollständig an die Position dieses Objektes gebunden ist. Deren eigene Position muss allerdings entsprechend animiert werden. Hierzu lassen sich beim Erzeugen eines Objekts der Klasse ***CameraAnimation*** zusätzlich zu den gespeicherten Frames ebenso die verschiedenen Positionen angeben. Darunter fallen auch die Koordinaten der Kamera bei der Gesamtübersicht oder dem Schlussdiagramm, falls ein solches erwünscht ist. Mit der Methode ***createAnimation()*** kann daraufhin die Kamera samt Tracking-Objekt und Animationen erzeugt werden.
Wurde auch dies erfolgreich durchgeführt, so generiert das Python-Skript mit Hilfe der Klasse ***Title*** den Titelbildschirm des Präsentationsvideos. Neben einer Hintergrundebene in der entsprechenden Farbe, lassen sich ebenso die unterschiedlichen Text-Objekte auf Basis der Benutzereingaben erstellen. Auch in diesem Fall wird zunächst eine Instanz initialisiert und erst dann die jeweiligen Blender-Objekte und deren Animationen erzeugt. Hierfür lassen sich zusätzlich die entsprechenden Animationszeiten und die globale Kameraposition verwenden. Ähnlich wie bei den Diagrammen, werden auch deren Legenden mit Hilfe einer einzigen Klasse erzeugt. Hierzu benötigt diese, neben den entsprechenden Beschreibungstexten, sowohl die einzelnen Diagrammpositionen als auch die Liste aller Start- und Endframes. Letzteres ist vor allem für das Ein- und Ausblenden der jeweiligen Legende wichtig. Zudem werden bei der Initialisierung eines neuen Objekts der Klasse ***Key*** weitere Parameter speziell für die Positionierung der später zu generierenden Blender-Objekte verlangt. Über die Methode ***createKeys()*** lässt sich das Erzeugen der Legenden schließlich durchführen.
Für den Fall, dass der Benutzer sich für das Anzeigen eines Schlussdiagramms entschieden hat, wird an dieser Stelle ein solches erstellt. Auch hierfür stellt das Hauptmodul die entsprechende Klasse bereit. Diese steht allerdings eng mit sämtlichen Klassen zur Generierung zweidimensionaler Diagramme in Verbindung. Falls nun ein Schlussdiagramm erzeugt werden soll, greift die Methode ***createChart()*** der Klasse ***SummaryChart*** auf Objekte der zweidimensionalen Diagrammklassen zurück. Dort lassen sich auf ähnliche Weise, wie bei den dreidimensionalen Gegenstücken, die benötigten Blender-Objekte erzeugen und animieren. Die jeweilige Positionierung wird allerdings in der Klasse ***SummaryChart*** berechnet und der entsprechenden Methode übergeben. Dies wird aufgrund von relativen Positionen in Abhängigkeit von der Hintergrundebene und der ebenfalls zu erzeugenden Legende getan. Somit lassen sich über eine globale Berechnung die gesamten Elemente des Schlussdiagramms auf eine einfache Art und Weise positionieren. Alle oben genannten Schritte setzen selbstverständlich den entsprechenden Datensatz am Ende der CSV-Datei voraus. Dieser wurde beim Einlesen über ein Objekt der Klasse ***Summary*** verfügbar gemacht, welches lediglich für die Speicherung der Daten zuständig und nicht mit der Klasse ***SummaryChart*** zu verwechseln ist. Wurde im Gegensatz hierzu eine Bilddatei hochgeladen, so erzeugt das Python-Skript

nur die Hintergrundebene und weist dieser die entsprechende Grafik zu. In beiden Fällen wird das Schlussdiagramm durch Einblenden animiert, welches sich mit Hilfe der Klasse ***Ipo*** und der dort verfügbaren statischen Methode ***fadingIpo()*** erreichen lässt.
Hiernach kehrt das Skript ein letztes Mal in die Methode ***createScene()*** der Hauptklasse zurück, um dort die Umgebungsfarbe der Szene auf Basis der Benutzerangaben einzustellen. Ist dies getan, so wurde die Szene erfolgreich generiert und lässt sich daraufhin rendern.
Dieser Vorgang wird nachfolgend anhand der Methode ***renderPreview()*** beispielhaft dargestellt. Das Rendern des finalen Präsentationsvideos verläuft allerdings auf eine ähnliche Art und Weise. Auch hier wurden der Methodenkopf und die Fehlerbehandlung entfernt.

```
1  render_settings = self.__user_data.getRenderSettings();
2
3  context = Scene.GetCurrent().getRenderingContext();
4  self.renderSettings(context);
5
6  for i in range( 0, len(self.__preview_frames) ):
7      context.imageSizeX( render_settings.getVideoWidth() );
8      context.imageSizeY( render_settings.getVideoHeight() );
9      context.extensions = True;
10     context.renderPath = self.__user_data.getUserDir();
11     context.imageType = Scene.Render.PNG
12     context.cFrame = int(self.__preview_frames[i]);
13
14     context.render();
15     context.saveRenderedImage( "frame0" + str(i) );
```

Zu Beginn des Listings speichert das Python-Skript zunächst eine Referenz auf die benötigten Einstellungen bezüglich des Rendervorgangs. Dies soll lediglich die Handhabung vereinfachen und unübersichtliche Zeilen vermeiden. Daraufhin wird über die Methode ***getRendering-Context()*** der aktuellen Szene ein Objekt des Typs ***RenderData*** in der Variablen `context` gespeichert. Hierbei handelt es sich um ein Klasse der Blender API, welche für den gesamten Renderprozess zuständig ist und die entsprechende Funktionalität bereitstellt. Auf Basis eines solchen Objekts kann nun in Zeile 4 die Methode ***renderSettings()*** zum Setzen verschiedener Grundeinstellungen verwendet werden. Diese erhält das Skript entweder aus der Konfigurationsdatei oder direkt aus dem Quellcode. So wird beispielsweise die Kantenglättung von außerhalb gesteuert, während Radiosity oder Panorama-Renderings grundsätzlich abgeschaltet sind.
Anschließend lassen sich die einzelnen Bilder mit Hilfe einer for-Schleife rendern. Diese wurden beim Erzeugen der Szene in einer Liste gespeichert, worauf nun zugegriffen wird. Hierfür legt das Python-Skript zunächst in den Zeilen 7 und 8 die Größe des auszugebenden Bildes fest. Dies geschieht selbstverständlich auf Grundlage der jeweiligen Benutzereinstellungen,

welche nach dem Einlesen in einem Objekt der Klasse ***RenderSettings*** gespeichert wurden. Die Zeilen 9 und 10 spielen im Gegensatz dazu eine wichtige Rolle bei der Ausgabe der Bilddateien. Während ersteres die automatische Dateinamenserweiterung einschaltet, wird mit letzterem das Zielverzeichnis der Bilddateien angegeben. Auch dieses wurde beim Einlesen der XML-Datei zwischengespeichert. Schlussendlich wird der Dateityp und das zu rendernde Frame gesetzt, das gewählte Bild gerendert und daraufhin im entsprechenden Verzeichnis gespeichert. Zum Rendern einer Animation lassen sich an Stelle eines einzelnen Bildes jeweils ein Start- und ein Endframe angeben. Zudem lässt sich hierfür die Methode ***renderAnim()*** verwenden, welche ein Video oder diverse Einzelbilder automatisch abspeichert.
Wurde auch der Rendervorgang erfolgreich durchgeführt, so werden, wie bereits oben erwähnt, alle wichtigen Daten und ein Beschreibungstext per E-Mail an den Benutzer gesendet. Diese Beschreibungstexte befinden sich in unterschiedlichen TXT-Dateien und lassen sich je nach Bedarf laden und der E-Mail hinzufügen. Das Versenden einer solchen E-Mail mit dem Modul ***SimpleMail*** wurde aber bereits in Abschnitt 2.3 erklärt.
Nach einem weiteren Datenbankzugriff beendet das Skript anschließend Blender, womit die Aufgaben des Backends abgeschlossen sind. Auch diese Vorgänge wurden bereits zu Beginn des Abschnitts beschrieben.

4.4. Der JobListener

Dieser Abschnitt beschäftigt sich mit dem JobListener des ChartFlight Services und dessen Implementierung. Zunächst werden in einer kurzen Übersicht die einzelnen Bestandteile dieses Java-Programms dargestellt und deren Aufgaben erklärt. Diese beinhaltet ebenso eine Vorstellung der diversen Startmodi. Daraufhin wird detaillierter auf verschiedene Bereiche der Implementierung eingegangen und anhand unterschiedlicher Codebeispiele dessen Funktionsweise erläutert. Abschließend stellt ein separater Abschnitt zusätzliche Funktionen unter Linux dar. Hierfür existiert zum aktuellen Zeitpunkt keine spezielle Implementierung unter Windows. Da dieser Service aber hauptsächlich unter Linux betrieben werden soll, ist es momentan nicht nötigt, die entsprechende Funktionalität für Windows zu ergänzen.

4.4.1. Überblick

Im Gegensatz zum Backend befindet sich der Quellcode des JobListeners in einer einzigen Klasse, welche den gleichen Namen trägt. Diese kann allerdings in die drei nachfolgenden Bereiche eingeteilt werden:

Bei der Initialisierung werden alle wichtigen Einstellungen geladen und in den entsprechenden Member-Variablen gespeichert. Hierunter befinden sich zunächst diverse Pfade zu den anderen Komponenten des ChartFlight Services, wie beispielsweise das Verzeichnis, in welchem

Initialisierung	Laden der Konfigurationsdatei und des Datenbanktreibers
Bereinigung	Entfernen diverser Aufträge aus Dateisystem und Datenbank
Bearbeitung	Durchführen diverser Aufträge mit Blender

Tabelle 4.7.: Aufgaben des JobListeners

sich die Startdatei von Blender befindet. Weiterhin beinhaltet die Konfigurationsdatei eine ganze Reihe von Zeitangaben, mit deren Hilfe sich die Wartedauer verschiedener Vorgänge festlegen lässt. Auch diese werden anhand von Member-Variablen zwischengespeichert. Eine detailliertere Vorstellung der einzelnen Einstellungsmöglichkeiten befindet sich allerdings in Abschnitt 4.7.
Eine weitere wichtige Aufgabe bei der Initialisierung ist das Bestimmen des verwendeten Betriebssystems. Dies wird vor allem beim Starten von Blender mit Hilfe der Konsole benötigt und setzt bei jedem Betriebssystem einen unterschiedlichen Befehl voraus. Aktuell unterstützt der JobListener die Systeme *Windows* und *Linux*. Die letzte Aufgabe besteht darin, den Treiber für die SQLite Datenbank zu laden, um so den Zugriff auf eine solche zu ermöglichen.
Der Bereich der Bereinigung beschäftigt sich mit dem Entfernen abgebrochener oder abgelaufener Aufträge aus dem Dateisystem und der Datenbank. Hierzu steht, speziell für den zweiten Fall, eine Methode zur Benachrichtigung der Benutzers per E-Mail bereit. Zudem kann der JobListener Verzeichnisse und Dateien mit den entsprechenden Rechten löschen, was für abgebrochene Aufträge ohne eine solche Nachfrage geschieht. In beiden Fällen verwendet das Programm die zuvor geladenen Zeitangaben.
Der dritte und letzte Bereich beinhaltet Methoden zum Bearbeiten eines Auftrags. Diese ermöglichen den Zugriff auf die Datenbank und das Herausfiltern einer solchen Zeile. Zudem ist es von dort aus möglich, Blender auf Basis des zuvor bestimmten Betriebssystems zu starten und gegebenenfalls auch auf eventuelle Abstürze zu überwachen.

Der JobListener kann nun auf mehrere verschiedene Arten gestartet werden. Somit ist er entweder für den Dauereinsatz, das Bearbeiten eines einzigen Auftrags oder die Bereinigung des Dateisystems und der Datenbank zuständig. Tabelle 4.8 listet alle verfügbaren Modi mit einem kurzen Beschreibungstext auf.

check	Suchen und Bearbeiten eines Auftrags aus der Datenbank
clean	Bereinigung des Dateisystems und der Datenbank
both	Kombination der ersten beiden Modi
multi	Ausführen der ersten beiden Modi in gegebenen Zeitintervallen
debug	Kombination der ersten beiden Modi mit Fehlerausgabe über die Konsole

Tabelle 4.8.: Startmodi des JobListeners

Wie Tabelle 4.8 zu entnehmen ist, kann der erste Modus dazu verwendet werden, die Datenbank nach dem nächsten zu bearbeitenden Auftrag zu durchsuchen. Dieser Vorgang wird lediglich ein einziges Mal durchgeführt. Hierbei kann es durchaus vorkommen, dass sich kein

entsprechender Eintrag in der Datenbank finden lässt und das Programm seine Arbeit sofort wieder beendet. Wurde im Gegensatz dazu ein Auftrag gefunden, so wird dieser mit Hilfe von Blender bearbeitet und daraufhin der JobListener geschlossen.
Ein ähnliches Verhalten lässt sich beim zweiten Modus feststellen. Dort wird allerdings kein Auftrag bearbeitet, sondern lediglich das Dateisystem und die Datenbank bereinigt. Wurden hierbei keine zu löschenden Daten bzw. Datenbankeinträge gefunden, so beendet der JobListener auch in diesem Fall seine Arbeit sofort. Anderenfalls führt er zuerst die entsprechenden Bereinigungsvorgänge durch.
Der Modus **both** kombiniert nun die beiden ersten, wobei zunächst das Dateisystem und die Datenbank bereinigt werden und erst daraufhin der nächste Auftrag bearbeitet wird. Auch hier arbeitet der JobListener lediglich einmal die beiden genannten Aufgabenbereiche ab und beendet danach seine Ausführung.
Im Gegensatz dazu kann der JobListener unter Verwendung des Modus **multi** beliebig lange als Hintergrundprozess im System ausgeführt werden. Hiermit ist es nun möglich, Datenbank und Dateisystem in vorgegebenen Zeitabständen regelmäßig zu bereinigen oder nach einem neuen Auftrag zu durchsuchen und ihn entsprechend zu bearbeiten. Zur Anpassung dieser Intervalle lässt sich die Konfigurationsdatei verwenden. Zu erwähnen ist hierbei allerdings, dass der JobListener lediglich durch einen Eingriff von außen oder das Auftreten eines Fehlers beendet werden kann.
Der letzte Modus arbeitet ebenfalls auf einer Kombination der ersten beiden. Dieser unterscheidet sich von dem Modus **both** in nur einem einzigen Punkt. Beim Starten von Blender über die Konsole wartet der JobListener bis alle Operationen beendet wurden und sammelt deren Ausgaben. So ist es beispielsweise möglich, Fehler, welche bei der Ausführung des Python-Skripts auftraten, an den JobListener weiterzureichen. Diese lassen sich daraufhin in der Konsole ausgeben und entsprechend begutachten. Bei allen anderen Modi gehen solche Ausgaben verloren.
Abschließend soll die Verwendung des JobListeners am Beispiel der Modi **check** und **multi** gezeigt werden. Hierfür lässt sich mit Hilfe des zusätzlichen Komandozeilenparameters ***-mode*** der jeweiligen Modus einleiten. Gleich darauf wird dieser mit dem entsprechenden Namen angegeben.

```
java -jar JobListener.jar -mode check
java -jar JobListener.jar -mode multi &
```

Damit der JobListener beim Verwenden des Modus **multi** auch tatsächlich im Hintergrund startet, ist bei linuxartigen Systemen ein zusätzliches Et-Zeichen (&) am Ende der Zeile vonnöten. Werden die Paramater nicht in der oben genannten Reihenfolge angegeben, so beendet das Programm mit den entsprechenden Hinweisen zur Handhabung und einer Auflistung der unterstützten Modi.

4.4.2. Implementierung

Dieser Abschnitt soll nun einen detaillierteren Einblick in die Implementierung des JobListeners geben. Hier sollen zunächst Methoden zur Initialisierung und daraufhin solche zum Bereinigen der Datenbank und des Dateisystems vorgestellt werden. Zu guter Letzt zeigt dieser Abschnitt, wie das Bearbeiten eines Auftrags im JobListener abläuft. Zu erwähnen ist auch hier, dass Auszüge aus dem Quellcode nicht immer vollständig sind und auf die entsprechende Fehlerbehandlung verzichtet wurde.
Wie allgemein üblich, läuft die Initialisierung innerhalb des Konstruktors ab. Hierbei werden zunächst alle Member-Variablen mit Standardwerten versehen, auf welche sich im Notfall zurückgreifen lässt. So kann es beispielsweise vorkommen, dass im nächsten Schritt, dem Laden der Konfigurationsdatei, eine oder mehrere Variablen nicht initialisiert werden können. Für den Zugriff auf eine solche Konfigurationsdatei steht auch in Java eine bereits vorgefertigte Klasse zur Verfügung. Dieser Vorgang soll nachfolgend am Einlesen des Pfades zur Datenbank und deren Namen beispielhaft dargestellt werden.

```
1 File file = new File("./config.properties");
2 FileInputStream fis = new FileInputStream( file.getCanonicalPath() );
3
4 Properties config_file = new Properties();
5 config_file.load(fis);
6
7 this.users_dir = config_file.getProperty("UsersDir");
8 this.user_queue = config_file.getProperty("UserQueue");
```

Zunächst wird die Konfigurationsdatei mit der Bezeichnung `config.properties` unter Verwendung eines Objekts der Klasse ***FileInputStream*** für das Lesen vorbereitet (Zeile 1, 2). Diese befindet sich hierbei im gleichen Verzeichnis, wie der JobListener selbst. Über ein Objekt des Typs ***Properties*** und der dort verfügbaren Methode ***load()*** kann nun die vorbereitete Konfigurationsdatei geladen werden. Hiernach lässt sich mit der Methode ***getProperty()*** und unter Angabe der gewünschten Eigenschaft, deren Wert erhalten. Diese werden anschließend in den entsprechenden Member-Variablen gespeichert.
Wurde die Konfigurationsdatei erfolgreich geladen, so lässt sich mit Hilfe des Konstrukts `System.getProperty("os.name")` das zugrundeliegende Betriebssystem bestimmen. Auf Basis dieser Information können daraufhin verschiedene Member-Variablen gesetzt werden. Hierbei handelt es sich beispielsweise um den Namen der verwendeten Konsole oder den Befehl zum Starten von Blender, welche auf den beiden Betriebssystemen unterschiedlich aussehen.
Zu guter Letzt wird über die Zeile `Class.forName("org.sqlite.JDBC")` der Treiber für die SQLite Datenbank geladen. War dies nicht erfolgreich, so wird der JobListener mit einer

entsprechenden Fehlermeldung beendet.
Nach einer erfolgreichen Initialisierung steht nun ein Objekt der Klasse ***JobListener*** zur Verfügung und kann zu den bereits erwähnten Aufgaben verwendet werden. In diesem Zusammenhang spielt die Methode ***listen()*** eine wichtige Rolle. Hierüber lässt sich der gesamte Überwachungsvorgang starten, unabhängig davon, welcher Modus gewählt wurde. Die Hauptschleife dieser Methode soll im nachfolgenden Listing ein wenig genauer betrachtet werden.

```
1  float expired_time = 0.0f;
2
3  do {
4      this.time_to_sleep = this.checking_interval;
5
6      if ((this.clean && !this.continuous) ||
7          (this.clean && expired_time >= this.clean_up_interval)) {
8          this.cleanUp();
9          expired_time = 0.0f;
10     }
11
12     if (this.check) {
13         this.jobCheck();
14     }
15
16     if (this.continuous) {
17         Thread.sleep( (long) (this.time_to_sleep * 60000.0) );
18         expired_time += this.checking_interval;
19     }
20 }
21 while (this.continuous);
```

