Entwicklung einer GUI-gestützten Datenbankanwendung zu Abfrage, Visualisierung und Reporting von Emissionsstudiendaten aus dem Luftverkehr

BACHELORARBEIT

für die Prüfung zum

Bachelor of Engineering

des Studienganges Informationstechnik

an der Dualen Hochschule Baden-Württemberg Mannheim

von

Oliver Seebach

20. September 2010

Bearbeitungszeitraum	12 Wochen
Matrikelnummer, Kurs	290981, TIT07ANS
Ausbildungsfirma	Deutsches Zentrum für Luft- und Raumfahrt e.V. Köln-Porz
Betreuer der Ausbildungsfirma	Dr. Christos Lois
Gutachter der Dualen Hochschule	Prof. Dr. Rainer Colgen

Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit von mir selbstständig angefertigt wurde und ich keine weiteren als die angegebenen Hilfsmittel und Quellen benutzt habe.

Köln, den 15. September 2010

Kurzfassung

Das Institut für Flughafenwesen und Luftverkehr beschäftigt sich mit Forschungen im Luftverkehr. Gegenstand der Forschung ist unter anderem die Untersuchung der Frage, wie sich das System Luftverkehr unter verschiedenen Einflussbedingungen verändert.
Ziel der Arbeit ist es, eine Anwendung zu entwickeln und prototypisch zu implementieren, die es ermöglicht, Emissionskataster, d.h. räumlich untergliederte Raster von Emissionen, graphisch und tabellarisch auszuwerten und als Bericht zu exportieren.
Um die Kataster zu visualisieren, mussten die Daten bisweilen händisch vorverarbeitet und anschließend in eine GIS-Software übertragen werden. Daher konnte die Ausführung nur von Mitarbeitern mit GIS- und IT-Vorbildung durchgeführt werden. Die Ergebnisse wurden in heterogenen Formaten an verschiedenen Orten gespeichert.
Da es für alle wissenschaftlichen Mitarbeiter möglich sein soll, Auswertungen von Emissionskatastern vorzunehmen, wurde eine intuitiv bedienbare Anwendung mit graphischer Benutzeroberfläche entwickelt. Diese wurde mit Hilfe der Entwicklungsumgebung NetBeans in der Version 6.9 als NetBeans Module-Suite, d.h. eine Zusammenstellung gekapselter Module, in der Programmiersprache Java implementiert und bereitet die Emissionskataster automatisiert auf. Beim Entwurf der softwaretechnischen Architektur wurde nach dem MVC-Entwurfsmuster vorgegangen. Die Emissionskataster liegen in einer MySQL-Datenbank vor. Ein Export der Ergebnisse ist in Form einer Weltkarte mit graphisch aufbereiteten Emissionsdaten und als Microsoft Excel-Dokument möglich.
Mit Hilfe der entwickelten Anwendung lassen sich Auswertungen von Emissionskatastern ohne technisches Fachwissen vornehmen.
Aufgrund des modularen Aufbaus ist es möglich, die Anwendung zu erweitern. So ist die Definition weiterer Auswertungstypen jederzeit möglich. Ferner ist es möglich, dass externe Firmen die Anwendung nutzen, sofern diese über eine entsprechende Datengrundlage verfügen.
Die erstellte Anwendung liefert ein Werkzeug zum Erstellen von Berichten, die für die Bewertung der Klimawirksamkeit zukünftiger Flugpläne und Flugzeug-Technologien wichtig sind.

Abstract

One focus of research of the Institute of Air Transport and Airport Research is to examine how the system air traffic evolves under certain conditions.

The aim of this bachelor thesis is the development and prototypical implementation of an application that analyses emission inventories, i.e. spatially arranged grids of emissions, via graphics or tables. The results can be exported in form of reports.

So far data has been manually pre-processed and transferred to a GIS-software in order to visualize inventories. Therefore, the execution has been limited to employees with GIS- and IT-knowledge. The results have been stored in heterogenous formats at different locations.