Da es sich bei der oben dargestellten Schleife um ein `do-while`-Konstrukt handelt, wird diese unabhängig vom gewählten Modus mindestens einmal ausgeführt. Soll sie allerdings beliebig lange durchlaufen werden, so muss die boolesche Member-Variable `continuous` den Wert ***true*** enthalten. Dies entspricht dem Modus **multi**, wobei dort zudem die Variablen `check` und `clean` den gleichen Wert ausweisen.
Zu Beginn der Schleife wird zunächst in Zeile 4 das aus der Konfigurationsdatei eingelesene Zeitintervall in einer weiteren Member-Variablen gespeichert. Somit kann deren Wert nun beliebig geändert werden, ohne den ursprünglichen zu verlieren. Dies spielt sowohl in Zeile 18 als auch in der Methode ***jobCheck()*** eine wichtige Rolle. Für den Fall, dass der Modus **multi** gewählt wurde, leitet ersteres am Ende jedes Schleifendurchlaufs eine inaktive Phase ein. Hierbei wartet das Programm die in der Member-Variablen `time_to_sleep` gespeicherte Zeit, bevor es seine Arbeit wieder aufnimmt. In der Methode ***jobCheck()*** ist dies allerdings abhängig von den vorhandenen Einträgen in der Datenbank. Somit kann es auch

hier zu einer inaktiven Phase kommen. Insgesamt haben diese aber immer die in der Variablen `checking_interval` angegebene Länge. So sind beispielsweise zwei inaktive Phasen mit jeweils halber oder eine einzige mit voller Länge möglich. Schlussendlich bedeutet dies, dass die Datenbank immer in regelmäßigen Abständen überprüft wird.
Welche Aufgaben der JobListener nun durchführt, ist von den Member-Variablen `check` und `clean` abhängig. So wird zunächst in den Zeilen 6 und 7 überprüft, ob die Methode ***cleanUp()*** Datenbank und Dateisystem bereinigen soll. Hierbei ist zwischen einem einmaligen Durchlauf der Schleife und dem ständigen Betrieb mit Hilfe der Variablen `continuous` zu unterscheiden. Beide Varianten sind selbstverständlich zunächst von dem in `clean` gespeicherten Wert abhängig. Allerdings arbeitet die zweite Variante auf Basis eines zusätzlichen Intervalls, welches sich ebenfalls über die Konfigurationsdatei angeben lässt. Hierzu wird in jedem Schleifendurchlauf die vergangene Zeit in einer lokalen Variablen zwischengespeichert (Zeile 18). Falls diese die Größe des Intervalls erreicht, ruft das Programm die Methode ***cleanUp()*** auf und setzt die lokale Variable auf 0.0 (Zeile 8, 9). Eine solche Vorgehensweise erfordert allerdings, dass `clean_up_interval` ein Vielfaches des in `checking_interval` gespeicherten Wertes ist. Anschließend lässt sich in Zeile 12 mit Hilfe der Member-Variablen `check` überprüfen, ob die Datenbank auf zu bearbeitende Aufträge durchsucht werden soll. Ist dies der Fall, so startet das Programm mit ***jobCheck()*** den Suchvorgang. Der Aufbau der beiden zuletzt vorgestellten Methoden soll nun im Nachfolgenden ein wenig detaillierter erklärt werden.
Betrachten wir zunächst die Methode ***cleanUp()***, welche für die Bereinigung der Datenbank und des Dateisystems zuständig ist. Dieser Vorgang kann in drei Teilbereiche gegliedert werden, wofür weitere Methoden zu Verfügung stehen. Der erste Bereich beschäftigt sich mit abgebrochen Aufträgen, also solchen, die noch nicht in die Datenbank aufgenommen wurden. Diese entstehen dann, wenn ein Benutzer zwar Dateien hochlädt, den Auftrag danach allerdings nicht zu Ende führt, sondern dessen Bearbeitung vorzeitig abbricht. Hierfür steht die Methode ***cleanUpCancelledJobs()*** zur Verfügung, welche von ***cleanUp()*** aufgerufen wird und nach folgendem Prinzip vorgeht. Zunächst betrachtet sie alle Benutzerverzeichnisse und überprüft, ob ein entsprechender Eintrag in der Datenbank registriert wurde. Ist dies der Fall, so übergeht sie den jeweiligen Ordner. Wurde allerdings keine Zeile mit der gleichen Auftragsnummer gefunden, so könnte es sich dabei um einen abgebrochenen Auftrag handeln. Die endgültige Entscheidung hierüber lässt sich mit Hilfe des Zeitintervalls `del_cancelled_job` feststellen, welches sich ebenfalls in der Konfigurationsdatei befindet. Dazu wird das in der Bezeichnung des jeweiligen Verzeichnisse kodierte Datum mit dem aktuellen abgeglichen. Ist der Unterschied größer als die oben genannte Member-Variable, so löscht das Programm den entsprechenden Ordner samt Inhalt.
Die beiden übrigen Bereiche beschäftigen sich lediglich mit registrierten Aufträgen. Hierbei muss allerdings zwischen solchen, die sich noch in Bearbeitung befinden und solchen, bei denen dieser Vorgang bereits abgeschlossen ist, unterschieden werden. Ersteres bezieht

sich speziell auf den Status ***preview*** eines Auftrags der Haupttabelle der Datenbank. Da an dieser Stelle ein Eingriff des Benutzers zur weiteren Bearbeitung nötig ist, kann es durchaus vorkommen, dass ein solcher Auftrag in Vergessenheit gerät. Hierzu steht die Methode ***cleanUpPreviewJobs()*** zur Verfügung, welche ebenfalls von ***cleanUp()*** aufgerufen wird. Auch diese greift auf Zeitintervalle aus der Konfigurationsdatei zurück. Zunächst betrachtet sie aber für alle Aufträge mit den Status ***preview*** deren Werte im Feld **date**. Diese werden daraufhin, neben dem aktuellen Datum, für einen Zeitvergleich herangezogen. Unter Verwendung der Member-Variablen `del_preview_job` und `user_notification` können nun drei unterschiedliche Situationen eintreten. Zunächst wird überprüft, ob der errechnete Zeitraum größer ist als der in `del_preview_job` enthaltene. Ist dies der Fall, so löscht das Programm den entsprechenden Benutzerordner und den zugehörigen Datenbankeintrag. Anderenfalls lässt sich mit einer zweiten Abfrage überprüfen, ob das Zeitintervall größer als die Differenz obiger Variablen ist. Dies würde bedeuten, dass der JobListener den Auftrag zwar nicht entfernt, allerdings den jeweiligen Benutzer über dessen Ablauf informiert. Hierzu wird das in Abschnitt 2.3 vorgestellte JavaMail verwendet. Tritt auch eine solche Situation nicht ein, so betrachtet das Programm den nächsten Auftrag.
Auf eine ähnliche Weise arbeitet auch die Methode ***cleanUpDoneJobs()***. Hierbei werden allerdings nur die Aufträge mit dem Status ***done*** begutachtet und entfernt. Als Zeitintervalle dienen die Variablen `del_done_job` und `user_notification`, welche über die Konfigurationsdatei eingelesen werden. Bei letzterer handelt es sich um dieselbe Variable wie oben.
Als nächstes soll nun die Methode ***jobCheck()*** betrachtet werden, welche die Hauptarbeit des JobListeners übernimmt. Hierbei durchsucht das Programm die Datenbank nach zu bearbeitenden Aufträgen und startet sie gegebenenfalls. Dieser Vorgang geschieht in drei Schritten. Zunächst werden aus der Datenbank alle Aufträge herausgefiltert, welche sich entweder im Status ***preview***, ***error*** oder ***done*** befinden. Alle Übrigen speichert das Programm in einem Objekt der Klasse ***HashMap***, das als Schlüssel den jeweiligen Status und als Wert die Anzahl der Aufträge dieser Sorte entgegennimmt. So kann daraufhin recht schnell festgestellt werden, ob ein zu bearbeitender Auftrag vorhanden ist. Weiterhin lässt sich auf diese Weise ebenso einfach überprüfen, ob sich bereits ein Auftrag in Bearbeitung befindet. Mit Hilfe der HashMap werden somit weitere unnötige Datenbankzugriffe unterbunden. Wurden nun Aufträge in der Datenbank gefunden, so teilt sich deren Bearbeitung in zwei Teilbereiche auf. Zunächst stellt das Programm unter Betrachtung der HashMap fest, ob gerade für einen Auftrag Vorschaubilder erstellt werden. Diese Information lässt sich über einen Eintrag mit dem Schlüssel ***rendering_preview*** und einem dort gespeicherten Wert von 1 erhalten. Konnte ein solcher Eintrag gefunden werden, so lässt sich unter Linux überprüfen, ob sich Blender tatsächlich in der Ausführung befindet. Da dieser Vorgang nur unter Linux verfügbar ist, soll er in Abschnitt 4.4.3 detaillierter beschrieben werden. Existiert kein solcher Eintrag in der HashMap, so wird daraufhin überprüft, ob ein Auftrag für das Rendern von Vorschaubildern bereitsteht. Hierzu lässt sich ein weiteres Mal die HashMap heranziehen, um nach Schlüsseln

mit dem Status ***rejected*** oder ***waiting*** zu suchen. Ersteres wird dabei bevorzugt behandelt. Konnte einer der beiden Einträge gefunden werden, so greift der JobListener ein zweites Mal auf die Datenbank zu und manipuliert diese, wie bereits in Abschnitt 4.2 beschrieben. Daraufhin startet er mit Hilfe der Methode ***startJob()*** den gefundenen Auftrag zum Rendern der Vorschaubilder. Diese soll allerdings erst am Ende des Abschnitts genauer betrachtet werden, da sie auch für den dritten Schritt wichtig ist.
Hierin betrachtet der JobListener die Aufträge, welche für den tatsächlichen Rendervorgang, also das Erzeugen des finalen Präsentationsvideos, bereitstehen und startet gegebenenfalls eine weitere Instanz von Blender. Zuvor versetzt sich der JobListener allerdings in eine inaktive Phase. Diese nimmt die Hälfte der durch `time_to_sleep` angegebenen Wartezeit in Anspruch. Die somit erhaltene zeitliche Verzögerung hat nun den Vorteil, dass zwei unterschiedliche Instanzen von Blender nicht unmittelbar nacheinander gestartet werden und nahezu zeitgleich auf die Datenbank zugreifen. Hierdurch lassen sich unnötige Programmabbrüche vermeiden, welche durch einen schreibenden Zugriff auf die Datenbank entstehen könnten. Somit kann man diese zeitliche Verzögerung als einen zusätzlichen Sicherheitsmechanismus betrachten. Mit einer solchen Vorgehensweise vergeht zwischen dem Starten zweier unterschiedlicher Blender Instanzen also mindestens die Hälfte des über die Konfigurationsdatei vorgegebenen Zeitintervalls. Aus diesem Grund sollte ein solches Intervall nicht zu kurz gewählt werden.
Nach der inaktiven Phase widmet sich das Programm erneut dem zuvor erzeugten Objekt der Klasse ***HashMap***. Hierin wird nun nach einem Schlüssel mit der Bezeichnung ***rendering*** gesucht und überprüft, ob dessen Wert 1 beträgt. Ist dies der Fall, so befindet sich bereits ein Auftrag in Bearbeitung. Auch hier lässt sich unter Linux Systemen testen, ob Blender sich tatsächlich in Ausführung befindet. Wurde im Gegensatz dazu kein entsprechender Schlüssel gefunden, so ist momentan auch keine Blender Instanz mit dem Erzeugen eines Präsentationsvideos beschäftigt. Somit lässt sich erneut mit Hilfe der HashMap überprüfen, ob ein solcher Auftrag bereitsteht. Trifft dies zu, so kontaktiert der JobListener die Datenbank und manipuliert sie wie in Abschnitt 4.2 beschrieben. Hierdurch erhält die anschließend gestartete Blender Instanz die Möglichkeit, den gefundenen Auftrag zu lokalisieren und auf dessen Basis das finale Präsentationsvideo zu erstellen. Im Falle einer gesetzten ***debug*** Variablen, was bei gleichnamigem Modus geschieht, stoppt der JobListener an dieser Stelle seine Ausführung und bearbeitet unterschiedliche Ausgaben von Blender. Bei normalem Betrieb wartet das Progamm allerdings nicht auf die Fertigstellung des Rendervorgangs, sondern kehrt aus der Methode ***jobCheck()*** zurück. Daraufhin befindet sich das Programm erneut in der Methode ***listen()*** und kann dort mit den weiteren Operationen fortfahren.
Wie den obigen Beschreibungen zu entnehmen, spielt die Methode ***startJob()*** beim Ausführen gefundener Aufträge eine wichtige Rolle. Aus diesem Grund soll sie nachfolgend detaillierter betrachtet werden. Auch hier wird der Methodenkopf sowie die Fehlerbehandlung und das Hinzufügen von Einträgen zur Protokolldatei aus Platzgründen nicht dargestellt.

```
1  ProcessBuilder builder = new ProcessBuilder(this.cmd_line,
2                                              this.arg,
3                                              this.command);
4  builder.directory( new File(this.blender_path) );
5  Process p = builder.start();
6
7  if (this.debug)
8  {
9      Scanner s = new Scanner( p.getInputStream() ).useDelimiter("\\Z");
10     System.out.println( s.next() );
11 }
```

Zum Starten eines externen Programms wird in Java üblicherweise die Klasse ***ProcessBuilder*** verwendet, welche im vorliegenden Fall beim Erzeugen eines Objekts drei Parameter erhält (Zeile 1-3). Hierbei gibt der erste Wert die zu startende Konsole an. Da dies abhängig vom verwendeten Betriebssystem ist, wurde der benötigte Befehl bereits bei Initialisierung in der Member-Variablen `cmd_line` gespeichert. Gleiches gilt für die beiden übrigen Parameter, welche nun für das Starten von Blender zuständig sind. Mit Hilfe von `arg` wird der Konsole mitgeteilt, dass sie den in `command` gespeicherten Befehl ausführen soll. Unter Linux kann dies also folgendermaßen aussehen:

```
bash -c ./blender ".blender/start.blend" &
```

Hierbei wären sämtliche Zeichen nach dem Parameter *-c* ein solcher auszuführender Befehl. Dieser würde Blender im Hintergrund starten und dort die angegebene Projektdatei (.blend) öffnen. Da allerdings noch nicht bekannt ist, in welchem Verzeichnis sich die Datei `blender` befindet, kann das Programm zunächst nicht gestartet werden. Um dieses Problem zu beheben, lässt sich, wie in Zeile 4 zu sehen, das jeweilige Verzeichnis mit Hilfe der Methode ***directory()*** angeben. Daraufhin kann der ***ProcessBuilder*** den Prozess starten und ihn in einem Objekt der Klasse ***Process*** speichern. Wurde beim Starten des JobListeners der Modus **debug** angegeben bzw. die gleichnamige boolesche Variable gesetzt, so wird in den Zeilen 9 und 10 zusätzlich auf Ausgaben des laufenden Prozesses reagiert.

4.4.3. Spezielle Funktionen unter Linux

Dieser Abschnitt beschäftigt sich mit Funktionen, welche speziell für Linux Betriebssysteme entwickelt wurden. Hierfür existiert keine entsprechende Windows-Variante, was aber nicht bedeutet, dass der ChartFlight Service dort nicht lauffähig ist. Die zusätzliche Funktionalität unter Linux erklärt sich aus der Tatsache, dass der Service hauptsächlich auf solchen

Systemen betrieben werden soll. Um dort eine gewisse Ausfallsicherheit zu garantieren, wurden dem Service zwei weitere Bestandteile hinzugefügt. Während der erste für die ständige Ausführung des JobListeners im Hintergrund zuständig ist, lässt sich mit Hilfe des zweiten Bestandteils Blender überwachen. In beiden Fällen werden hierfür spezielle Konsolenbefehle von Linux verwendet.
Die Überwachung des JobListeners wird mit Hilfe eines sogenannten Cronjobs durchgeführt. Hierbei handelt es sich um Aufgaben oder Aktionen, welche das Betriebssystem in einem festgelegten Intervall ausführt. Ein solcher Cronjob lässt sich über den Konsolenbefehl `crontab -e` und das Bearbeiten der hierdurch geöffneten Tabelle erzeugen. Zur Überwachung des JobListeners wird dort folgende Zeile eingetragen:

```
*/30 * * * * /home/lutz/joblistener/monitor.sh
```

Dieses Konstrukt startet nun zu jeder halben und vollen Stunde das Shell-Skript `monitor.sh`, was durch den Wert ***/30** in der ersten Spalte erreicht wird. Die übrigen vier Zeitspalten sind nacheinander für die Angabe einer bestimmten Stunde, eines Tages, eines Monats und eines Wochentages verantwortlich. Da das Shell-Skript allerdings in jeder Stunde, an jedem Tag usw. ausgeführt werden soll, lässt sich mit Hilfe Sternsymbols (*) eine Art Wildcard setzen. So kann der erste Wert auch als eine beliebige Minute, welche durch 30 teilbar ist, interpretiert werden. Zu diesem Zeitpunkt wurde allerdings noch keine Überwachung des JobListeners durchgeführt. Hierfür steht das angegebene Shell-Skript zur Verfügung, das im nachfolgenden Listing vorgestellt wird.

```
1  #!/bin/bash
2
3  working_dir="/home/korn/joblistener/"
4  cronjob_log="/home/korn/public_html/logs/cronjob.log"
5  application="JobListener.jar -mode multi"
6
7  if !(ps ax | grep -v grep | grep "$application" > /dev/null);
8  then
9  cd $working_dir
10 date +"%Y-%m-%d %H:%M:%S Couldn't find process named '$application' -> Starting
       new instance" >> $cronjob_log
11 java -jar $application &
12 fi
```

Wie bei Shell Skripten allgemein üblich, teilt dieses zunächst dem Interpreter mit, dass es sich bei der vorliegenden Datei um ein solches handelt. Daraufhin werden die Variablen `working_dir`, `cronjob_log` und `application` initialisiert. Während die ersten beiden Pfade das Verzeichnis des JobListener und einer Protokolldatei angeben, speichert die dritte

Variable den exakten Namen des Programms. Sämtliche Werte sollten vor dem Eintragen des Cronjobs in die oben genannte Tabelle entsprechend der jeweiligen Verzeichnisstruktur angepasst werden. Anschließend überprüft das Shell-Skript, ob sich bereits eine Instanz des JobListeners in der Ausführung befindet. Hierbei werden unter anderem die Konsolenbefehle **ps** und **grep** verwendet. Ersterer liefert die Prozesstabelle des Betriebssystems und mit dem zweiten Befehl kann diese daraufhin durchsucht werden. Somit lässt sich herausfinden, ob der Text in der Variablen `application` unter den laufenden Prozessen auftaucht. Trifft dies zu, so beendet das Skript seine Ausführung. Anderenfalls wird zunächst in das Arbeitsverzeichnis gewechselt und dort eine neue Instanz des JobListeners gestartet (Zeile 9, 11). Zudem protokolliert das Shell-Skript diesen Vorgang (Zeile 10).
Die zweite Funktion ist für die Überwachung einer laufenden Instanz von Blender zuständig und somit Aufgabe des JobListeners. Diese hängt stark mit der aktuellen Situation in der Datenbank zusammen, was bedeutet, dass dort ein Auftrag mit dem Status ***rendering_preview*** bzw. ***rendering*** vorhanden sein muss. Ähnlich wie beim Starten einer Instanz von Blender wird nun mit Hilfe der Klasse ***ProcessBuilder*** ein spezielles Shell-Skript aufgerufen. Dieses durchsucht die Prozesstabelle nach der laufenden Blender-Instanz und benachrichtigt den JobListener entsprechend. Auch hierfür werden unter anderem die Konsolenbefehle **ps** und **grep** verwendet. Erhält das Java Programm die Nachricht, dass sich keine Instanz von Blender in der Prozesstabelle befindet, so setzt dieses den Status des jeweiligen Auftrags auf ***waiting*** bzw. ***confirmed*** zurück. Abschließend soll im nachfolgenden Listing das Shell-Skript zur Überwachung von Blender dargestellt werden. Es besteht lediglich aus der oben erklärten Abfrage (Zeile 5) und einer entsprechenden Ausgabe von 0 oder 1 über den Befehl **echo** (Zeile 6, 7). Zudem wird die zu suchende Projektdatei (.blend) in einer Variablen gespeichert, wobei auch diese an die jeweilige Verzeichnisstruktur anzupassen ist.

```
1 #!/bin/bash
2
3 blend_file="/home/lutz/.blender/start.blend"
4
5 if (ps ax | grep "blender" | grep -v "grep" | grep $blend_file > /dev/null)
6 then echo 1
7 else echo 0
8 fi
```

4.5. Die Benutzerschnittstelle

Der vorliegende Abschnitt beschäftigt sich mit der Implementierung der Benutzerschnittstelle. In diesem sollen verschiedene Mechanismen vorgestellt werden, welche hauptsächlich im Hintergrund arbeiten. Hierzu zählen beispielsweise Klassen zum Erzeugen und Einlesen

von speziellen XML-Dateien oder zur Überprüfung verschiedener Benutzereingaben. Weiterhin wird ein Einblick in die Struktur solcher XML-Dateien gegeben und der Aufbau des verwendeten Java Applets ein wenig genauer betrachtet. Auf eine detaillierte Beschreibung verschiedener Designaspekte, wie beispielsweise dem Ein- und Ausblenden der erweiterten Einstellungen, soll allerdings verzichtet werden.