In order to enable all employees to analyse emission inventories, an application with an intuitively usable graphical user interface has been developed. This application has been implemented with the aid of the NetBeans IDE 6.9 as a NetBeans Module-Suite, i.e. a compilation of encapsulated modules, in the programming language Java and processes emission inventories automatically. The architectural design is based on the design pattern MVC. The emission inventories are stored in a MySQL database. It is possible to export the results as a world map with graphically processed emissions and as a Microsoft Excel document.

With the aid of the application developed in the course of this thesis, analysis of grid inventories can be executed without technical knowledge.

Due to the modular design it is possible to upgrade the application. Thus, it is possible to define further types of analysis. Additionally, external companies are able to use the application, given that they provide an adequate database.

The application provides a tool for creating reports, which are important with regard to the evaluation of flight plans and aircraft technologies in the future.

Abkürzungsverzeichnis

4D-FATE	**F**our-dimensional calculation of **A**ircraft **T**rajectories and **E**missions
API	**A**pplication **P**rogramming **I**nterface
ATS	**A**ir **T**ransport **S**ystem
AWT	**A**bstract **W**indow **T**oolkit
BLOB	**B**inary **L**arge **Ob**ject
DAO	**D**ata **A**ccess **O**bject
DBMS	**D**aten**b**ank**m**anagement**s**ystem
DLR	**D**eutsches Zentrum für **L**uft und **R**aumfahrt e.V.
ERM	**E**ntity **R**elationship **M**odel
EU	**E**uropäische **U**nion
GIS	**G**eographisches **I**nformations**s**ystem
GUI	**G**raphical **U**ser **I**nterface
IATA	**I**nternational **A**ir **T**ransport **A**ssociation
IDE	**I**ntegrated **D**evelopment **E**nvironment
JAR	**J**ava **Ar**chive
JTI	**J**oint **T**echnology **I**nitiative
MVC	**M**odel-**V**iew-**C**ontroller
MVP	**M**odel-**V**iew-**P**resenter
OAG	**O**fficial **A**irline **G**uide
OSGi	**O**pen **S**ervices **G**ateway **i**nitiative
PDF	**P**ortable **D**ocument **F**ormat
REST	**R**epresentational **S**tate **T**ransfer
RFC	**R**equest **F**or **C**omments
SQL	**S**tructured **Q**uery **L**anguage
ST	**S**tandard **T**ime
URL	**U**niform **R**esource **L**ocator
UTC	**U**niversal **T**ime **C**oordinated

Abbildungsverzeichnis

Listings

Inhaltsverzeichnis

1. Einleitung

Dieses Kapitel beschreibt das betriebliche Umfeld, sowie die Motivation und Aufgabenstellung. Anschließend wird die Vorgehensweise, sowie die Struktur der schriftlichen Ausarbeitung beschrieben, um dem Leser einen Überblick zu verschaffen.