4.5.1. Überblick

Wie schon in den vorherigen Abschnitten soll auch hier zunächst ein Überblick über die diversen PHP-Skripte und die verwendete Verzeichnisstruktur gegeben werden. Da es sich in diesem Fall um eine Vielzahl von unterschiedlichen Dateien handelt, lässt sich nicht jede einzeln behandeln. Anhand der Verzeichnisstruktur sollen sowohl die verschiedenen PHP-Skripte als auch alle übrigen Elemente in mehrere Kategorien eingeteilt werden. Tabelle 4.9 listet die Namen der einzelnen Ordner auf und gibt zudem Informationen über die darin enthaltenen Dateien. Hierbei wurde diese in drei unterschiedliche Bereiche unterteilt. Während die erste Teiltabelle Verzeichnisse enthält, die besonders wichtig für das Funktionieren der Benutzeroberfläche sind, listet die zweite solche zum Erzeugen des Layouts und zusätzlicher Informationen auf. In der letzten Teiltabelle befinden sich die Ordner, auf welche nicht nur die PHP-Skripte zugreifen, sondern auch die anderen Komponenten der ChartFlight Services.

.	Hauptverzeichnis der Benutzerschnittstelle, verschiedene Startseiten
applet	PHP-Skript zum Starten des Applets samt benötigter Java Klassen
service	PHP-Skripte für Benutzereingaben (diverse Formulare, Informationsseiten)
settings	Einstellungen der Benutzerschnittstelle
templates	grundlegende PHP-Skripte zur Präsentation der Ergebnisse (Bilder, Videos)
tools	PHP-Klassen und Funktionen für verschiedene Aufgaben (Validierung, XML-Dateien)
css	Informationen über das Seitenlayout (main.css)
examples	Webseiten mit unterschiedlichen Beispielen
images	Bilddateien für Layout, Beispiele und Anleitungen
javascript	verschiedene JavaScript Dateien (Tigra Color Picker, Tooltips)
tutorials	Webseiten mit unterschiedlichen Anleitungen
videos	Ordner für diverse Beispielvideos
email	Informationen zum Versenden von E-Mails
logs	Protokolldateien für Blender, den JobListener und den Cronjob
users	Benutzerdaten in diversen Unterverzeichnissen, SQLite-Datenbank

Tabelle 4.9.: Verzeichnisse der Benutzeroberfläche

Im Nachfolgenden sollen lediglich die Verzeichnisse der oberen Teiltabelle ein wenig detaillierter beschrieben werden, da diese die Grundlage der Benutzeroberfläche bilden. Hierunter befindet sich zunächst das Hauptverzeichnis, welches neben der Datei `index.php` auch die Startskripte aller Kategorien der Webseite enthält. Für das Funktionieren des ChartFlight

Services spielt in diesem Ordner vor allem die Datei `start_service.php` eine wichtige Rolle. Von dort aus gelangt man in das Verzeichnis **service**. Die darin enthaltenen PHP-Skripte sind allesamt für das Entgegennehmen und die Bearbeitung eines Auftrags zuständig. So befinden sich darunter die diversen Formularseiten, der Haftungsausschluss und sämtliche Seiten zur Bestätigung der unterschiedlichsten Aktionen. Der Ordner **applet** beinhaltet alle für das Java-Applet benötigten Klassen und das entsprechende PHP-Skript zum Ausführen. Dieses unterscheidet sich allerdings von der gleichnamigen Datei `applet.php` aus dem Verzeichnis **service**. Hierbei handelt es sich lediglich um ein Teilformular, welches mit dem eigentlichen Starten des Applet nichts zu tun hat, sondern nur den Link zur entsprechenden Webseite bereitstellt. Der Ordner **settings** beinhaltet zum aktuellen Zeitpunkt lediglich eine einzige PHP-Datei. Diese stellt sämtliche Einstellungen zur Verfügung, die in Bezug auf die Benutzeroberfläche gemacht werden können. Hierunter befinden sich beispielsweise Informationen über Namen und Verzeichnis der SQLite-Datenbank oder über die Anzahl der maximal anzunehmenden Aufträge. Der Ordner **templates** enthält Vorlagen für die Präsentation der Vorschaubilder oder des gerenderten Videos. Diese werden allerdings nicht an Ort und Stelle verwendet, sondern nach Absenden eines Auftrags in das jeweilige Benutzerverzeichnis kopiert. Somit erhält jeder Benutzer seine eigenen Präsentationsseiten, auf welche nur mit der richtigen Auftragsnummer zugegriffen werden kann. Die URL dieser Seiten wird dem Benutzer entsprechend per E-Mail mitgeteilt. Das letzte Verzeichnis mit dem Namen **tools** beinhaltet wichtige Klassen und Funktionen für die unterschiedlichsten Aufgaben. So befinden sich hierunter beispielsweise Klassen zur Überprüfung der Benutzereingaben und der hochgeladenen Dateien, sowie solche zum Schreiben und Lesen von XML-Dokumenten. Da diese neben den PHP-Skripten aus dem Verzeichnis **service** die Hauptfunktionalität der Benutzerschnittstelle ausmachen, sollen sie abschließend in Tabelle 4.10 aufgelistet und deren Aufgaben erläutert werden.

CSVValidation.php	Validierung der hochgeladenen CSV-Dateien
DataValidation.php	Validierung der übrigen Benutzereingaben
HTMLFunctions.php	Funktionen zum Erzeugen diverser HTML-Konstrukte
SettingsReader.php	Laden gespeicherter Benutzereinstellungen aus einer XML-Datei
SettingsWriter.php	Schreiben sämtlicher Benutzereinstellungen in eine XML-Datei
Upload.php	Klasse zum Hochladen der unterschiedlichen Dateien

Tabelle 4.10.: Klassen des Verzeichnisses **tools** und deren Aufgaben

4.5.2. Implementierung

Dieser Unterabschnitt soll nun einen detaillierteren Einblick in die Implementierung der Benutzeroberfläche geben. Allerdings werden hierbei lediglich die technischen Aspekte betrachtet und auf die Beschreibung unterschiedlicher Verfahren bezüglich des Layouts verzichtet. Zunächst beschäftigt sich der Abschnitt mit den PHP-Skripten, auf welche während des

Erzeugens eines Auftrags zurückgegriffen wird. Wie bereits erwähnt, befinden sich diese hauptsächlich in den Verzeichnissen **service** und **tools**. Abschließend soll ein kurzer Blick auf die Implementierung des verwendeten Java-Applets geworfen werden.
Die Grundfunktionalität der gesamten Benutzeroberfläche basiert auf der Möglichkeit mit Hilfe von PHP sogenannte Sessions anzulegen. Hierbei handelt es sich um eine Technik zum Zwischenspeichern verschiedener Werte in einem assoziativen Array. Im Gegensatz zu normalen Variablen bleibt dieses auch über mehrere Webseiten hinweg erhalten und ist somit eine ideale Methode zum Arbeiten mit unterschiedlichen Formularen. Zum Verwenden einer solchen Session bedarf es einer einfachen Angabe der Funktion `session_start()` am Anfang jedes daran beteiligten PHP-Skripts. Daraufhin steht das assoziative Array mit dem Namen `$_SESSION` zur Verfügung. Hierin lassen sich nun beliebige Werte, andere Arrays oder Objekte speichern. Benötigt man das Array anschließend nicht mehr, so können über die Befehle `session_unset()` und `$_SESSION = array()` die darin enthaltenen Werte entfernt werden. Mit Hilfe dieser Technik ist es nun möglich ein komplexeres Eingabeformular für sämtliche Benutzerdaten über mehrere Seiten zu verteilen. Hiermit können allerdings lediglich die unterschiedlichen Eingaben zwischengespeichert werden, was zum endgültigen Erzeugen eines Auftrags nicht ausreicht.
Die zweite wichtige Aufgabe der Benutzeroberfläche ist der Zugriff auf die SQLite-Datenbank. Dies wird an verschiedenen Stellen benötigt und bedarf deshalb einer zusätzlichen Behandlung. Wie bereits in Abschnitt 2.2 vorgestellt, lässt sich in PHP mit Hilfe der sogenannten PDO-Treiber eine Verbindung zu einer solchen Datenbank aufbauen. Dies wird zum ersten Mal vor dem Starten eines Auftrags getan und zunächst überprüft, wie viele Einträge sich in der Datenbank befinden. Hierbei ist allerdings zu erwähnen, dass lediglich zu bearbeitende Aufträge beachtet und bereits fertig gestellte ignoriert werden. Ist diese Zahl größer oder gleich der maximal erlaubten Aufträge, so teilt die Startseite dem Benutzer mit, dass die Datenbank die volle Kapazität erreicht hat. Für einen solchen Vergleich wird die Variable `$max_job_count` aus der Konfigurationsdatei `settings.php` herangezogen. Ist andererseits die Anzahl dieser Aufträge kleiner als `$max_job_count`, so stellt das PHP-Skript dem Benutzer den entsprechenden Link zum Starten eines neuen zur Verfügung.
Der nächste wichtige Datenbankzugriff wird nach dem Betätigen der Schaltfläche zum Absenden eines Auftrags im letzten Teilformular durchgeführt. Hierbei entnimmt das PHP-Skript dem oben vorgestellten Session-Array die entsprechenden Werte und trägt sie in die Datenbank ein. Allerdings muss für diesem Vorgang zunächst überprüft werden, ob es sich beim aktuellen Auftrag um einen neuen oder einen abgeänderten handelt. Ersteres erfordert das Hinzufügen einer weiteren Zeile zur Datenbank, was mit Hilfe einer `INSERT`-Operation erreicht wird. Handelt es sich aber um einen bereits vorhandenen Auftrag, bei welchem lediglich einige Einstellungen verändert wurden, so lässt sich unter Verwendung der entsprechenden `UPDATE`-Anweisung die Datenbank aktualisieren, ohne einen neuen Eintrag zu erzeugen.
Der letzte Datenbankzugriff findet beim Bestätigen eines Auftrags auf der Präsentationsseite

der Vorschaubilder statt. Dabei wird erneut eine UPDATE-Anweisung benötigt, welche den Status und das aktuelle Datum des entsprechenden Auftrags abändert. Zudem setzt diese gegebenenfalls das Feld **notified** auf Null zurück. Ebenso kann auf der oben genannten Webseite ein Auftrag aus der Datenbank entfernt werden. Hierzu lässt sich mit Hilfe der SQL-Operation DELETE der entsprechende Eintrag löschen.
Weiterhin existieren noch einige kleinere Zugriffe auf die Datenbank, welche zur Abfrage der Position eines Auftrags in der Warteschlange dienen. In diesen Fällen erfolgt keine Manipulation der Datenbank, sondern lediglich das Verwenden eines SELECT-Statements.
Wie oben erwähnt, wird nach dem Absenden eines Auftrag im letzten Teilformular dieser in die Datenbank eingetragen. Zudem ist das zugrundeliegende PHP-Skript für weitere wichtige Aufgaben zuständig. So erzeugt es, noch bevor ein Eintrag in die Datenbank erfolgt, die entsprechende Einstellungsdatei im XML-Format. Hierfür verwendet es die für diesen Zweck entwickelte PHP-Klasse ***SettingsWriter***, welcher die aktuell aufgezeichneten Daten in Form des Arrays $_SESSION übergeben werden. Daraufhin lässt sich mit Hilfe des Pfades zum jeweiligen Benutzerverzeichnis die XML-Datei mit sämtlichen Einstellungen erzeugen. Eine genauere Beschreibung des eigens hierfür entwickelten XML-Formats stellt Unterabschnitt 4.5.4 zur Verfügung. Dort werden auch die Klassen zum Einlesen solcher Dateien behandelt. Eine weitere wichtige Aufgabe des PHP-Skripts ist das Kopieren der Templates für die Präsentation der Ergebnisse. Dieser Vorgang wird mit Hilfe der bereits vorhandenen Funktion ***copy()*** durchgeführt. Hierbei sind sowohl Quell- als auch Zieldatei samt Pfad als Parameter zu übergeben. Anschließend müssen unter Verwendung der Funktion ***chmod()*** die Benutzerrechte der gerade kopierten Datei entsprechend angepasst werden, damit auch der JobListener beim Löschen darauf zugreifen kann.

Die Hauptfunktionalität der Benutzeroberfläche steckt allerdings nicht nur in den einzelnen Formularseiten, sondern auch in den Klassen des Verzeichnisses **tools**. Diese wurden in Tabelle 4.10 bereits aufgelistet und sollen nachfolgend ein wenig detaillierter betrachtet werden. Beginnen wir mit der Klasse ***DataValidation***, welche für die Korrektheit der Benutzereingaben zuständig ist. Hierbei lassen sich beliebige Eingabefelder auf unterschiedliche Werte überprüfen, welche der Klasse unter Verwendung des Arrays $_POST übergeben werden. Ein weiteres assoziatives Array gibt daraufhin die zu überprüfenden Werte der jeweiligen Eingabefelder an. Die Erzeugung eines Objekts vom Typ ***DataValidation*** kann deshalb folgendermaßen aussehen:

```
1 $mixed_fields = array('text' => NULL,
2                       'email1' => "email",
3                       'email2' => '==' . $_POST['email1'],
4                       'color' => "/^#[a-fA-F0-9]{6}$/");
5 $mixedValidation = new DataValidation($_POST, $mixed_fields);
```

Wie in obigem Listing zu sehen, enthält das assoziative Array `$mixed_fields` als Schlüssel die Bezeichnung des jeweiligen Eingabefeldes und als Wert einen Befehl zu dessen Überprüfung. Hierbei lassen sich unterschiedliche Angaben machen. So kann mit Hilfe des Wertes `NULL` der Klasse ***DataValidation*** mitgeteilt werden, dass diese das entsprechende Formularfeld lediglich auf eine leere Eingabe testen soll. Ein Spezialfall stellt die zweite Zeile des Listings dar. Mit der Angabe von `"email"` als Wert auf der rechten Seite, lässt sich überprüfen, ob eine korrekte E-Mail-Adresse eingegeben wurde. Hierzu verwendet das PHP-Skript eine statische Methode der Klasse ***Validate***, welche bereits bei der Installation von PHP vorhanden sein sollte. Ist dies nicht der Fall, so lässt sie sich durch einfaches Kopieren in das Verzeichnis **tools** nachinstallieren. Im Gegensatz dazu soll mit der Angabe aus Zeile 3 ein weiteres Textfeld auf Gleichheit mit einem vorgegebenen Wert getestet werden. Durch das Voranstellen zweier Gleichheitszeichen (==) an den zu überprüfenden Text lässt sich diese Vorgehensweise der Klasse ***DataValidation*** mitteilen. Ein solches Konstrukt ist beispielsweise beim Abgleich zweier Textfelder mit derselben E-Mail-Adresse sinnvoll, was auch das obige Listing darstellt. Zu guter Letzt wird mit Hilfe von regulären Ausdrücken eine komplexere Möglichkeit zum Testen von Eingaben zur Verfügung gestellt. Mit solchen können Zeichenketten auf eine beliebige Struktur überprüft und bei Übereinstimmung als korrekt angesehen werden. Das Beispiel in Zeile 4 testet nun auf diese Weise, ob ein Textfeld eine erlaubte Farbe in hexadezimaler Form enthält.
Zudem ist es auch möglich, anstatt des oben erwähnten assoziativen Arrays, lediglich ein normales mit den Namen der zu überprüfenden Eingabefelder beim Erzeugen des Objekts zu übergeben. Dies hat zur Folge, dass die Klasse alle angegebenen Formularelemente ausschließlich auf leere Einträge testet. Hierzu muss allerdings ein dritter Parameter mit dem Wert `TRUE` vorhanden sein, was Zeile 2 des nachfolgenden Listings zeigt.

```
1 $text_fields = array('text1', 'text2', 'text3');
2 $textValidation = new DataValidation($_POST, $text_fields, TRUE);
3
4 $color_fields = array('color1', 'color2');
5 $colorValidation = new DataValidation($_POST, $color_fields, TRUE,
6                                       "/^#[a-fA-F0-9]{6}$/");
```

Obige Funktionalität lässt sich durch die Angabe eines regulären Ausdrucks als vierten Parameter erweitern. Auf diese Weise werden die Eingabefelder nicht auf leere Einträge überprüft, sondern unter Verwendung des regulären Ausdrucks validiert. Wie in Zeile 5 dargestellt, lassen sich somit Textfelder auf die korrekte Eingabe von Farben in hexadezimaler Form testen. Bis zum aktuellen Zeitpunkt wurden lediglich die entsprechenden Objekte der Klasse ***DataValidation*** erzeugt, allerdings noch keine Eingaben tatsächlich validiert. Dies lässt sich durch einen Aufruf der Methode ***validate()*** erreichen, woraufhin das übergebene Array `$_POST` auf Basis der oben erklärten Möglichkeiten überprüft und die Menge aller inkorrekten Ein-

gaben gesichert wird. Mit deren Hilfe lässt sich dem Benutzer anschließend mitteilen, in welchen Formularfeldern er fehlerhafte Eingaben gemacht hat. Hierbei werden unterschiedliche Techniken, wie das Einfärben der entsprechenden Elemente oder das Einblenden eines Warnungstextes, verwendet.

Die nächste wichtige PHP-Klasse, welche im Zuge der Entwicklung dieses Services implementiert wurde, ist für das Hochladen der unterschiedlichen Dateien verantwortlich. Hierbei wird sowohl deren Dateityp als auch deren Größe analysiert und je nach Vorgaben die jeweilige Datei entweder akzeptiert oder abgelehnt. Wie auch schon zuvor, soll zunächst allerdings das Erzeugen eines Objekts der Klasse ***Upload*** vorgestellt und anschließend erklärt werden, wie dieses die erhaltenen Informationen verarbeitet.

```
1  $field_name = "map";
2  $allowed_types = array('image/jpg', 'image/png', 'image/gif');
3  $max_size = 5497558138880;
4
5  $upload = new Upload($allowed_types, $max_size, $user_folder);
6  $upload->copy($_FILES[$field_name], $_SESSION['user_id'] . "_" . $field_name);
7
8  if ( $upload->isSuccessful() ) {
9    echo 'Dateigröße: ' . $upload->getResult('size');
10 }
```

Wie in obigem Listing zu sehen, handelt es sich dabei um den Quellcode zum Hochladen der zugrundeliegenden Karte. Speziell für diese befindet im entsprechenden Formular ein Eingabefeld mit dem Namen ***map***, über welches sich eine Bilddatei auswählen lässt. Hierfür sollen drei unterschiedliche Datei- bzw. Mime-Typen akzeptiert und eine Dateigröße von 5 Megabyte nicht überschritten werden. Diese Informationen speichert das PHP-Skript zunächst in den Variablen `$allowed_types` und `$max_size` und greift anschließend in Zeile 5 darauf zu. Dort wird ein Objekt der Klasse ***Upload*** erzeugt, welches drei Parameter entgegennimmt. So erhält ein solches Objekt, neben den oben erwähnten Werten, zusätzlich den Pfad zu einem Verzeichnis. Dieses liefert den Ort, an den eine hochgeladene Datei kopiert werden soll. Selbstverständlich enthält die Variable `$user_folder` den absoluten Pfad zum jeweiligen Benutzerverzeichnis.
Wurde nun das Objekt erzeugt, so lässt sich die entsprechende Datei mit Hilfe der Methode ***copy()*** zunächst auf die unterschiedlichen Vorgaben überprüfen und bei Erfolg in den Benutzerordner kopieren. Hierzu muss diese allerdings über ein spezielles HTML-Formular hochgeladen werden, woraufhin sie temporär über das superglobale Array `$_FILES` zur Verfügung steht. Wie in Zeile 6 zu sehen, verlangt die Methode ***copy()*** zunächst die temporäre Datei und daraufhin deren neuen Namen als Parameter. Letzteres setzt sich aus der Auftragsnummer, welche in `$_SESSION['user_id']` gespeichert ist, einem Unterstrich und

der Bezeichnung des jeweiligen Eingabefeldes zusammen. Auf diese Weise lassen sich beispielsweise auch mehrere Dateien vom gleichen Typ und gleicher Größe in das vorgegebene Verzeichnis kopieren. Bei einem solchen Kopiervorgang kann es allerdings vorkommen, dass entweder eine zu große Datei oder ein falscher Mime-Typ hochgeladen wurde. In diesem Fall verweigert die Methode das Kopieren der entsprechenden Datei und speichert den jeweiligen Fehler für eine spätere Abfrage. Mit Hilfe der Methode ***isSuccessful()*** kann nun überprüft werden, ob der letzte getätigte Kopiervorgang erfolgreich war oder nicht. In beiden Fällen lassen sich darüber zusätzliche Informationen, wie beispielsweise die tatsächliche Dateigröße oder eventuell aufgetretene Fehler, erhalten. Hierzu steht die Methode ***getResult()*** zur Verfügung, mit welcher über Schlüsselwörter auf die unterschiedlichsten Informationen zugegriffen werden kann. Ein einfaches Beispiel zeigen die Zeilen 8 bis 10 des obigen Listings, in dem, für den Fall eines erfolgreichen Uploads, die entsprechende Dateigröße mit Hilfe des Befehls **echo** ausgegeben wird.

Als nächstes soll nun eine Klasse betrachtet werden, welche von dem gerade vorgestellten PHP-Skript für den Dateiupload abhängig ist. Diese beschäftigt sich mit der Validierung der hochgeladenen Diagrammdaten in Form einer CSV-Datei, deren konkreter Aufbau sich in Abschnitt 3.4 befindet. Wie schon bei den beiden vorherigen Klassen, soll auch hier zunächst die Erzeugung eines Objekts vom Typ ***CSVValidation*** betrachtet und anhand dessen die unterschiedlichen Methoden erläutert werden.

```
1  $csvVal = new CSVValidation($user_folder . "/" . $_SESSION['diagram_data'],
2                              $_SESSION['diagram_type'],
3                              $_SESSION['use_summary_chart'],
4                              $_SESSION['summary_diagram_type']);
5
6  $csvVal->validate();
7
8  if ( count($csvVal->getErrors()) == 0 ) {
9    $csvVal->writeFile();
10 }
```

Was es benötigt, um ein solches Objekt zu generieren, zeigt das obige Listing anhand bereits bestehender Variablen. Bei dessen Initialisierung werden vier unterschiedliche Parameter angegeben (Zeile 1-4). Zunächst muss das Objekt selbstverständlich wissen, welche CSV-Datei überprüft werden soll. Diese Information befindet sich im Feld `diagram_data` des assoziativen Arrays `$_SESSION`, welches zu Beginn des Abschnitts behandelt wurde. Darin ist allerdings lediglich der Name der zu überprüfenden CSV-Datei enthalten; den Pfad dorthin liefert die Variable `$user_folder`. Mit Hilfe des nächsten Parameters lässt sich der vom Benutzer gewählte Diagrammtyp übergeben. Dieser wird benötigt um festzustellen, welche numerischen Werte innerhalb eines Datensatzes erlaubt sind. So existieren beispielsweise

Unterschiede zwischen denen eines Kreis- und eines Liniendiagramms. Letzteres akzeptiert zusätzlich auch Werte kleiner Null. Gleiches gilt für den Typ des Schlussdiagramms, welcher allerdings lediglich bei der Generierung eines solchen in Betracht gezogen wird. Diese Informationen werden unter Verwendung der beiden übrigen Parameter angegeben (Zeile 3, 4).
Nach der erfolgreichen Erzeugung des Objektes lässt sich nun mit der Methode ***validate()*** die CSV-Datei auf Korrektheit testen (Zeile 6). Hierbei werden intern zwei unterschiedliche Methoden verwendet, welche wir im Nachfolgenden betrachten. Die erste trägt den Namen ***validateStructure()*** und ist für die Überprüfung des Aufbaus einer CSV-Datei zuständig. Dazu wird eine solche Datei zunächst geöffnet und mit Hilfe der in PHP verfügbaren Funktion ***fgetcsv()*** Schritt für Schritt eingelesen. Während dieses Vorgangs lässt sich anhand diverser Abfragen überprüfen, ob die jeweilige Struktur eines Datensatzes korrekt ist. Typische Fehler wären beispielsweise Zeilen mit unterschiedlich vielen Einträgen oder solche mit weniger als zwei Spalten. Eine weitere Aufgabe dieser Methode ist das Zwischenspeichern sämtlicher Werte in mehrdimensionalen Arrays. Hierdurch wird für die nachfolgende Bearbeitung die Möglichkeit gegeben, komfortabel auf die Informationen der CSV-Datei zugreifen zu können, ohne sie ein weiteres Mal öffnen zu müssen. Auf Basis solcher zwischengespeicherter Daten arbeitet unter anderem die zweite Methode mit der Bezeichnung ***validateValues()***. Diese ist für die Überprüfung sämtlicher Werte und deren Vorkommen zuständig. So meldet sie beispielsweise einen Fehler, wenn innerhalb eines Datensatzes nicht-numerische Werte auftauchen oder eine Zeile mehr als die insgesamt erlaubten neun Spalten enthält. Zudem berücksichtigt sie auch die bei der Initialisierung übergebenen Informationen über den Diagrammtyp und das Schlussdiagramm.
Nach dem Abschluss beider Methoden lässt sich unter Verwendung einer weiteren mit dem Namen ***getErrors()*** eine Liste aller Fehler samt zugehöriger Zeilennummer erfragen. Ist diese leer, so sind bei der Überprüfung der CSV-Datei keinerlei Fehler aufgetreten (Zeile 8-10). In einem solchen Fall können daraufhin die zwischengespeicherten Daten in eine korrekte Form gebracht und in einer neuen CSV-Datei gesichert werden. Hierfür ist die Methode ***writeFile()*** zuständig, welche auf Basis bestehender PHP-Funktionen diese Aufgabe durchführt. Mit einer solchen Vorgehensweise wird nun sichergestellt, dass die Struktur jeder CSV-Datei stets die gleiche ist, ohne dabei im Gegenzug den Benutzer bei deren Erstellung zu sehr zu überfordern. So bleibt diesem beispielsweise die Freiheit am Ende einer CSV-Datei eine oder sogar mehrere Leerzeilen einzufügen, während intern immer ersteres zutrifft.

Für die letzten beiden Klassen mit den Namen ***SettingsWriter*** und ***SettingsReader*** soll lediglich das Erzeugen eines Objekts und dessen Verwendung erläutert werden. Allerdings finden sich weitere Informationen hierüber und ebenso über das verwendete XML-Format in Abschnitt 4.5.4. Das nachfolgende Listing zeigt anhand einer Einstellungsdatei im XML-Format, welche Methoden nötig sind, um eine solche zu erstellen bzw. einzulesen. Diese befindet sich in einem Benutzerverzeichnis, das auch hier in der Variablen `$user_folder` gespeichert wurde.

```
1 $settingsWriter = new SettingsWriter($_SESSION);
2 $settingsWriter->writeFile($user_folder . "/",
3                            $_SESSION['user_id'] . "_settings.xml");
4
5 $settingsReader = new SettingsReader();
6 $settingsReader->readFile($user_folder . "/",
7                           $_SESSION['user_id'] . "_settings.xml");
```

Wie in obigem Listing zu sehen, weisen die beiden Klassen bei deren Verwendung einige Gemeinsamkeiten auf. So lässt sich nach dem Erzeugen eines Objekts mit Hilfe einer speziellen Methode die jeweilige Aufgabe ausführen (Zeile 2, 6). Diese erhält als ersten Parameter ein gültiges Verzeichnis und als zweiten den Namen der zu betrachtenden Datei. Hierbei entsteht somit der Vorteil, dass sich die Methoden ***writeFile()*** und ***readFile()*** in ihrer Handhabung nicht unterscheiden, obwohl sie vollkommen verschiedene Aufgaben erfüllen. Während erstere für das Erstellen einer XML-Datei mit allen benötigten Einstellungen zuständig ist, lässt sich mit der zweiten eine solche einlesen und anschließend darauf zugreifen. Speziell zu diesem Zweck steht die Methode ***getData()*** der Klasse ***SettingsReader*** zur Verfügung. Zum Schreiben einer XML-Datei benötigt die Klasse ***SettingsWriter*** allerdings noch die entsprechenden Daten, welche bei der Initialisierung in Zeile 1 in Form einer Referenz auf das Array `$_SESSION` übergeben werden.

4.5.3. Java-Applet für Diagrammpositionen

Dieser Unterabschnitt beschäftigt sich mit dem Java-Applet, welches für die Angabe der Diagrammpositionen zur Verfügung steht. Hierhin soll dessen grundlegende Struktur vorgestellt und anhand von Codebeispielen erläutert werden. Wie der Aufbau eines Applets im Allgemeinen aussieht lässt sich allerdings in Abschnitt 2.4 erfahren.
Im Gegensatz zu dieser Implementierung wurden im vorliegenden Fall einige Modifikationen vorgenommen. So betrifft die erste Änderung das Ausgeben von Grafiken auf dem Bildschirm. Hierbei kann es nicht selten zu einem Flackern kommen, welches bei wiederholtem Zeichnen der Grafiken auftaucht und auf rechenintensive Operationen zurückzuführen ist. Um diesem Effekt entgegenzuwirken, wird das sogenannte Double Buffering angewandt. Hierbei handelt es sich um eine Technik, welche das Zeichnen der gesamten Grafik in den Hintergrund verlagert und das Ergebnis erst nach Abschließen dieses Prozesses dem Benutzer präsentiert. So können rechenintensive Operationen keine zeitlichen Verzögerungen und somit kein Bildschirmflackern auslösen, da der tatsächliche Zeichenvorgang dem Benutzer verborgen bleibt. Das nachfolgende Listing zeigt anhand einiger Ausschnitte des Quellcodes, wie die vorgestellte Technik implementiert werden kann. Dabei verzichtet dieses aus Platzgründen aber auf das Zeichnen komplexerer Objekte. Für den gesamten Vorgang werden ausschließlich Klassen

der Java API und deren Methoden verwendet.

```
1  private Image buffer;
2  private Graphics graphics;
3
4  public void init() {
5    this.buffer = this.createImage(this.getSize().width, this.getSize().height);
6    this.graphics = this.buffer.getGraphics();
7  }
8
9  public void paint(Graphics g) {
10   ...
11   this.graphics.fillOval(50, 30, 20, 20);
12   ...
13   g.drawImage(this.buffer, 0, 0, this);
14 }
```

Zur Implementierung des Double Buffering Verfahrens sind zunächst zwei Variablen vonnöten (Zeile 1, 2). Erstere soll die Zeichenfläche zur Verfügung stellen, auf welcher sich später im Hintergrund arbeiten lässt. Zu diesem Zweck wird innerhalb der Methode ***init()*** in Zeile 5 ein Objekt des Typs ***Image*** mit den entsprechenden Abmessungen des Applets erzeugt. Auf dessen Basis kann nun die zweite Variable mit dem Namen `graphics` initialisiert werden. Diese dient lediglich dem Zwischenspeichern eines Objekts der Klasse ***Graphics***, welches über die entsprechende Methode der Instanz des Typs ***Image*** verfügbar ist (Zeile 6). Im Gegensatz zu der in Kapitel 2 vorgestellten Variante können nun mit Hilfe des gerade erhaltenen Objekts sämtliche Grafiken gezeichnet werden. Die Zeilen 10 bis 12 sollen diesen Vorgang zeigen, wobei dort selbstverständlich wesentlich komplexere Anweisungen stehen können. Zu diesem Zeitpunkt, also vor dem Ausführen der dreizehnten Zeile, kann der Benutzer keine Änderungen an der angezeigten Grafik feststellen, obwohl bereits alle Operationen abgeschlossen sind. Die Erklärung hierfür ist, dass sämtliche grafischen Elemente auf einer unabhängigen Zeichenfläche im Hintergrund erzeugt wurden. Um nun das zugrundeliegende Objekt der Klasse ***Image*** für den Benutzer sichtbar zu machen, muss dieses lediglich mit Hilfe der Methode ***drawImage()*** auf die Zeichenfläche des Applets übertragen werden. Hierzu lässt sich, in Anlehnung an das Beispiel aus Abschnitt 2.4, das ***Graphics***-Objekt der Methode ***paint()*** verwenden. Auf diese Weise kann also zunächst eine Grafik erzeugt und erst nach deren Fertigstellung über das Applet anzeigt werden.

Die Hauptaufgabe des Applets ist es, die Punkte für die Diagrammpositionen entgegenzunehmen. Hierzu sind zunächst drei wichtige Funktionen zu implementieren. Als Erstes muss die zugrundeliegende Karte geladen und angezeigt werden. Auf Basis dieser kann der Benutzer nun an beliebigen Stellen durch Betätigen der rechten Maustaste eine Markierung platzieren.

Schlussendlich überträgt eine Methode alle gesetzten Positionen an das entsprechende PHP-Skript. Selbstverständlich besitzt das Applet noch weitere Funktionen, auf welche allerdings nicht detaillierter eingegangen werden soll, da die oben genannten dessen Hauptfunktionalität ausmachen.
Zum Laden von Bilddateien in Applets ist wichtig zu wissen, dass ein solches zunächst von dem verwendeten Browser heruntergeladen und auf dem Computer des Benutzers ausgeführt wird. Soll dieses nun ein Bild anzeigen, welches sich auf dem jeweiligen Server befindet, so muss eine entsprechende URL zur Verfügung stehen. In Java ist dafür bereits die gleichnamige Klasse vorhanden. Wie in Zeile 1 des nachfolgenden Listings zu sehen, erhält ein Objekt dieser bei der Initialisierung die oben erwähnte URL in Form einer Zeichenkette.

```
1 this.image_url = new URL(getParameter("map_image") + "?forcereload=" + rand);
2 this.image = getImage(this.image_url);
3
4 this.tracker = new MediaTracker(this);
5 this.tracker.addImage(this.image, 0);
6 this.tracker.waitForAll();
```

Da sich die entsprechende Bilddatei allerdings bei jedem Benutzer an einer unterschiedlichen Stelle im Dateisystem des Servers befindet, muss diese dem Applet von außen her mitgeteilt werden. Hierzu lässt sich dem bereits erwähnten `applet`-Tag, welches dem Platzieren und Starten des Applets innerhalb einer HTML-Seite dient, ein Unterelement mit der Bezeichnung `param` hinzufügen. Dieses besitzt die Attribute `name` und `value` und funktioniert im Prinzip wie eine normale Variable. Im vorliegenden Fall würde also das erste Attribut den Wert *"map_image"* und das zweite die entsprechende URL enthalten. Auf letzteres lässt sich nun innerhalb des Java-Codes mit Hilfe der Methode ***getParameter()*** der Applet-Klasse und der Angabe von *"map_image"* zugreifen. Diese URL wird zusätzlich um eine Zufallszahl ergänzt, was das erneute Laden der benötigten Bilddatei bei jedem Start des Applets erzwingt. Anderenfalls würde bei mehrfacher Ausführung stets auf die Bilddatei im Browsercache zurückgegriffen und eine gegebenenfalls neu hochgeladene Grafik vernachlässigt werden. Selbstverständlich geschieht dies nur dann, wenn zwei unterschiedliche Bilddateien den selben Namen besitzen. Durch deren Umbenennen nach dem Hochladen kann eine solche Situation allerdings für jeden Benutzer auftreten.
Mit Hilfe dieser URL und der Methode ***getImage()*** lässt sich nun ein Objekt der Klasse ***Image*** erzeugen, welches daraufhin in Zeile 5 an den zuvor initialisierten ***MediaTracker*** übergeben wird. Hiermit ist es nun möglich das gerade registrierte Bild sofort zu laden und nicht bis zu den Zeitpunkt zu warten, an welchem das Applet es tatsächlich benötigt. Dieser Vorgang wird über einen Aufruf der Methode ***waitForAll()*** in Zeile 6 erreicht. Nach Abschluss des oben gezeigten Quellcodes steht somit die entsprechende Bilddatei zur Verfügung und kann beliebig bearbeitet und positioniert werden.