1.1. Umfeld der Bachelorarbeit

Die Bachelorarbeit wurde beim Deutschen Zentrum für Luft- und Raumfahrt e.V. (DLR) am Standort Köln-Porz im Institut für Flughafenwesen und Luftverkehr erstellt.
Das DLR ist das nationale Forschungszentrum der Bundesrepublik Deutschland für Luft- und Raumfahrt. Seine Schwerpunkte liegen in der nationalen und internationalen Forschung auf den Gebieten der Luftfahrt, Raumfahrt, Verkehr und Energie. Neben der Forschung liegt die Koordination und Umsetzung der deutschen Raumfahrtaktivitäten im Aufgabenfeld des DLR. Das DLR forscht an dreizehn Standorten in Deutschland, an denen 29 Institute angesiedelt sind. Insgesamt beschäftigt das DLR ca. 6500 Mitarbeiterinnen und Mitarbeiter. [2]
Das Institut für Flughafenwesen und Luftverkehr beschäftigt sich mit Forschungen im Luftverkehr. Gegenstand der Forschung ist die Untersuchung der Frage, wie sich sowohl das System Luftverkehr als auch das System Flughafen im Zeitablauf unter verschiedenen Einflussbedingungen verändern.
Im Bereich der Luftverkehrsforschung zielen die Aktivitäten auf langfristige, strategische Maßnahmen, z.b. Änderungen an der Infrastruktur oder regulative Maßnahmen auf das Luftverkehrssystem als Ganzes. In der Flughafenforschung beziehen sich die Maßnahmen auch auf den taktisch-operativen Bereich. Der Flughafen wird dabei als Protoyp eines Verkehrsknotens angesehen, sodass Erkenntnisse der Forschung auch auf andere intermodale Verkehrsknoten übertragbar sind. Die in der Einrichtung Flughafenwesen und Luftverkehr integrierte Forschungskette von strategischen bis hin zu operativen Maßnahmen ermöglicht die Erarbeitung harmonisierter, aufeinander abgestimmter Verfahren. [3]

1.2. Motivation

Das Institut für Flughafenwesen und Luftverkehr ist in mehreren EU-Projekten aktiv. Eines dieser Projekte ist die Joint Technology Initiative (JTI) *Clean Sky*. Eine JTI ist ein von der EU geförderten Projekt, um anspruchsvolle und komplexe Tätigkeiten, z.B. die Evaluation weit entwickelter Technologien durchzuführen.
Das Projekt *Clean Sky* hat die Entwicklung und Evaluierung neuer Technologien mit geringem Schadstoffausstoß zum Ziel. Somit wird die nachhaltige Klimaverbesserung unterstützt. Für die Evaluierung spielt der sogenannte *Technology Evaluator* eine Schlüsselrolle. Das Institut für Flughafenwesen und Luftverkehr leistet dazu unter anderem mit der Entwicklung der Anwendung, die im Rahmen dieser Arbeit erstellt wurde, seinen Beitrag. Die im DLR entwickelten Tools VarMission und 4D-FATE liefern hierfür Emissionskataster[1] als Rohdaten.
Diese Rohdaten lassen sich auswerten und in geeigneter Form (tabellarisch, graphisch) darstellen. Es ist angestrebt, auf Basis von Luftverkehr-Prognosemodellen Flugpläne (Flugbewegungen) für die Zukunft zu ermitteln. Faktoren, wie Verkehrswachstum und neue Flugzeugtechnologien, können als Parameter einbezogen werden.
Dadurch ist es möglich, die Auswirkungen veränderter Technologien und Flugpläne auf die Emissionsmengen und deren Verteilungen zu erkennen. Die Darstellung der Ergebnisse soll im Rahmen der Arbeit in Form von Tabellen und einer Weltkarte mit geographischer Zuordnung der Emissionen erfolgen. In Folgearbeiten ist eine Erweiterung um weitere Darstellungsarten möglich. [4]
Ein weiterer Aspekt der zu entwickelnden Anwendung ist es, wissenschaftlichen Mitarbeitern ohne Programmier- und Datenbankkenntnisse die Möglichkeit zu geben, selbstständig Auswertungen der vorhandenen Daten durchzuführen.

1.3. Aufgabenstellung

Als Beitrag zu den oben genannten Zielen des Projekts, wurde folgende Aufgabenstellung formuliert: Es soll eine Anwendung entwickelt werden, die es dem Anwender

[1] Ein Emissionskataster ist ein räumlich untergliedertes Raster von Emissionen

mit Hilfe einer Benutzeroberfläche ermöglicht, statistische Abfragen auf Emissionsstudiendaten aus dem Flugverkehr aus einer Datenbank durchzuführen, die Ergebnisse zu visualisieren und anschließend Berichte in üblichen Formaten zu erstellen.
Daraus ergeben sich folgende Arbeitsschwerpunkte:

1. Strukturierung der Input-Daten, die für die Abfragen benötigt werden
2. Erstellen des relationalen Datenmodells, das der Anwendung zugrunde liegt
3. Speicherung der Rohdaten in einer relationalen Datenbank
4. Validierung der Rohdaten
5. Anforderungsanalyse
6. Konzeptionierung und Implementierung
7. Testen der Software
8. Dokumentation der Ergebnisse

1.4. Zugrundeliegende Programme

Dieses Kapitel beschreibt die zugrundeliegenden Programme VarMission und 4D-FATE, welche die Rohdaten für die zu entwickelnde Software liefern.