Nach dem Beenden der ***init()***-Methode kann der Benutzer nun mit dem Applet interagieren und unter anderem die verschiedenen Diagrammpositionen angeben. Bei jedem Betätigen der rechten Maustaste wird daraufhin ein neuer Punkt den bereits bestehenden hinzugefügt und dieser in einer dafür vorgesehenen Liste gespeichert. Somit kann bei einem Aufruf der ***paint()***-Methode auf die Menge aller Punkte zurückgegriffen werden, um sie anschließend erneut zu zeichnen. Weiterhin dient eine solche Liste der Verwaltung aller Diagrammpositionen und lässt sich gegebenenfalls entsprechend manipulieren. So existiert beispielsweise eine Schaltfläche, welche eine einfache Methode zum Leeren der Liste aufruft und somit sämtliche Punkte entfernt. Zur Steuerung des Applets mit der Maus müssen, wie in Java üblich, die Interfaces ***MouseListener***, ***MouseMotionListener*** und ***MouseWheelListener*** implementiert werden. Für die Grundfunktionalität reicht hierbei jedoch ersteres aus. Dort lässt sich durch Implementierung der Methode ***mouseReleased()*** das Verhalten der rechten Maustaste festlegen. Weiterhin spielt ebenso die Position des Mauszeigers eine wichtige Rolle, da über diese die Koordinaten des zu erzeugenden Punktes angegeben werden. Hierzu lassen sich die Methoden ***getX()*** und ***getY()*** des entsprechenden ***MouseEvents*** verwenden, welches sämtliche Informationen bezüglich der Maus enthält. Auf Basis beider Werte kann nun ein neuer Punkt erzeugt und der List aller Diagrammpositionen hinzugefügt werden. Anschließend lässt sich dieser durch Neuzeichnen des Applets mit Hilfe der Methode ***repaint()*** darstellen. Prinzipiell würde eine solche Vorgehensweise ausreichen, allerdings bietet der finale Quellcode einige Erweiterungen, wie beispielsweise eine Überprüfung der aktuellen Mausposition, um ausschließlich Punkte im Bereich der Karte zu erlauben.
Wurden nun alle Diagrammpositionen festgelegt, so müssen sie an das zugrundeliegende PHP-Skript übertragen werden. Zu diesem Zweck lässt sich auf die eigens dafür entwickelte Methode ***savePoints()*** zurückgreifen, deren Arbeitsweise das nachfolgende Listing darstellt.

```
1  URLConnection conn = this.getDocumentBase().openConnection();
2  connection.setDoOutput(true);
3  OutputStream out = conn.getOutputStream();
4
5  String str = "";
6  int c = 1;
7  for (Point p : this.points) {
8      str += "P" + c + "x=" + p.getX() + "&" + "P" + c + "y=" + p.getY() + "&";
9      c++;
10 }
11
12 out.write(str.getBytes());
13 out.close();
14
15 InputStreamReader input = new InputStreamReader(conn.getInputStream());
16 input.close();
```

Um mit einem PHP-Skript zu kommunizieren, benötigt es zunächst einer Verbindung zu diesem. Hierzu kann unter Verwendung der Methode ***getDocumentBase()*** die URL zu jener Datei erhalten werden, welche das Applet mit Hilfe des oben erwähnten `applet`-Tags darstellt. Im aktuellen Fall handelt es sich also um das zugrundeliegende PHP-Skript. Eine Verbindung zu diesem lässt sich nun mit Hilfe der Methode ***openConnection()*** aufbauen und das dabei entstandene Objekt in einer entsprechenden Variablen speichern. Anschließend richtet Zeile 2 die Verbindung für eine Ausgabe ein, bevor der entsprechende Ausgabestrom der Variablen `out` zugewiesen wird. Auf Basis eine solchen Stroms lassen sich nun Informationen an das PHP-Skript senden, welches mit Hilfe des Arrays `$_POST` darauf zugreifen kann. Zunächst muss allerdings, wie in den Zeilen 5 bis 10 gezeigt, die Liste aller Diagrammpositionen durchlaufen und die erhaltenen Daten in einem ***String***-Objekt gespeichert werden. Hierbei lassen sich Variablen und deren Werte mit Hilfe der Notation `variable=value` angeben. Weiterhin ist es erforderlich, zwei unterschiedliche Werte durch ein Et-Zeichen (&) voneinander zu trennen. Eine Menge von drei Punkten kann deshalb wie folgt aussehen:

```
P1x=0.454&P1y=0.278&P2x=0.239&P2y=0.754&P3x=0.630&P3y=0.894&
```

Nach dem Erzeugen einer solchen Zeichenkette, kann diese nun in den zuvor gespeicherten Ausgabestrom geschrieben und das entsprechende Objekt mit der Methode ***close()*** geschlossen werden (Zeile 12, 13). Hierdurch wurden allerdings noch keine Daten verschickt. Dies geschieht erst nach dem Zugriff auf den entsprechenden Eingabestrom (Zeile 15, 16). Dabei sendet das Applet eine versteckte POST-Anfrage an den Server, um einen solchen Strom zu öffnen und hängt sämtliche Informationen aus dem Ausgabestrom an. Auf diese Weise erhält das PHP-Skript alle erzeugten Diagrammpositionen und speichert sie daraufhin im Array `$_SESSION`. Im Anschluss hieran kann das Applet geschlossen werden.

4.5.4. Benutzerdaten im XML-Format

Wie bereits in Abschnitt 4.1 erwähnt, werden die Benutzerdaten in einem eigens dafür entwickelten XML-Format gespeichert. Dieses enthält sämtliche Informationen, welche zur Erzeugung eines Auftrags nötig sind und wird nach dem Absenden eines solchen im Benutzerordner erzeugt. Hierfür steht eine entsprechende PHP-Klasse zur Verfügung, die auf eine bereits vorgefertigte Bibliothek der PHP-Installation zurückgreift. In der oben erwähnten PHP-Klasse wurde eine Methode mit dem Namen ***writeFile()*** implementiert. Durch deren Aufruf bei einem zuvor erzeugten Objekt lässt sich nun eine XML-Datei mit der nachfolgenden Struktur erzeugen. Hierbei können aus Platzgründen allerdings nicht alle Einstellungen gezeigt werden, was an den gegebenen Stellen durch drei Punkte signalisiert wird.

```
1  <settings>
2    <user_id>0809231128012109</user_id>
3    <user_dir source="/home/lutz/public_html/0809231128012109/"/>
4    <image source="0809231128012109_map.png" width="2084" height="2909"
         alpha_mapping="1"/>
5    <summary diagram_type="3" header="Bundesergebnis"/>
6    <point_list data_source="0809301815195684_data.csv" diagram_type="1">
7      <point posX="0.501" posY="0.0441"/> ...
8    </point_list>
9    <anim_settings> <anim_mode>1</anim_mode> ... </anim_settings>
10   <app_settings> <material_mode>0</material_mode> ... </app_settings>
11   <render_settings> <video_width>800</video_width> ... </render_settings>
12   <title_settings> <title>Bundestagswahlen</title> ... </title_settings>
13   <email address="email@domian.de"/>
14   <publication allowed="0"/>
15 </settings>
```

Zunächst besteht eine solche XML-Datei aus einem globalen Element mit der Bezeichnung *settings*. Dieses beinhaltet sowohl alle vom Benutzer vorgegebenen Einstellungen als auch diverse Standardwerte, welche beim Erstellen des Auftrags nicht beachtet wurden. So kann der Benutzer beispielsweise die Animationszeiten ändern, muss dies aber nicht. Im zweiten Fall treten dann die jeweiligen Standardwerte auf. Die Gesamtheit aller Einstellungen wird nun strukturiert und innerhalb des oben genannten Elements angeordnet. Hierbei muss zwischen Elementen, welche Informationen als Attribute darstellen und solchen, die weitere Unterelemente beinhalten unterschieden werden. Erstere speichern hauptsächlich Einstellungen eines bestimmten Sachverhalts, wie beispielsweise die Auftragsnummer oder sämtliche Informationen zur verwendeten Bilddatei. Elemente des zweiten Typs lassen sich als eine Art Sammlung verschiedener Einstellungen bezüglich eines Überbegriffs darstellen. Ein Beispiel hierfür wäre somit das Element *title_settings*, welches sämtliche Informationen über den Titelbildschirm des Präsentationsvideos zusammenfasst. Diese werden innerhalb eines eigenen Unterelements gespeichert, wie am Beispiel des Titeltextes in Zeile 12 zu sehen.
Eine Ausnahme stellt hierbei allerdings das Element *point_list* dar, welches sowohl Informationen mit Hilfe von Attributen speichert, als auch zusätzliche Unterelemente verwendet. Dieses kapselt die gesamten Daten, die zum Erzeugen der Diagramme notwendig sind. Während per Attribut die Bezeichnung der jeweiligen CSV-Datei und der zu verwendende Diagrammtyp gespeichert wird, lässt mit Hilfe der Unterelemente auf die relativen Positionen der einzelnen Diagramme zurückgreifen. Auf Basis dieser und der Größe der Karte kann Blender bzw. das Python-Skript beim Ausführen des jeweiligen Auftrags die entsprechenden absoluten Positionen berechnen.
Nachdem sich nun ein solches XML-Dokument erzeugen lässt, muss es an verschiedenen Stellen auch wieder eingelesen werden, um die darin enthaltenen Daten verarbeiten zu kön-

nen. Aus diesem Grund existieren zwei weitere Klassen. Eine in der Programmiersprache Python und die andere in PHP. Erstere wird in Verbindung mit dem Python-Skript zur Generierung der dreidimensionalen Szene in Blender verwendet. Diese basiert auf einem bereits vorhandenen Modul zum Öffnen und Einlesen von XML-Dokumenten. Die hierdurch erhaltenen Daten werden daraufhin in den dafür vorgesehenen Klassen gespeichert, welche in Abschnitt 4.3 vorgestellt wurden. Im Gegensatz hierzu lässt sich die PHP-Klasse zum Einlesen von XML-Dateien dazu verwenden, dem Benutzer die Möglichkeit zu geben, diverse Änderungen an den zuvor gemachten Einstellungen vorzunehmen. Auch dabei wird eine bereits vorgefertigte Bibliothek als Grundlage zum Öffnen und Durchsuchen des XML-Dokuments verwendet. Auf dieser Basis kann nun ein Objekt der eigens hierfür entwickelten PHP-Klasse die unterschiedlichen Einstellungen Schritt für Schritt einlesen und an die entsprechenden Teilformulare weiterreichen. Abschließend soll noch erwähnt werden, dass aufgrund der Flexibilität des XML-Formats und der Klassen zum Erzeugen und Einlesen solcher Dateien sich recht einfach zusätzliche Daten anzufügen lassen.

4.6. Probleme bei der Entwicklung

Durch die Kombination von verschiedenen Techniken und mehreren Programmiersprachen kam es während der Implementierung des ChartFlight Services zu unterschiedlichen Problemen, von welchen dieser Abschnitt die schwerwiegendsten behandelt. Hierbei werden sowohl Probleme betrachtet, die im Verlauf der Entwicklung zu lösen waren als auch solche, die noch offen sind bzw. bei welchen es einer entsprechenden Alternative bedurfte.
Eines der größten Probleme stellte die Verwendung von Blender im Hintergrund dar, welche sich mit Hilfe des Kommandozeilenparameters `-b` erreichen lässt. Hierdurch öffnet sich Blender ohne eine grafische Oberfläche und kann somit zum Rendern unterschiedlicher Szenen verwendet werden. Diese Möglichkeit schien zunächst die optimale für die Kombination eines Webservices mit Blender, welcher auf Anfrage Bilder und Videos rendern soll. Allerdings traten hierbei mehrere Probleme auf, die eine Verwendung des oben genannten Parameters nicht zuließen. Dies betrifft sowohl das zugrundeliegende Python-Skript als auch diverse Einstellungen, die lediglich über die Oberfläche von Blender zu erreichen sind. Die Möglichkeit Python-Skripte auch im Hintergrundmodus starten zu können, bestätigte zunächst den Ansatz diesen tatsächlich zu verwenden. Nach einigen Testläufen stellte sich allerdings heraus, dass beim jeweiligen Rendervorgang Blender mit der Angabe eines Segmentation Faults regelmäßig abstürzte. Bei einer anschließenden Betrachtung der gesicherten Projektdatei tauchten zudem weitere Probleme auf. Unterschiedliche Bestandteile der generierten Szene wurden nicht korrekt erzeugt, was beispielsweise die diversen Beschriftungstexte betraf. Selbst nach längeren Recherchen konnten hierfür bisher keine Lösung gefunden werden. Ein weiterer Nachteil, welcher beim Verwenden des Hintergrundmodus von Blender auftritt,

betrifft das Laden sämtlicher Standardeinstellungen. Diese lassen sich bekanntlich den eigenen Bedürfnissen anpassen und daraufhin als Standard speichern. Hierzu wird intern die Datei `.B.blend` im Ordner ***.blender*** verwendet. Allerdings wird die genannte Datei nicht geladen, wenn Blender im Hintergrund startet. Dies hat zur Folge, dass die ursprünglichen, im Quellcode von Blender vorgegebenen Einstellungen als Basis dienen. Prinzipiell sollte es dabei keine Nachteile geben, da sich eine Vielzahl von Einstellungen auch mit Hilfe des Python-Skripts setzen lassen. Betrachtet man aber die Blender Python API ein wenig genauer, so lässt sich keine Möglichkeit finden, speziellere Videoeinstellungen zu manipulieren. Hierunter zählen beispielsweise solche zum Konfigurieren des Videocodecs. Somit müssen all diese Einstellungen auf Basis der Datei `.B.blend` getan werden, was allerdings erneut die Verwendung des Hintergrundmodus ausschließen würde. Aus diesem Grund lässt sich Blender zum momentanen Zeitpunkt ausschließlich mit grafischer Oberfläche starten, wozu das Betriebssystem die entsprechende Möglichkeit beispielsweise in Form eines X-Servers unter Linux bereitstellen muss.

Ein anderes Problem, welches unter Blender auftritt, beschäftigt sich mit Transparenzen unter Verwendung des Ray-Tracing Verfahrens. Hierdurch ist es möglich, diverse Glaseffekte zu erzielen und entsprechend zu rendern. Allerdings lässt sich zu diesem Zweck eine bestimmte Rekursiontiefe für das oben genannte Verfahren einstellen, welche angibt, wie viele sich überdeckende transparente Flächen maximal hintereinander auftauchen dürfen. Wird hierbei die angegebene Anzahl überschritten, so kann es beim Rendern zu Grafikfehlern kommen, da das Ray-Tracing Verfahren nicht alle Flächen als transparent behandelt. Das eigentliche Problem dabei ist nun, dass sich in Blender maximal zehn solcher Flächen einstellen lassen, es in unterschiedlichen Situationen aber zu mehr als diesen kommen kann. Somit muss auch hier auf eine solche Funktionalität verzichtet werden.

Das letzte größere Problem des Backends ließ sich allerdings recht einfach lösen. Dabei handelte es sich um die Frage, wie es möglich ist, einem Benutzer, der Änderungen an einem Auftrag vorgenommen hat, dessen Ergebnisse nicht zu lange vorzuenthalten. Da anfangs lediglich eine einzige Instanz von Blender zur gleichen Zeit erlaubt war, konnte es somit vorkommen, dass diese bereits einen Auftrag ausführte und die vom Benutzer getätigten Änderungen zwar in die Datenbank eingetragen, allerdings nicht sofort verarbeitet wurden. Ein solcher Nachteil ließ sich aber unter Verwendung von zwei unterschiedlichen Instanzen von Blender beheben. Während eine lediglich für das Rendern von Vorschaubildern zuständig ist, erzeugt die andere das finale Präsentationsvideo. Genauere Informationen darüber befinden sich in den vorangegangenen Abschnitten.

Im Gegensatz zum Backend traten beim JobListener verhältnismäßig wenige Probleme auf. Die größte Schwierigkeit bereitete das Nachrüsten der Bibliotheken zum Versenden von E-Mails und für den Zugriff auf eine SQLite-Datenbank. Erstere ließ sich dabei verhältnismäßig einfach über ein Zusatzpaket der Firma *Sun Microsystems* hinzufügen. Zum Nachrüsten der entsprechenden Datenbanktreiber musste allerdings auf die Bibliothek eines Drittanbieters

zurückgegriffen werden, welche aber trotzdem voll funktionsfähig ist.
Ein kleineres Problem, welches beim Ausführen von Blender mit Hilfe des JobListeners unter Linux auftauchte, war das fehlende Et-Zeichen (&) am Ende des dafür vorgesehenen Kommandos. Hierbei wurde ein Auftrag, welcher bereits sofort nach dem Starten des JobListener in der Datenbank verfügbar war, vollkommen korrekt bearbeitet. Für solche, die allerdings erst später hinzukamen oder erst an zweiter Stelle in der Datenbank standen, ließ sich Blender nicht mehr Starten. Durch Hinzufügen des Et-Zeichens am Ende des Kommandos zum Ausführen von Blender konnte dieser Fehler aber behoben werden. Unter Windows traten hierbei allerdings keine Probleme auf.
Die Hauptschwierigkeit bei der Entwicklung der Benutzerschnittstelle war das Layout und die Funktionalität der einzelnen Formularseiten. Diese sollten einerseits den Benutzer nicht überfordern, andererseits aber auch genügend Informationen bezüglich den diversen Einstellungsmöglichkeiten geben. Aufgrund deren Vielzahl musste allerdings zusätzlich die Oberfläche in zwei Modi eingeteilt werden, einen für normale Benutzer und eine Art Expertenmodus mit spezielleren Optionen. Das Problem hierbei war die Auswahl der Einstellungen für den jeweiligen Modus. Auch hierbei sollte weder der Benutzer überfordert werden noch zu wenige Konfigurationsmöglichkeiten zur Verfügung stehen. Dieses Verhältnis wurde allerdings nach eigenem Ermessen gewählt und basiert nicht auf einem Testlauf mit einer größeren Anzahl an unterschiedlichen Benutzern.
Wie auch schon beim JobListener, stellten die SQLite-Treiber auch hier zunächst ein Problem dar. Diese waren zwar verfügbar, allerdings konnten lediglich Datenbanken der Version 2 und nicht solche der Version 3 geöffnet werden. In Python war genau die umgekehrte Situation der Fall. Hier war es nicht möglich Datenbanken der Version 2 zu öffnen. Unter Windows ließ sich dieses Problem recht einfach beheben, da die Treiber mit der PHP-Installation schon mitgeliefert und lediglich aktiviert werden mussten. Unter Linux hingegen bedurfte es zunächst einer Installation eines SQLite3 Pakets für PHP, welches die vollständige Konfiguration der Treiber automatisch durchführte. Danach war in beiden Systemen der Zugriff auf die Datenbank möglich.
Selbstverständlich traten noch eine ganze Reihe anderer Probleme auf, welche entweder nicht so schwerwiegend waren, um hier genannt zu werden oder deren Erklärung zu komplex ist und zu vieler Voraussetzungen bedarf.

4.7. Installation und Konfiguration

Im letzten Abschnitt dieses Kapitels soll erläutert werden, wie die Installation und Konfiguration des ChartFlight Services abläuft und welche Anforderungen dafür nötig sind. Ebenso müssen auch auf Benutzerseite diverse Voraussetzungen erfüllt sein. Bevor wir die Installation nun genauer betrachten, sollen zunächst sämtliche Anforderungen aufgelistet werden.

Anforderungen für den Server
- Blender 2.46 + Video Codecs
- Python 2.5
- Java RE Version 6
- Apache 2
- PHP 5
- SQLite 3 für PHP
- E-Mail-Server (bspw. Postfix)

Anforderungen für den Benutzer
- Java RE Version 6 (empfohlen)
- JavaScript (aktiv)
- Videoplayer + Codecs

Wie oben zu sehen, sind auf Serverseite deutlich mehr Installationen vonnöten, da dort der Hauptteil des Services abläuft. Unter Linux können allerdings sämtliche Anforderungen mit Hilfe der Paketverwaltung komfortabel installiert werden, wobei es wichtig ist, auf die angegebenen Versionen zu achten. Für Windows steht auf der Webseite des jeweiligen Herstellers eine entsprechende Installationsdatei zur Verfügung. Auf dem Computer des Benutzers müssen lediglich das Applet und das finale Präsentationsvideo ausgeführt werden. Während für ersteres das *Java Runtime Environment* installiert sein sollte, benötigt man zum Abspielen des Videos einen gewöhnlichen Videoplayer mit den entsprechenden Codecs. Zudem sollte im Browser des Benutzers *JavaScript* aktiviert sein, da sonst unterschiedliche Teile der Webseite nicht funktionieren.
Sind nun alle Anforderungen verfügbar, so lässt sich daraufhin der ChartFlight Service installieren. Da dieser hauptsächlich unter Linux betrieben werden soll, bezieht sich die nachfolgende Anleitung lediglich hierauf. Allerdings funktioniert die Installation unter Windows auf ähnliche Art und Weise.

Installation

Unter Linux werden sämtliche Dateien in dem Home-Verzeichnis des Administrator bzw. eines anderen Benutzers abgelegt. Hierbei ist zwischen drei verschiedenen Ordnern zu unterscheiden. Zur Installation des Backends wird lediglich das Verzeichnis ***.blender*** benötigt. Dieses sollte bereits nach der Installation von Blender für jeden Benutzer des Systems in dessen Home-Verzeichnis vorhanden sein, wobei es üblicherweise unsichtbar ist. Darin befindet sich ein weiterer Ordner mit der Bezeichnung ***scripts***, in welchen alle Python-Skripte samt der Konfigurationsdatei `config.ini` kopiert werden. Zudem sollte sich in einem der beiden Verzeichnisse die Startdatei des Backends mit dem Namen `start.blend` befinden. Vorzugsweise wird dafür ersteres verwendet. Wurde bei der Installation von Blender die oben erklärte Verzeichnisstruktur nicht erstellt, so lässt sich, durch einfaches Kopieren des entsprechenden Ordners von der beigelegten CD in das jeweilige Home-Verzeichnis, das Backend installieren. Der selbe Vorgang ist für die Installation des JobListeners erforderlich. Dieser befindet sich im gleichnamigen Ordner auf der CD und kann an einen beliebigen Ort innerhalb des Home-Verzeichnisses kopiert werden. Abbildung 4.4 zeigt dabei lediglich eine

Empfehlung. Zu guter Letzt gilt es die Installation der Benutzeroberfläche in Form der Webseite durchzuführen. Für diesen Vorgang ist das Kopieren des Ordners ***public_html*** von der CD in das Home-Verzeichnis erforderlich. Sollte dort bereits ein solcher existieren, so müssen sämtliche Dateien und Unterordner darin platziert werden. Hierfür lässt sich ebenso ein zusätzliches Unterverzeichnis anlegen, sodass man die gesamte Benutzeroberfläche auch an der Stelle `/home/<user_name>/public_html/chartflight/` platzieren könnte. Selbstverständlich muss dabei die Zeichenkette `<user_name>` durch den jeweiligen Namen des Linux-Users ersetzt werden. Diese Schreibweise taucht ebenfalls im nachfolgenden Unterabschnitt zur Konfiguration der unterschiedlichen Komponenten auf.
Um die Installation der Webseite abzuschließen, müssen lediglich noch die entsprechenden Benutzerrechte gesetzt werden. Dabei ist es wichtig, dass sowohl der Ordner ***users*** als auch die darin enthaltene SQLite-Datenbank für alle Benutzer des Systems zugreifbar und vor allem modifizierbar sind. Dies ist deshalb erforderlich, da die Datenbank als zentraler Anlaufpunkt des Services gilt und somit von verschiedenen Komponenten manipuliert wird. Sollten an anderer Stelle Probleme mit der Webseite oder den übrigen Bestandteilen des ChartFlight Services auftauchen, so ist es empfehlenswert zunächst die entsprechenden Rechte zu überprüfen. Abbildung 4.4 stellt abschließend den Installationsvorgang aller drei Komponenten grafisch dar.

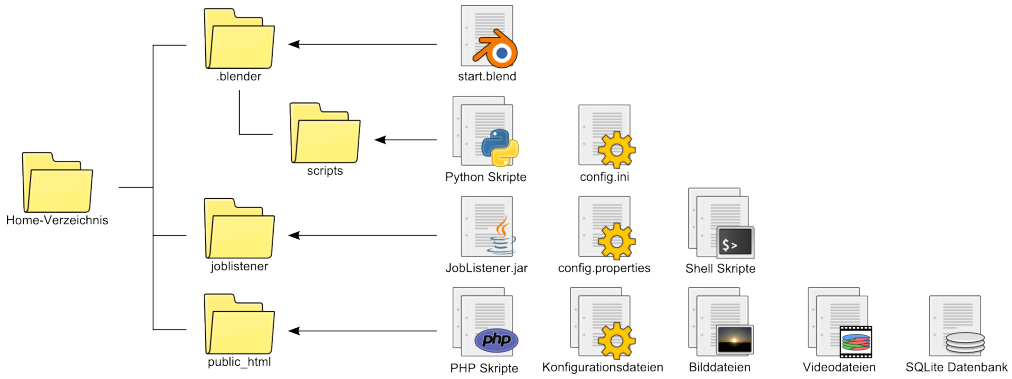

Abbildung 4.4.: Installationsvorgang des ChartFlight Services

Konfiguration

Die Konfiguration des ChartFlight Services läuft ebenfalls komponentenweise ab. Zunächst werden sämtliche Einstellungen bezüglich Blender und der verwendeten Python-Skripte gemacht, wofür die Datei *config.ini* zur Verfügung steht. Hierbei handelt es sich um eine gewöhnliche Konfigurationsdatei, welche Einstellungen in fünf unterschiedlichen Kategorien enthält. Für das Funktionieren des Services ist die erste Kategorie besonders wichtig, da sie diverse Pfadangaben enthält. Das nachfolgende Listing zeigt deren Einträge im Detail.

```
1 [Paths]
2 ApplicationDir=/home/<user_name>/public_html/
3 BaseURL=http://www.chartflight.de/
4 UsersDir=users/
5 UserQueue=user_queue.db
6 EmailDir=email/
7 EmailData=email_data.ini
8 LogsDir=logs/
9 LogFile=blender.log
```

Bei der Konfiguration des Backends gilt es zunächst den Pfad zum Hauptverzeichnis des ChartFlight Services anzugeben (Zeile 2). Hierbei handelt es sich bei einer standardmäßigen Installation um den Ordner ***public_html*** im Home-Verzeichnis. Wurde wie oben beschrieben allerdings ein Unterordner erzeugt und sämtliche Dateien der Webseite darin platziert, so muss selbstverständlich der Pfad zu diesem in die Konfigurationsdatei eingetragen werden. Bei der Angabe eines korrekten Pfades können die Python-Skripte anschließend auf das entsprechende Verzeichnis und alle darin enthaltenen Unterordner zugreifen.
Der zweite Eintrag in der Konfigurationsdatei erhält eine gültige URL zum Hauptverzeichnis des Services. Mit dieser ist es nun möglich innerhalb des Python-Skripts die Links zu den jeweiligen Präsentationsseiten zu generieren. Obiges Listing gibt dafür die fiktive Adresse `www.chartflight.de` an.
Wurde die Verzeichnisstruktur im Ordner ***public_html*** beibehalten, so lassen sich ohne weitere Bearbeitung die übrigen Einträge aus den Zeilen 4 bis 9 übernehmen. Anderenfalls müssen diese entsprechend angepasst werden.
Wie bereits erwähnt, befinden sich innerhalb der Konfigurationsdatei vier weitere Kategorien mit unterschiedlichen Einstellungen, wovon allerdings nur die drei letzten vollständig im nachfolgenden Listing dargestellt werden.

```
1  [Render]
2  Shadow=0
3  ...
4
5  [Appearance]
6  Specularity=0
7
8  [Lighting]
9  LightingMode=1
10 LightEnergy=0.55
11
12 [Misc]
13 SaveBlendFile=1
```

Wie der Kategorie **Render** zu entnehmen, beschäftigt sich diese mit diversen Einstellungen zur Bilderzeugung. Hierbei kann beispielsweise die Berechnung von Schatten oder die Qualität des Anti-Aliasings verändert werden. Alle darin verfügbaren Einträge entsprechen jeweils einer Option in Blender, weshalb es an dieser Stelle nicht notwendig ist darauf einzeln einzugehen. Ähnlich verhält es sich mit der Kategorie **Appearance**, in welcher sich mit dem Eintrag ***Specularity*** angeben lässt, ob bei der Beleuchtung der Szene die Berechnung von Glanzpunkten eine Rolle spielen soll. Mit Hilfe der Kategorie **Lighting** können weitere Einstellungen bezüglich der Beleuchtung einer Szene vorgenommen werden. Hierbei lässt sich deren Modus und die allgemeine Lichtstärke abändern. Ersteres besitzt allerdings einen experimentellen Status, weshalb empfohlen wird, die vorgegebene Einstellung zu übernehmen. Die letzte Option der Kategorie **Misc** gibt schließlich an, ob das Python-Skript für den jeweiligen Auftrag nach dessen finalen Rendervorgang eine Projektdatei (.blend) speichern soll. Diese enthält alle Informationen der aktuellen Szene und lässt sich unter anderem zum Auffinden von Fehlern verwenden.

Wurden sämtliche Einstellungen für das Backend gemacht, so lässt sich mit Hilfe einer weiteren Konfigurationdatei der gleiche Vorgang für den JobListener durchführen. Diese heißt `config.properties` und befindet sich im selben Verzeichnis, wie die Jar-Datei des Java-Programms. Auch darin existieren mehrere Kategorien von Einstellungen, wobei die erste wiederum für die Angabe verschiedener Pfade zuständig ist. Hierbei sind unter anderem die bereits oben vorgestellten Einstellungen zu machen, welche aber nicht vollständig im nachfolgenden Listing auftauchen. Zudem enthält die Datei Informationen zum Starten von Blender und den entsprechenden Python-Skripten. Diese befinden sich in der Zeilen 2 bis 4.

```
1 # Paths
2 BlenderPath=/usr/bin/
3 RunBlender=blender
4 BlendFile=.blender/start.blend
5 ApplicationDir=/home/<user_name>/public_html/
6 ...
```

Der erste dieser Einträge mit der Bezeichnung ***BlenderPath*** gibt den Pfad zu dem Verzeichnis an, in welchem sich die Datei zum Starten von Blender befindet. Deren spezifischer Dateiname ist allerdings erst nach einem Zugriff auf die Option ***RunBlender*** verfügbar. Das obige Beispiel zeigt somit auf die Datei `blender` im Verzeichnis `/usr/bin/`. Bei der zweiten Einstellung muss aber nicht zwingend die tatsächliche Startdatei angegebenen werden, sondern es besteht ebenso die Möglichkeit ein Shell-Skript zu verwenden. Dieses könnte beispielsweise Blender mit reduzierter Fenstergröße öffnen, um den Overhead beim Starten der grafischen Oberfläche zu minimieren. Eine weitere Aufgabe eines solchen Shell Skripts wäre die Ausgabe von Blender auf der grafischen Oberfläche des Linux Betriebssystems mit Hilfe

der Umgebungsvariablen `DISPLAY` und dem Wert *:0*. Dies würde sicherstellen, dass Blender stets korrekt geöffnet wird. Unter dem Punkt ***BlendFile*** lässt sich die zu öffnende Projektdatei angeben, welche die entsprechenden Scriptlinks zum Starten der Python-Skripte enthält. Bei einer normalen Installation handelt es sich hierbei um die Datei `start.blend`, deren absoluter Pfad eingetragen werden sollte. Befindet sie sich allerdings im Ordner ***.blender***, so reicht es aus diesen als oberstes Verzeichnis anzugeben, wie in obigem Listing zu sehen.

Die zweite Kategorie der Konfigurationsdatei beschäftigt sich mit den Zeitintervallen der unterschiedlichen Aufgaben des JobListeners. Diese können in Form von Fließkommazahlen angegeben werden und bezeichnen, je nach Option, eine bestimmte Anzahl an Minuten oder Stunden. Der nachfolgende Ausschnitt der Konfigurationsdatei zeigt sämtliche verfügbaren Einstellungen dieser Kategorie.

```
1 # Time intervals
2 CheckingInterval=1.0
3 CleanUpInterval=2.0
4 DeleteCancelledJob=10.0
5 DeletePreviewJob=168.0
6 DeleteDoneJob=168.0
7 UserNotification=24.0
```

Wie bereits oben erwähnt, werden bei den verschiedenen Optionen unterschiedliche Zeiteinheiten verwendet. Während sich mit den ersten beiden Einträgen ein Intervall in Minuten einstellen lässt, sind die übrigen für die Angabe von Stunden ausgelegt. Dies sollte bei der Manipulation der einzelnen Werte beachtet werden. Wirft man einen genaueren Blick auf die Namen der diversen Einträge, so beschreiben sie bereits deren jeweilige Aufgabe. So gibt ***CheckingInterval*** an, in welchen Abständen die Datenbank auf neue Aufträge überprüft werden soll. Auf die gleiche Weise lässt sich mit Hilfe des zweiten Eintrags der Zeitraum zum Bereinigen der Datenbank und des Dateisystems festlegen (Zeile 3). Bei diesen beiden Einstellungen sollte man allerdings auf Werte kleiner als 1.0 verzichten, da solche Intervalle aufgrund mehrerer Datenbankzugriffe nicht immer eingehalten werden können. Die nächste Option mit der Bezeichnung ***DeleteCancelledJob*** gibt die Zahl der Stunden an, welche vergehen müssen um einen Auftrag als abgebrochen anzusehen. Hierbei handelt es sich lediglich um solche Aufträge, die vom Benutzer während der Eingabe sämtlicher Daten über die diversen Formularseiten aufgegeben wurden und zudem keinen Eintrag in der Datenbank besitzen. Beim Verändern dieser Option ist allerdings zu beachten, dass man den eingestellten Wert nicht zu klein wählt, da ein Benutzer sonst zu wenig Zeit beim Erstellen eines Auftrags besitzt. Im Gegensatz dazu beziehen sich die nächsten beiden Optionen mit den Namen ***DeletePreviewJob*** und ***DeleteDoneJob*** auf solche Aufträge, die bereits in die Datenbank aufgenommen wurden. Während erstere den Zeitraum definiert, nach welchem ein Eintrag mit dem Status ***preview*** aus der Datenbank gelöscht wird, gibt letzteres dies

für einen bereits abgeschlossenen Auftrag an. Da allerdings keine Datenbankeinträge ohne vorherige Benachrichtigung des jeweiligen Benutzers entfernt werden, muss der JobListener wissen, wann er dies geschehen soll. Hierfür steht die Option ***UserNotification*** zur Verfügung, welche angibt, wie viele Stunden vor dem Ablaufen eines Auftrags eine entsprechende E-Mail versendet wird. Weitere Details zu diesen Vorgängen befinden sich allerdings in Abschnitt 4.4.
Die letzte Kategorie der Konfigurationsdatei beschäftigt sich mit einer Funktion, die lediglich in Linux verfügbar ist und auch nur dort aktiviert werden sollte. Hierbei handelt es sich um die Überwachung einer laufenden Instanz von Blender. Diese lässt sich mit Hilfe der Option ***ObserveBlender*** und einer Angabe der Werte 0 oder 1 entsprechend aus- bzw. eingeschalten. Da für die Überwachung von Blender ein spezielles Shell-Skript verwendet wird, existiert dafür ein zusätzlicher Eintrag mit dem Namen ***BlenderMonitor***. Wie schon in Abschnitt 4.4.3 erklärt, müssen für die Verwendung solcher Shell-Skripte die darin enthaltenen Pfade entsprechend der jeweiligen Verzeichnisstruktur angepasst werden.

Genau wie die beiden vorgestellten Komponenten, lässt sich ebenso die Benutzerschnittstelle entsprechend konfigurieren. Hierzu wird die Datei `settings.php` im gleichnamigen Ordner verwendet, welche sämtliche benötigten Einstellungen beinhaltet. Darunter befinden sich sowohl Variablen zur Konfigurationen diverser technischer Aspekte als auch solche, die sich auf das Layout der Webseite beziehen. Nachfolgend soll allerdings lediglich ersteres dargestellt und behandelt werden.

```
1  // paths
2  $application_dir = "/home/<user_name>/public_html/";
3  $base_url = "http://www.chartflight.de/";
4  $users_dir = "users/";
5  $user_queue = "user_queue.db";
6  $email_dir = "email/";
7  $email_data = "email_data.ini";
8
9  // misc
10 $max_job_count = 5;
11 $keep_session = 12.0; // in hours
12 $debug = TRUE;
```

Im Gegensatz zu den oben vorgestellten Konfigurationsdateien, handelt es sich bei dieser um ein einfaches PHP-Skript, welches eine Reihe von Variablen enthält. Somit muss es nicht über spezielle Klassen eingelesen werden, sondern lässt mit Hilfe der Funktion ***require_once()*** in das jeweilige PHP-Skript einblenden. Daraufhin stehen dort die oben gelisteten Variablen zur Verfügung. Hierbei ist zu erkennen, dass alle Einstellungen der Kategorie **paths** bereits in den beiden anderen Konfigurationsdateien vorkamen und entsprechend erläutert wurden. Da

es sich bei dieser Datei allerdings um ein PHP-Skript handelt, sollten zum einen sämtliche Zeichenketten mit doppelten Anführungszeichen als solche markiert und zum anderen nach jeder Anweisung ein Semikolon gesetzt werden.
Die Optionen der Kategorie **misc** hingegen sind lediglich für das Frontend interessant und tauchen deshalb auch in keiner der beiden anderen Konfigurationsdateien auf. Hierunter befindet sich zunächst die Variable `$max_job_count`, welche die maximale Anzahl an Aufträgen in der Datenbank definiert. Auf deren Basis wird beispielsweise bestimmt, ob ein Benutzer einen neuen Auftrag starten kann oder nicht. Wurde dies erlaubt, so gibt `$keep_session` an, wie lange das Erstellen eines solchen maximal dauern darf. Ein Wert von 12.0 heißt somit, dass sämtliche vom Benutzer getätigten Einstellungen entweder nach dem Absenden des Auftrags oder nach spätestens zwölf Stunden aus dem Array `$_SESSION` und somit auch aus den Teilformularen gelöscht werden. Auch hierbei sollte man auf zu geringe Werte verzichten. Die letzte Variable mit dem Namen `$debug` ist lediglich zum Auffinden von Fehlern interessant. Enthält diese den Wert ***TRUE***, so listet das jeweilige PHP-Skript unterhalb des Teilformulars sämtliche Einträge des Arrays `$_SESSION` auf.

Abschließend soll die Konfigurationsdatei `email_data.ini` vorgestellt werden. Diese ist für alle drei Komponenten des Services wichtig und befindet sich bei einer normalen Installation im Ordner ***email*** des Hauptverzeichnisses. Wie deren Name bereits sagt, beinhaltet sie sämtliche Einstellungen bezüglich des Versendens von E-Mails. Auch hierin werden die unterschiedlichen Einstellungen in drei Kategorien eingeteilt, was das nachfolgende Listing verdeutlicht.

```
1  [Basic]
2  User=chartflight
3  Password=123456
4  EmailAddress=chartflight@informatik.uni-trier.de
5
6  [Advanced]
7  SMTPServer=jupiter.uni-trier.de:25
8  UseTLS=1
9
10 [Templates]
11 Preview=preview_template.txt
12 ...
```

In der Kategorie **Basic** befinden sich sämtliche Zugangsdaten, die zum Versenden einer E-Mail benötigt werden. Hier lässt sich zunächst der Benutzer und das Passwort und anschließend unter dem letzten Punkt in Zeile 4 die entsprechende E-Mail-Adresse angeben. Will man allerdings den lokalen E-Mail-Server verwenden, so müssen die ersten beiden Optionen nach dem Gleichheitszeichen freigelassen werden. Anderenfalls ist zusätzlich eine entspre-

chende Anpassung der Einstellungen des Bereichs **Advanced** vonnöten. Hierin lassen sich der SMTP-Server sowie die Verschlüsselung von E-Mails mit Hilfe von TLS konfigurieren. Die erste Einstellung beinhaltet sowohl den jeweiligen Server als auch die durch einen Doppelpunkt abgetrennte Portnummer. Bei der Option ***UseTLS*** sind allerdings lediglich die Werte 0 oder 1 erlaubt. Die dritte Kategorie beschäftigt sich mit den Vorlagen für die unterschiedlichen E-Mails, was am Beispiel des Eintrags ***Preview*** gezeigt wird. Dieser gibt eine TXT-Datei im gleichen Verzeichnis an, welche einen Standardtext zur Erklärung der Situation bei den Vorschaubildern enthält. Ebenso stehen weitere Textdateien zur Verfügung, womit sämtliche Formen von E-Mails abgedeckt sind. So existieren unter anderem solche zur Benachrichtigung des Benutzer bei Fehlern oder im Falle eines abgeschlossenen Auftrags. All diese Vorlagen werden unter der Kategorie ***Templates*** angegeben.