1.4.1. VarMission

VarMission ist eine Software zur Berechnung von Treibstoffverbrauch und Schadstoffausstoß auf Flugmissionen als Funktion der Flugposition und -zeit. Die Software ermöglicht die Simulation von Flugmissionen als Abfolge beliebiger Flugphasen und berechnet neben dem Treibstoffverbrauch die daraus resultierenden Emissionen von CO_2, H_2O, SO_x, NO_x, CO, HC und Ruß. Somit lassen sich Emissionsprofile für alle verfügbaren Flugzeug-Triebwerk-Kombinationen erzeugen. Hierbei handelt es sich um tabellarische Protokolle standardisierter Flugmissionen für verschiedene Nutzlasten und Reichweiten. Das folgende Schema veranschaulicht die Quell- und Ergebnisdateien von VarMission. [5, Seite 30 f.]

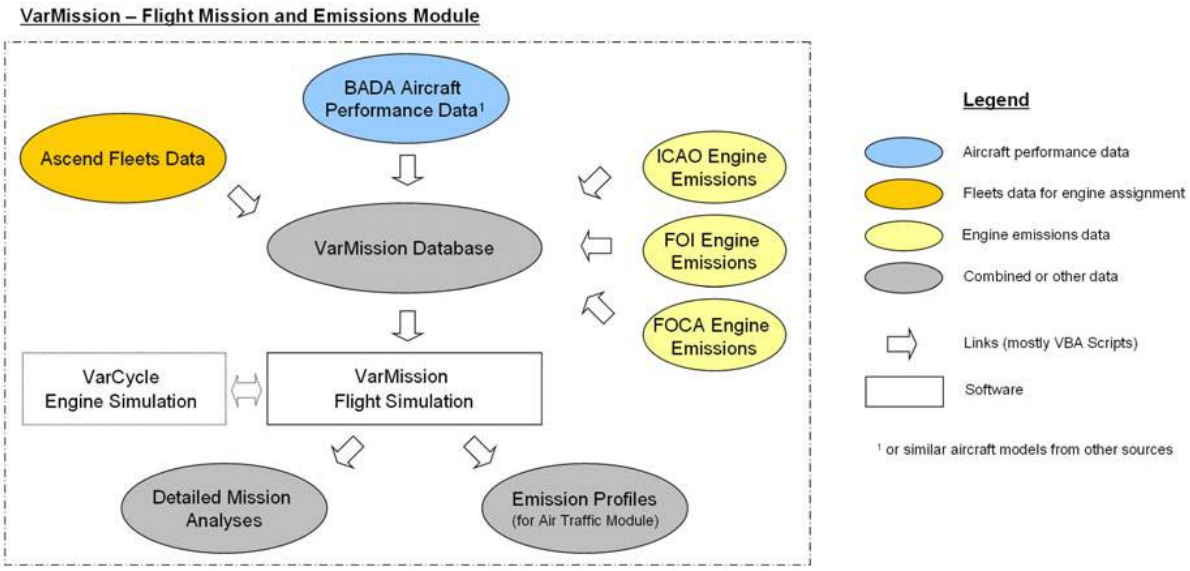


Abbildung 1: *Schematische Darstellung des Tools* VarMission *[1]*

Unter den Abkürzungen *ICAO*, *FOI* und *FOCA* sind Gesellschaften oder Ämter zu verstehen, die Daten zur Verfügung stellen. *BADA Aircraft Performance Data* und *Ascend Fleets Data* sind Datenbanken, die weitere Daten liefern. Diese werden für die Berechnungen von VarMission benötigt. Da diese Quellen nicht Bestandteil der Arbeit sind, wird nicht weiter darauf eingegangen.