5 Kapitel 5.

Evaluation

Dieses Kapitel soll über erste Erfahrungen mit dem ChartFlight Service berichten und unterschiedliche Messungen zur Dauer des Rendervorgangs und der jeweiligen Dateigrößen präsentieren. Hierbei werden in mehreren Abschnitten sowohl Vergleiche aller Auflösungen als auch sämtlicher Diagrammtypen dargestellt. Weiterhin betrachten wir die Renderdauer bei Verwendung des Transparenzeffekts und des Animationstyps. Im letzten Abschnitt sollen schließlich zwei vollständig gerenderte Präsentationsvideos gegenüber gestellt werden.
Um sämtliche Messungen miteinander vergleichen zu können, wurde bei allen Tests die folgende Hardware verwendet. Obwohl das System vollständig angegeben ist, sind für das Rendern von Videos unter Blender hauptsächlich die CPU und der Arbeitsspeicher die ausschlaggebenden Komponenten.

CPU : Intel Core 2 Duo 6400 @2.13GHz
MEM : 1024MB Corsair DDRII 667MHz
MAIN : ASUS P5B
GFX : Mad Moxx NVIDIA GeForce 7900 GTO 512MB Burstfire
OS : Windows XP Professional

Ebenso bedarf es einer einheitlichen Testszene, welche lediglich in den zu messenden Punkten variiert wird. Die hierin verwendeten Diagramme enthalten vier Zeilen und vier Spalten, woraus pro Position auf der Karte genau vier Teildiagramme mit jeweils vier dargestellten Werten entstehen. Als zugrundeliegende Grafik wurde die Deutschlandkarte aus Kapitel 3 herangezogen.
Da sich die Dauer des Rendervorgang nicht in allen Teilen des Videos unterscheidet - so ist es beispielsweise bei Renderings in der selben Auflösung unnötig, die Titelanimation zur vergleichen - werden sämtliche Messungen auf Basis eines einzelnen Diagramms durchgeführt.

Der hierfür verwendete Zeitraum beginnt bei dem Einzelbild, in welchem die entsprechende Legende das erste Mal auftaucht und endet nach der Bewegung der Kamera zum nächsten Diagramm. Allerdings kann eine solche Bewegung je nach der Entfernung zweier Punkte unterschiedlich lange andauern. Aus diesem Grund wurde eine Distanz von 5 Blender Units als Richtwert verwendet, welche die halbe Länge der kürzesten Seite der Karte darstellt. Auf Basis der oben genannten Voraussetzungen können nun die einzelnen Messungen durchgeführt werden.

5.1. Vergleich unterschiedlicher Auflösungen

Dieser Abschnitt beschreibt Messungen bezüglich der Dauer des Rendervorgangs und der Größen der daraus resultierenden Videodateien für die vier angebotenen Auflösungen. Hierzu wurde die oben beschriebene Testszene verwendet und je Auflösung ein Video von rund 1000 Einzelbildern gerendert. Die Ergebnisse dieser Messung zeigt Tabelle 5.1.

	Video 1	**Video 2**	**Video 3**	**Video 4**
Diagrammtyp	Kreisdiagramm	Kreisdiagramm	Kreisdiagramm	Kreisdiagramm
Auflösung	512 x 384	640 x 480	800 x 600	1024 x 768
Einzelbilder	1074	1074	1074	1074
Videodauer	42,96 s	42,96 s	42,96 s	42,96 s
Renderdauer	18:42 min	26:07 min	36:35 min	56:08 min
Hochrechnung	2:29:36 h	3:28:56 h	4:52:40 h	7:29:04 h
pro Einzelbild	1,045 s	1,459 s	2,044 s	3,136 s
Dateigröße	3.374 KB	4.380 KB	5.790 KB	7.814 KB

Tabelle 5.1.: Vergleich unterschiedlicher Auflösungen

Wie zu erwarten, steigt bei einer Erhöhung der Auflösung sowohl die Dauer des Rendervorgangs als auch die Dateigröße des jeweiligen Präsentationsvideos. Selbst bei 1074 gerenderten Einzelbildern, was einer Videolänge von knapp 43 Sekunden entspricht, ist hierbei schon ein deutlicher Unterschied zu erkennen. Rechnet man nun die Renderdauer auf ein Präsentationsvideo mit acht unterschiedlichen Diagrammen und Daten zu je vier Spalten und Zeilen hoch, so erhält man ohne Berücksichtigung der Titelanimation die in der siebten Zeile der obigen Tabelle dargestellten Ergebnisse. Weiterhin käme eine Verlängerung der Renderdauer hinzu, wenn zusätzlich ein Schlussdiagramm erstellt werden soll. Betrachtet man nun die Zeit zum Erzeugen eines einzelnen Bildes, so lässt sich erkennen, dass sich dieser Wert von der geringsten zur höchsten Auflösung verdreifacht hat. Auch bei den Dateigrößen zeichnet sich ein ähnlicher Verlauf ab. Allerdings liegt das Verhältnis vom ersten und zum letzten Video bei ca. 1:2,32 und nicht bei 1:3, wie es bei der Renderdauer der Fall war. Dies könnte eine Folge der Videokomprimierung sein, was aber durch eine größere Versuchreihe und den Vergleich mit Videos ohne Kompression bestätigt werden müsste. Unterstützend zu oben

gezeigter Tabelle visualisiert Abbildung 5.1 abschließend die Renderdauer der verschiedenen Auflösungen.

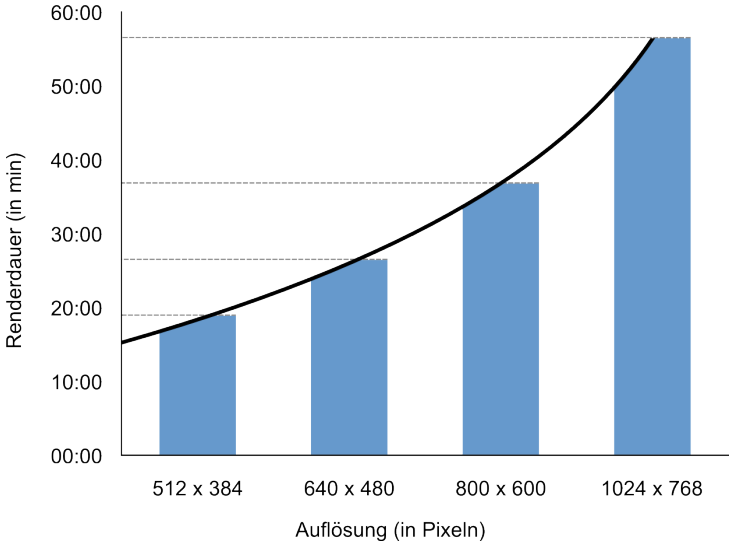

Abbildung 5.1.: Renderdauer bei unterschiedlichen Auflösungen

5.2. Vergleich unterschiedlicher Diagrammtypen

Da die einzelnen Diagrammtypen auf verschiedene Art und Weise generiert werden und somit unterschiedliche Geometrien beinhalten, stellte sich die Frage, ob es auch hier Auffälligkeiten bei der Dauer der Rendervorgangs gibt. Aus diesem Grund wurden auf Basis der oben vorgestellten Testszene diverse Messungen durchgeführt, welche nun eine feste Auflösung von 800 x 600 Pixeln verwendeten. Mit einer solchen Auflösung ist die Renderdauer insgesamt nicht zu kurz, um eventuelle Unterschiede feststellen zu können. Auch in diesem Fall wird das Ergebnis anhand der nachfolgenden Tabelle aufgelistet.

	Video 1	**Video 2**	**Video 3**	**Video 4**
Diagrammtyp	Kreisdiagramm	Ringdiagramm	Balkendiagramm	Liniendiagramm
Auflösung	800 x 600	800 x 600	800 x 600	800 x 600
Einzelbilder	1074	1074	1074	344
Videodauer	42,96 s	42,96 s	42,96 s	13,76 s
Renderdauer	36:35 min	35:55 min	37:47 min	11:02 min
pro Einzelbild	2,044 s	2,007 s	2,111 s	1,924 s
Dateigröße	5.790 KB	5.452 KB	5.760 KB	2.224 KB

Tabelle 5.2.: Vergleich unterschiedlicher Diagrammtypen

Im Gegensatz zum vorherigen Abschnitt treten bei der Betrachtung der Messungen aus Tabelle 5.2 kaum Unterschiede auf. So sind die Zeitwerte bei den ersten drei Diagrammtypen

nahezu identisch und auch die Größe der Videodateien weicht lediglich bei den Ringdiagrammen von den beiden anderen ab. Dies könnte daran liegen, dass die unterschiedlichen Einzelbilder bei den Ringdiagrammen geringere Unterschiede aufweisen. Während es bei den beiden anderen Typen zu häufigen Überdeckungen zwischen den einzelnen Teildiagrammen kommt, überlagert der nachfolgende Ring lediglich den vorderen Teil des aktuellen. Somit bleibt ein Großteil der Oberfläche eines Rings unverändert. Dies könnte den Komprimiervorgang des Videostroms positiv beeinflussen und eine geringere Dateigröße zur Folge haben. Allerdings müsste eine solche These in einer größeren Versuchsreihe bestätigt oder widerlegt werden.
Im Falle der Liniendiagramme macht sich allerdings ein deutlicher Unterschied bemerkbar. Diese benötigen für die Darstellung der gleichen Daten deutlich weniger Einzelbilder, womit sich auch die Videodauer verkürzt. Hierdurch lässt sich sowohl eine kürzere Renderdauer als auch geringere Dateigröße erzielen. Der Grund für dieses Verhalten liegt in der Art und Weise, wie die Daten in einem Liniendiagramm dargestellt werden. Eine einzelne Linie visualisiert, im Gegensatz zu einem Kreis-/Ringsegment oder einem Balken, nicht einen einzigen Wert, sondern sämtliche in einer Zeile enthaltenen. So lassen sich mit einem Liniendiagramm zweidimensionale Daten darstellen, während dies bei den übrigen nur unter Verwendung mehrerer Teildiagramme möglich ist. Da solche Teildiagramme bei der Animation einzeln zu bearbeiten sind, wird sowohl für das Rendern als auch für das spätere Abspielen des Videos mehr Zeit benötigt. Betrachtet man allerdings die Renderdauer pro Einzelbild, so lassen sich auch bei Liniendiagrammen nur minimale Unterschiede zu den übrigen Diagrammtypen feststellen.

5.3. Weitere Messungen

Dieser Abschnitt stellt weitere interessante Messungen bezüglich unterschiedlicher Möglichkeiten des ChartFlight Services dar. Hierbei soll zunächst ein Vergleich zwischen den verschiedenen Materialeinstellungen betrachtet werden. Genauer gesagt stehen dafür zum momentanen Zeitpunkt die Optionen ***Normal*** und ***Transparent*** zur Verfügung. Auch für diese wurden anhand der Testszene Messungen bezüglich Renderdauer und Dateigröße durchgeführt und in Tabelle 5.3 gegenübergestellt.
Eine weitere Frage beschäftigte sich damit, ob Unterschiede beim Rendern der einzelnen Animationstypen auftreten. Auch hier waren zwei verschiedene Optionen zu testen. So wurde zunächst in der Auflösung 800 x 600 Pixel die Testszene im Modus ***Einblenden*** gerendert und daraufhin der gleiche Vorgang mit der Einstellung ***Wachsend*** wiederholt. Die daraus resultierenden Ergebnisse werden neben denen für die Materialeinstellungen in Tabelle 5.3 dargestellt, welche auch die Werte für eine Kombination beider Messungen enthält.

	Video 1	Video 2	Video 3	Video 4
Diagrammtyp	Kreisdiagramm	Kreisdiagramm	Kreisdiagramm	Kreisdiagramm
Material	Normal	Transparent	Normal	Transparent
Animationstyp	Einblenden	Einblenden	Wachsend	Wachsend
Auflösung	800 x 600	800 x 600	800 x 600	800 x 600
Einzelbilder	1074	1074	1074	1074
Videodauer	42,96 s	42,96 s	42,96 s	42,96 s
Renderdauer	36:35 min	39:56 min	35:38 min	39:43 min
Hochrechnung	4:52:40 h	5:19:28 h	4:45:04 h	5:17:44 h
pro Einzelbild	2,044 s	2,231 s	1,991 s	2,219 s
Dateigröße	5.790 KB	5.718 KB	5.852 KB	5.758 KB

Tabelle 5.3.: Ergebnisse der Materialeinstellungen und des Animationstyps

Wie in dieser Tabelle zu sehen, treten lediglich bei der ersten Messung deutliche Unterschiede auf. Hier dauerte der Rendervorgang eines einzigen Diagramms unter Verwendung der Materialeinstellung ***Transparent*** rund drei bis vier Minuten länger als bei der Option ***Normal***. Führt man auch hier die entsprechende Hochrechnung für acht Diagramme durch, so liegt der Unterschied schon bei ca. 27 Minuten. Da es sich bei dem transparenten Material allerdings nicht um eine tatsächlich rechenintensive Einstellung handelt, kann bei komplexeren Effekten, wie Spiegelungen, Refraktionen oder ähnlichem, eine deutlichere Erhöhung der Renderdauer erwartet werden. Dies ist in jedem Fall beim Hinzufügen weiterer Materialien zu beachten. Obwohl beim Animationstyp ***Einblenden*** ebenfalls Transparenzeffekte verwendet wurden, zeichnet sich im Vergleich mit der Option ***Wachsend*** kein nennenswerter Unterschied ab. Auch bei den diversen Dateigrößen lassen sich kaum Abweichungen feststellen.

5.4. Vergleich vollständiger Präsentationsvideos

Abschließend stellt der vorliegende Abschnitt einen Vergleich zweier vollständig gerenderter Präsentationsvideos dar. Auch hierfür wurde die bereits mehrfach erwähnte Testszene verwendet, welche neben vier verschiedenen Diagrammen ebenso eine Titelanimation enthält. Auf Basis der oben beschriebenen Ergebnisse ließ sich das erste Video nun so wählen, dass die Dauer des Rendervorgangs und die daraus resultierende Dateigröße möglichst gering blieb. Im Gegensatz dazu verwendet das Zweite sämtliche Optionen, welche die Renderdauer erhöhen. Zu erwähnen ist allerdings, dass in beiden Fällen die Einstellungen bezüglich der unterschiedlichen Animationszeiten im entsprechenden Teilformular nicht manipuliert wurden. Mit Hilfe dieser letzten Messung lässt sich nun darstellen, wie stark die Dauer des Rendervorgangs und die Dateigröße zweier Videos voneinander abweichen können, obwohl beide die gleichen Daten visualisieren. Auch hier zeigt eine entsprechende Tabelle (5.4) neben den jeweiligen Einstellungen die gemessenen Ergebnisse.

	Video 1	Video 2
Diagrammtyp	Liniendiagramm	Balkendiagramm
Diagramme ausblenden	deaktiviert	aktiviert
Material	Normal	Transparent
Auflösung	512 x 384	1024 x 768
Einzelbilder	1552	4472
Videodauer	1:08,08 min	2:58,88 min
Renderdauer	00:23:21 h	3:57:15 h
pro Einzelbild	0,903 s	3,183 s
Dateigröße	1.757 KB	33.626 KB

Tabelle 5.4.: Vergleich zweier Präsentationsvideos

Bevor wir nun einen genaueren Blick auf die Messwerte werfen, sollen zunächst die dafür verantwortlichen Einstellungen vorgestellt werden. Wie in Abschnitt 5.2 gezeigt, lässt sich unter Verwendung von Liniendiagrammen die Laufzeit des Videos und somit auch dessen Renderdauer im Gegensatz zu den übrigen minimieren. Dies rechtfertigt die Entscheidung bezüglich der Diagrammtypen. Ebenso konnte ein nennenswerter Unterschied zwischen den beiden verwendeten Materialen festgestellt werden, was auch deren Verwendung begründet. Die nächste Einstellung, welche zu dieser Messung hinzugenommen wurde, ist für das Ausblenden der Diagramme nach dessen Erzeugung zuständig. Hierdurch lässt sich die Dauer des zweiten Videos, bei dem die Einstellung aktiviert ist, weiterhin erhöhen. Zusätzliche Informationen über diese Option finden sich in Abschnitt 3.2.3. Auf Basis der Ergebnisse der ersten Untersuchung wird ebenfalls die entsprechende Auflösung auf das Minimum reduziert bzw. auf das Maximum erhöht.
Die hieraus resultierende Anzahl an Einzelbildern und die jeweilige Videodauer lassen sich in den entsprechenden Zeilen von Tabelle 5.4 betrachten. Diese machen deutlich, wie stark die beiden Videos aufgrund der Optionen **Diagrammtyp** und **Diagramme ausblenden** alleine schon bei deren Länge voneinander abweichen. Zusammen mit den beiden verbliebenen Einstellungen **Material** und **Auflösung** ergibt sich nun bei der Dauer des Rendervorgangs ein Unterschied von mehr als dreieinhalb Stunden. Bei den Werten pro Einzelbild zeichnet sich ein Verhältnis von ca. 1:3,5 ab, welches sich durch die Auswahl der Option ***Transparent*** bei den Materialeinstellungen um einen zusätzlichen Betrag erhöht hat. Hiermit wird allerdings nur die Effizienz beim Rendern eines Einzelbildes angegeben, was bedeutet, dass lediglich die beiden zuletzt erwähnten Einstellungen an dieser Stelle eine Rolle spielen. In Verbindung mit der Geschwindigkeit des ChartFlight Services ist aber nicht nur dieses Verhältnis entscheidend, sondern dessen Kombination mit der Länge des Präsentationsvideos. So kann es beispielsweise sinnvoll sein, sich entweder für die Einstellung ***Transparent*** oder für die Option zum Ausblenden der Diagramme zu entscheiden, womit man die eine mit der anderen ausgleichen könnte. Auch bei den Dateigrößen zeichnet sich ein deutlicher Unterschied ab. Diese stehen in einem Verhältnis von ca. 1:19 zueinander, woraus allerdings zunächst keine besonderen Beobachtungen zu machen sind. Betrachtet man aber das vierte Video von Tabelle 5.1, so lässt sich erkennen, dass hier Gemeinsamkeiten zu finden sind. Rechnet man die

dort aufgeführte Dateigröße von 7.814 KB auf die Anzahl von vier Diagrammen hoch, ergibt sich ein Wert von 31.256 KB, welcher dem in diesem Abschnitt dargestellten ziemlich nahe kommt. Beachtet man nun weiterhin, dass bei dem oben beschriebenen Video zusätzlich die Titelanimation hinzukommt, so ist eine Dateigröße von 33.626 KB realistisch. Dieses Beispiel sollte lediglich verdeutlichen, dass auch eine Hochrechnung auf Basis der Dateigrößen möglich wäre.
Abschließend soll noch erwähnt werden, dass bei sämtlichen Versuchen die Anzahl der Messungen auf ein Minimum reduziert wurde und sich mit einer aufwendigeren Versuchsreihe sicherlich weitere Auffälligkeiten entdecken ließen.

6 Kapitel 6.

Zusammenfassung und Ausblick

In dieser Diplomarbeit wurde die Entwicklung eines Services zur Visualisierung von Daten auf einer vorgegebenen Hintergrundgrafik im dreidimensionalen Raum vorgestellt. Nach einer kurzen Einführung und der entsprechenden Motivation des Themas ließen sich mit Hilfe des zweiten Kapitels sämtliche Grundlagen dieser Arbeit legen. Hierin wurde zunächst deren Basis in Form der 3D Modellierungs- und Animationssoftware Blender in mehreren Unterabschnitten behandelt, um daraufhin weitere wichtige Elemente, wie die zugrundeliegende SQLite-Datenbank, Java-Applets oder verschiedene Bibliotheken zum Versenden von E-Mails näher zu betrachten.
Im Anschluss an dieses Kapitel folgte eine Beschreibung des ChartFlight Services aus der Sicht eines Benutzers. Hierbei wurde in diversen Abschnitten dessen Funktionalität detailliert erläutert und die unterschiedlichen Möglichkeiten bei der Erzeugung eines Auftrags anhand von Beispielen aufgezeigt. Ebenso enthält dieses Kapitel Informationen über den grundsätzlichen Aufbau eines Präsentationsvideos und eine Anleitung zum Erstellen der benötigten Bild- und Diagrammdaten. Abschließend wurden die Ergebnisse eines Auftrags anhand unterschiedlicher Abbildungen und eines weiteren Beispiels illustriert.
Das darauffolgende Kapitel über die Architektur des ChartFlight Services bot zunächst einen Überblick über die unterschiedlichen Komponenten und deren Zusammenarbeit. Weiterhin wurde der SQLite-Datenbank ein eigener Abschnitt gewidmet und darin erklärt, wie ein Auftrag innerhalb dieser Datenbank abläuft. Hiernach beschäftigten sich drei weitere Abschnitte mit einer detaillierteren Vorstellung der einzelnen Komponenten, wobei das Backend, der Job-Listener und die Benutzerschnittstelle im Vordergrund standen. Bei diesen konnte zunächst anhand eines Unterabschnitts ein Überblick über die jeweilige Menge an Klassen, Methoden und Funktionen gegeben werden, um anschließend unter Verwendung diverser Codebeispiele

die zugrundeliegende Implementierung zu erläutern. Ein weiterer Abschnitt innerhalb dieses Kapitels zeigte verschiedene Probleme auf, welche während der Implementierung der vorliegenden Diplomarbeit auftraten und gegebenenfalls umgangen werden mussten. Das Ende dieses Kapitels beschäftigte sich mit der Installation und der anschließenden Konfiguration des ChartFlight Services. Hierin wurde zunächst dessen Verzeichnisstruktur erläutert und daraufhin angegeben, wie sich die diversen Einstellungen innerhalb der unterschiedlichen Konfigurationsdateien manipulieren lassen und welche Auswirkungen dies hat.
Im abschließenden Kapital 5 wurden die Leistungsdaten des Services diskutiert. Dabei ließen sich anhand einiger Testläufe erste Ergebnisse bezüglich der Dauer diverser Rendervorgänge bei unterschiedlichen Auflösungen bzw. anderen leistungsbezogenen Einstellungen erzielen. Ebenso wurden dort die Dateigrößen der erstellten Präsentationsvideos betrachtet.

Zum Abschluss dieser Diplomarbeit soll eine zukünftige Weiterentwicklung des ChartFlight Services diskutiert werden. Hierbei stehen speziell im Bereich des Backends eine ganze Reihe interessanter Möglichkeiten zur Verfügung, die Qualität und das Design der verschiedenen dreidimensionalen Objekte zu ändern. Dies erfordert allerdings einen leistungsstarken Computer. Steht ein solcher zur Verfügung, so kann beispielsweise mit Einstellungen bezüglich des *Radiosity* Verfahrens oder der Algorithmen für *Ambient Occlusion* experimentiert werden. Beide Techniken erzeugen auf unterschiedliche Weise eine Art Umgebungsbeleuchtung, welche für weichere Schattierungen sorgt. Ebenso könnten neue Materialien und Farbschemata zur Auswahl hinzugefügt werden. Auch dafür stehen unter Blender ein ganze Reihe von unterschiedlichen Einstellungsmöglichkeiten zur Verfügung, welche ebenfalls über die Python API erreichbar sind.
Ein weitere Möglichkeit wäre das Einbinden der in Blender verfügbaren Game Engine zur Erzeugung sämtlicher Effekte in Echtzeit. Dies würde allerdings der Grundidee der vorliegenden Diplomarbeit entgegenwirken, da das hieraus resultierende Ergebnis eine Anwendung zum Ausführen auf dem Computer des Benutzer wäre. Eine solche könnte beispielsweise nicht in eine Präsentation eingebunden werden und wäre zudem abhängig von der Hardware des jeweiligen Computers. Trotz alledem könnte das Erzeugen von Echtzeitpräsentationen für einige Benutzer interessant sein.
Betrachtet man nun wiederum das Präsentationsvideo, so gibt es mindestens zwei Arten dieses dem Zuschauer vorzuführen. Hierunter wäre zunächst die Möglichkeit während des Videos unterstützende Erklärungen zu geben oder dies erst danach zu tun. Im zweiten Fall könnte das Video mit einer eigens dafür generierten Hintergrundmusik versehen und somit für den Betrachter auch auditiv interessant gemacht werden.
Speziell in Bezug auf die Benutzeroberfläche wäre es sinnvoll, weitere Testläufe mit mehreren neutralen Beteiligten durchzuführen. Zum einen, um den Service einer Art Stresstest zu unterziehen und zum anderen, um herauszufinden, wie gut sich der Benutzer beim Erzeugen eines Auftrags zurechtfindet und ob für diesen die verfügbaren Erklärungen ausreichend sind. Weiterhin sollte auch die Leistungsfähigkeit der SQLite-Datenbank, welche bei weni-

gen gleichzeitigen Zugriffen keine Probleme bereitet, bei einer großen Anzahl von Anfragen getestet werden. Sollte es dabei zu Schwierigkeiten kommen, so lässt sich prinzipiell durch eine Änderung der jeweiligen Treiber beispielsweise auf eine MySQL-Datenbank umsteigen. Auf diese Art und Weise finden sich sicherlich noch andere Möglichkeiten zur Weiterentwicklung des ChartFlight Services.

A Anhang A.

Anhang

```
def createScene(self):
  point_list = self.__user_data.getPointList();
  summary = self.__user_data.getSummary();
  anim_settings = self.__user_data.getAnimSettings();
  app_settings = self.__user_data.getAppSettings();
  render_settings = self.__user_data.getRenderSettings();
  title_settings = self.__user_data.getTitleSettings();

  self.__first_frame = 1;

  # load animation properties
  title_stalling_frames = anim_settings.getTitleStallingFrames();
  intro_stalling_frames = anim_settings.getIntroStallingFrames();
  fly_to_frames = anim_settings.getFlyToFrames();
  camera_anim_frames = anim_settings.getCameraAnimFrames();
  frames_per_camera = anim_settings.getFramesPerCamera();
  fly_off_frames = anim_settings.getFlyOffFrames();
  outro_stalling_frames = anim_settings.getOutroStallingFrames();
  fly_to_summary_frames = anim_settings.getFlyToSummaryFrames();
  summary_fading_frames = anim_settings.getSummaryFadingFrames();
  summary_stalling_frames = anim_settings.getSummaryStallingFrames();

  # load color schema
  color_schema = ...

  # load summary color schema
  if not (summary == None):
```

```
28    summary_color_schema = ...
29
30  # generate background map
31  self.__map = Map(self.__user_data.getImage());
32  self.__map.createImagePlane();
33
34  # generate lighting
35  self.__lighting = Lighting(self.__map,
36    self.__config.getLightingMode(),
37    self.__config.getLightEnergy());
38
39  self.__lighting.setSpecularity(self.__config.getSpecularity());
40  self.__lighting.createLamps();
41
42  # compute maximum diagram size
43  max_diagram_size = BasicChart.CHART_SIZE * app_settings.getDiagramSize();
44
45  # create pie chart object
46  if (point_list.getDiagramType() == PointList.PIES):
47    self.__diagrams = PieChart(point_list,
48      self.__map,
49      anim_settings.getAnimMode(),
50      max_diagram_size, # max_width
51      0.0, # max_height (automatic height computation)
52      app_settings.getMaterialMode(),
53      app_settings.getTextMode(),
54      color_schema,
55      app_settings.getFontColor());
56
57    self.__diagrams.animSettings(anim_settings.getAnimFrames(),
58      anim_settings.getFramesPerElement(),
59      anim_settings.getFramesPerChart(),
60      anim_settings.getFramesPerDiagram());
61
62    camera_offset = Vector(0.0, -3.0, 2.0); # offset for pie charts
63
64  # create ring chart object
65  elif (point_list.getDiagramType() == PointList.RINGS):
66    self.__diagrams = RingChart(point_list,
67      self.__map,
68      anim_settings.getAnimMode(),
69      max_diagram_size, # max_width
70      0.0, # max_height (automatic height computation)
71      app_settings.getMaterialMode(),
72      app_settings.getTextMode(),
```

```
73      color_schema,
74      app_settings.getFontColor());
75
76    self.__diagrams.animSettings(anim_settings.getAnimFrames(),
77      anim_settings.getFramesPerElement(),
78      anim_settings.getFramesPerChart(),
79      anim_settings.getFramesPerDiagram());
80
81    camera_offset = Vector(0.0, -2.5, 2.0); # offset for ring charts
82
83  # create bar chart object
84  elif (point_list.getDiagramType() == PointList.BARS):
85    self.__diagrams = BarChart(point_list,
86      self.__map,
87      anim_settings.getAnimMode(),
88      max_diagram_size, # max_width
89      1.0, # max_height
90      app_settings.getMaterialMode(),
91      app_settings.getTextMode(),
92      color_schema,
93      app_settings.getFontColor());
94
95    self.__diagrams.animSettings(anim_settings.getAnimFrames(),
96      anim_settings.getFramesPerElement(),
97      anim_settings.getFramesPerChart(),
98      anim_settings.getFramesPerDiagram());
99
100   camera_offset = Vector(0.0, -3.0, 2.0); # offset for bar charts
101
102 # create line chart object
103 elif (point_list.getDiagramType() == PointList.LINES):
104   self.__diagrams = LineChart(point_list,
105     self.__map,
106     anim_settings.getAnimMode(),
107     anim_settings.getLinesAnimMode(),
108     max_diagram_size, # max_width
109     max_diagram_size * 0.666, # max_height
110     max_diagram_size * 0.133, # width_offset
111     max_diagram_size * 0.1, # height_offset
112     app_settings.getMaterialMode(),
113     app_settings.getTextMode(),
114     color_schema,
115     app_settings.getFontColor(),
116     app_settings.getCanvasColor());
117
```

```
118     self.__diagrams.animSettings(anim_settings.getAnimFrames(),
119       anim_settings.getCaptionAnimFrames(),
120       anim_settings.getLineAnimFrames(),
121       anim_settings.getFramesPerElement(),
122       anim_settings.getFramesPerChart(),
123       anim_settings.getFramesPerDiagram());
124
125     camera_offset = Vector(0.0, -3.0, 1.25); # offset for line charts
126
127   # compute camera position
128   cam_posX = ...
129   cam_posY = ...
130   cam_posZ = ...
131
132   global_camera_pos = Vector(cam_posX, -cam_posY, cam_posZ);
133   global_view_point = Vector(cam_posX, self.__map.getHeight()/2.0, 0.0);
134   summary_camera_pos = Vector(cam_posX, -cam_posY, cam_posZ);
135   summary_view_point = Vector(cam_posX, self.__map.getHeight() * 1.5, 0.0);
136
137   # compute different start and end frames (1)
138   title_sFrame = self.__first_frame;
139   title_eFrame = title_sFrame + title_stalling_frames;
140   self.__preview_frames += [title_eFrame - title_stalling_frames/3];
141
142   intro_sFrame = title_eFrame;
143   intro_eFrame = intro_sFrame + intro_stalling_frames;
144   self.__preview_frames += [intro_sFrame];
145
146   diagram_anim_sFrame = intro_eFrame + fly_to_frames + frames_per_camera;
147
148   # diagram properties
149   self.__diagrams.setCameraAnimFrames(camera_anim_frames);
150   self.__diagrams.setFramesPerCamera(frames_per_camera);
151   self.__diagrams.setFramesPerCamera(fly_off_frames);
152   self.__diagrams.setMap(self.__map);
153   self.__diagrams.setShowValues(app_settings.getDiagramsShowValues());
154   self.__diagrams.setFadingOut(anim_settings.getFadingOutAnim());
155
156   # generate diagrams
157   self.__diagrams.createDiagrams(diagram_anim_sFrame);
158
159   # compute different start and end frames (2)
160   stalling_frames = self.__diagrams.getCameraStallingFrames();
161   self.__preview_frames += [(stalling_frames[0][1] + stalling_frames[0][0])/2];
162
```

```
163   outro_sFrame = stalling_frames[len(stalling_frames)-1][1] + fly_off_frames;
164   outro_eFrame = outro_sFrame + outro_stalling_frames;
165   self.__preview_frames += [int(outro_sFrame)];
166
167   if not (summary == None):
168     summary_sFrame = outro_eFrame + fly_to_summary_frames;
169     summary_eFrame = summary_sFrame + summary_fading_frames;
170   else:
171     summary_sFrame = -1;
172     summary_eFrame = -1;
173
174   # generate camera animation
175   self.__camera_anim = CameraAnimation(point_list,
176     self.__map,
177     camera_offset,
178     intro_sFrame,
179     intro_eFrame,
180     outro_sFrame,
181     outro_eFrame,
182     global_camera_pos,
183     global_view_point,
184     summary_camera_pos,
185     summary_view_point,
186     stalling_frames,
187     summary_sFrame,
188     summary_eFrame);
189
190   self.__camera_anim.createAnimation();
191
192   # generate title
193   self.__title = Title(title_settings.getTitle(),
194     title_settings.getSubtitle(),
195     title_settings.getAuthor(),
196     title_settings.getDate(),
197     title_settings.getFooter(),
198     title_settings.getFontColor(),
199     title_settings.getBackColor(),
200     title_settings.getImageSource(),
201     title_settings.getMode(),
202     title_settings.getLayout());
203
204   self.__title.createTitle(self.__camera_anim, title_sFrame, title_eFrame);
205
206   # generate keys
207   self.__key = Key(point_list,
```

```
208     self.__map,
209     camera_offset,
210     self.__diagrams.getCameraStallingFrames(),
211     AppSettings.MAT_ZTRANSP,
212     color_schema,
213     0.225, # max_width
214     1.0, # max_height
215     0.05, # text_size
216     0.06, # header_size
217     0.05); # line_distance
218 
219   self.__key.createKeys();
220 
221   # generate summary chart
222   if not (summary == None):
223     self.__summary_chart = SummaryChart(summary,
224       summary_color_schema,
225       [0.0, 0.0, 0.0], # font color
226       [1.0, 1.0, 1.0], # background color
227       app_settings.getSummaryShowValues());
228 
229     self.__summary_chart.createChart(self.__camera_anim,
230       summary_sFrame,
231       summary_eFrame);
232     self.__preview_frames += [int(summary_eFrame)];
233 
234     self.__last_frame = int(summary_eFrame + summary_stalling_frames);
235   else:
236     self.__last_frame = int(outro_eFrame);
237 
238   # world settings
239   world = World.GetCurrent();
240 
241   if (world == None):
242     world = World.New("World");
243     Scene.GetCurrent().world = world;
244 
245   world.setHor(app_settings.getWorldColor());
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Ble08a] Blender Foundation: *Blender*. http://www.blender.org, November 2008.

[Ble08b] Blender Foundation: *Blender Manual*. http://wiki.blender.org/index.php/Manual/, November 2008.

[Ble08c] Blender Foundation: *The Blender Python API Reference*. http://www.blender.org/documentation/246PythonDoc/, November 2008.

[Cra08] Crawshaw, David: *SQLiteJDBC*. http://www.zentus.com/sqlitejdbc/, November 2008.

[Fis08] Fischer, Marcus: *Ubuntu GNU/Linux - Aktuell zu Hardy Heron*. Galileo Computing, 3 Auflage, 2008.

[Fra08] Frank, Florian: *SelfLinux*. http://www.selflinux.org, November 2008.

[Hip08] Hipp, Dr. Richard: *SQLite*. http://www.sqlite.org, November 2008.

[KE08] Kaiser, Peter und Johannes Ernesti: *Python - Das umfassende Handbuch*. Galileo Computing, 2008.

[KHM05] Kücükyilmaz, Hakan, Thomas Haas und Alexander Merz: *Einsteigen und durchstarten mit PHP 5*. dpunkt.verlag GmbH, 1 Auflage, 2005.

[Kra04] Krause, Jörg: *PHP 5 - Grundlagen und Profiwissen*. Hanser Fachbuchverlag, 1 Auflage, 2004.

[Möh05] Möhrke, Carsten: *PHP PEAR - Anwendung und Entwicklung*. Galileo Computing, 1 Auflage, 2005.

[Pen08] Penz, Gerold: *simplemail*. http://gelb.bcom.at/trac/simplemail/, November 2008.

[Pil04] Pilgrim, Mark: *Dive Into Python*, 05 2004.

[PW08] Plötner, Johannes und Steffen Wendzel: *Linux*. Galileo Computing, 2 Auflage, 2008.

[Pyt08] Python Software Foundation: *Python Documentation*. http://www.python.org/doc/, November 2008.

[SEL08a] SELFHTML E.V.: *SelfHTML*. http://de.selfhtml.org, November 2008.

[SEL08b] SELFPHP OHG: *SelfPHP*. http://www.selfphp.de, November 2008.

[Sof08] SOFTCOMPLEX: *Tigra Color Picker*. http://www.softcomplex.com/products/tigra_color_picker/, November 2008.

[Sta08] STATISTISCHES BUNDESAMT: *Der Bundeswahlleiter*. http://www.bundeswahlleiter.de, November 2008.

[Sun08a] SUN MICROSYSTEMS, INC.: *The Java Tutorials*. http://java.sun.com/docs/books/tutorial/, November 2008.

[Sun08b] SUN MICROSYSTEMS, INC.: *JavaMail*. http://java.sun.com/products/javamail/, November 2008.

[Sun08c] SUN MICROSYSTEMS, INC.: *JDK 6 Documentation*. http://java.sun.com/javase/6/docs/, November 2008.

[Swa05] SWAROOP C H: *A Byte of Python*, 2005.

[The08] THE PHP GROUP: *PHP.net*. http://www.php.net, November 2008.

[Ull07] ULLENBOOM, CHRISTIAN: *Java ist auch eine Insel: Programmieren mit der Java Standard Edition Version 6*. Galileo Computing, 7 Auflage, 2007.

[War07] WARTMANN, CARSTEN: *Das Blender-Buch*. dpunkt.verlag GmbH, 3 Auflage, 2007.

[Wen06] WENZ, CHRISTIAN: *JavaScript und Ajax*. Galileo Computing, 7 Auflage, 2006.

[Wik08a] WIKIMEDIA FOUNDATION: *Atlas of Germany*. http://commons.wikimedia.org/wiki/Atlas_of_Germany/, November 2008.

[Wik08b] WIKIMEDIA FOUNDATION: *Blender 3D: Blending Into Python*. http://en.wikibooks.org/wiki/Blender_3D:_Blending_Into_Python/, November 2008.

[Wik08c] WIKIMEDIA FOUNDATION: *Blender 3D: Noob to Pro*. http://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/, November 2008.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe. Die Diplomarbeit habe ich bisher keinem anderem Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher auch nicht veröffentlicht.

Rainer Lutz Trier, den 24. November 2008