1.4.2. 4D-FATE

4D-FATE (Four-dimensional calculation of Aircraft Trajectories and Emissions) ist ein Tool zur Berechnung von Emissionskatastern. Es gestattet die wegpunktgenaue Berechnung des Flugweges und ermöglicht eine zeitliche Kodierung der betrachteten Größen. Dazu wird die Erde als Kugel modelliert, die mit einer Hülle von vierdimensionalen Zellen überzogen ist. Diese Zellen bilden das Kataster. Die Dimensionen einer Zelle sind Höhe, Länge und Breite, sowie die Zeit. 4D-FATE berechnet auf Großkreisbasis anhand von Start- und Zielpunkt einer Flugverbindung die Route und Schnittpunkte mit Zellen. Aus diesen Schnittpunkten lässt sich ermitteln, welche Strecke ein Flug innerhalb einer Zelle zurückgelegt hat. Mit Hilfe dieser Strecke können den einzelnen Zellen des Katasters Emissionsmengen zugeordnet werden. Diese werden in eine Ausgabedatei

geschrieben. 4D-FATE benötigt folgende Input-Dateien:

- Flugverbindungsdatei (Aus offiziellen Flugplänen abgeleitet)
- Emissionsprofildatei (Durch VarMission berechnet)

Die Flugverbindungsdatei enhält Start- und Zielpunkt eines Fluges. Daraus wird die Route auf Großkreisbasis berechnet. Diese Route entspricht der kürzesten Strecke zwischen dem Start- und Zielflughafen auf einer Kugeloberfläche. Die Daten, die sich aus einer Flugverbindungsdatei entnehmen lassen, sind in folgender Liste aufgeführt. Ein Auszug aus einer Flugverbindungsdatei ist im Anhang A.1 angefügt.

- IATA-Code des Startflughafens
- IATA-Code des Zielflughafens
- Flugzeugtyp
- fortlaufende Nummer für einen Programmdurchlauf
- Anzahl der Wegpunkte
- Tageskodierung
- Zeitangabe des Starts in Minuten in UTC[2]
- Zeitangabe der Landung in Minuten in UTC
- Flugfrequenz
- Längengrad des Startflughafens
- Breitengrad des Startflughafens
- Höhe des Startflughafens in Metern
- Längengrad des Zielflughafens
- Breitengrad des Zielflughafens
- Höhe des Zielflughafens in Metern

Zu jeder Flugzeug-Triebwerk-Kombination gibt es eine Emissionsprofildatei. Ein Bei-

[2]Unter *UTC* versteht man die koordinierte Weltzeit

spiel einer solchen Datei ist in Anhang A.2 aufgeführt. Neben einigen Meta-Informationen über den Flug enthält die Datei folgende Informationen:

- Flughöhe
- zurückgelegte Strecke auf dieser Höhe
- Aufenthaltsdauer in dieser Höhe
- durchschnittlich verbrauchter Treibstoff in dieser Höhe
- durchschnittliche NO_x-Emissionen in dieser Höhe
- durchschnittliche CO_2-Emissionen in dieser Höhe
- durchschnittliche HC-Emissionen in dieser Höhe

Als Ausgabe liefert 4D-FATE eine Ergebnis-, eine Kontroll- und eine Protokolldatei. Da für die weitere Bearbeitung nur die Ergebnisdatei von Interesse ist, wird nur auf diese weiter eingegangen. Die Bedeutung der einzelnen Spalten ist in der folgenden Auflistung aufgezählt. Ein Ausschnitt aus einer Ergebnisdatei ist als Anhang A.3 eingefügt.

- Längenkodierung des Katasters
- Breitenkodierung des Katasters
- Höhenkodierung des Katasters
- Uhrzeitkodierung des Katasters in UTC
- Uhrzeitkodierung in ST[3]
- zurückgelegte Strecke über Grund in km
- Aufenthaltsdauer im Kataster in Minuten
- verbrauchter Streibstoff im Kataster in kg
- NO_x-Emissionen in kg
- CO_2-Emissionen in kg
- Ruß-Emissionen

[3]Unter *ST* versteht man die Uhrzeit in einer bestimmten Zeitzone.

- Flugzeugtyp

Wie das Tool 4D-FATE intern arbeitet, ist aufgrund der Komplexität und unübersichtlichen Programmierung nicht nachzuvollziehen. Die Ergebnisse werden als plausibel angenommen. [6, Seite 15 ff.]

1.5. Vorgehensweise

Die Vorgehensweise bei der Bearbeitung der Aufgabe ist an das Wasserfallmodell nach *Balzert* [7, Seite 99 f.] angelehnt. Das begründet sich darin, dass die Entwicklungsphasen eine starke Abhängigkeit voneinander aufweisen. Aufgrund der kurzen Projektdauer wird das ursprüngliche Wasserfallmodell nach *Balzert* angepasst, sodass die Datengrundlage zuerst validiert wird. Rücksprünge zur vorherigen Arbeitsphase sind erlaubt. Die einzelnen Arbeitspakete sind wie folgt:

1. Erstellen des Datenbankmodells, Import und Validierung der Rohdaten
2. Anforderungsanalyse
3. Konzeptionierung und Implementierung
4. Validierung der Software

1.6. Aufbau der Bachelorarbeit

In diesem Abschnitt wird ein Überblick über den Aufbau der Arbeit gegeben, der sich vom klassischen Wasserfallmodell ableitet.

- Kapitel 2: Dieses Kapitel enthält die funktionalen Anforderungen an die Anwendung, die daraus abgeleiteten Usecases, sowie die nichtfunktionalen Anforderungen.
- Kapitel 3: In diesem Kapitel wird die Architektur vorgestellt, die der Anwendung zugrunde liegt.
- Kapitel 4: Dieses Kapitel beschreibt die Datenbank, die verwendet wird. Ebenso

werden das Datenmodell, die Datenvalidierung und Maßnahmen zur Laufzeitoptimierung beschrieben.

- Kapitel 5: In diesem Kapitel wird auf Abhängigkeiten und Funktionalitäten der einzelnen Module eingegangen.
- Kapitel 6: Abschließend wird die Arbeit in diesem Kapitel kritisch bewertet und ein Überblick über mögliche Erweiterungen und Nachfolgearbeiten gegeben.

2. Anforderungsanalyse

Dieses Kapitel beschreibt die funktionalen Anforderungen, die sich daraus ergebenden Usecases und die nichtfunktionalen Anforderungen.

2.1. Funktionale Anforderungen

In diesem Abschnitt werden die funktionalen Anforderungen, aus denen sich die Usecases ableiten, genannt und beschrieben:

- Der Zugriff auf die Anwendung muss identitäts- und rollenbasiert beschränkt sein. Da es sich bei den Rohdaten um vertrauliche Daten handelt, muss der Zugriff auf die Anwendung zur Auswertung der Rohdaten sicher sein.
  Durch ein Login-Fenster, das beim Start der Anwendung aufgerufen wird, kann der beabsichtigte oder unbeabsichtigte unrechtmäßige Zugriff auf die Anwendung verhindert werden.

- Es müssen Emissionswerte aus einer Datenbank verarbeitet werden. Die Verarbeitung sieht Datenbankabfragen vor, die Daten für die gewählten Darstellungsformen liefern. Um eine geographische Zuordnung der Emissionen zu liefern, müssen die Emissionswerte pro Zelle aufsummiert werden. Um dem Benutzer eine tabellarische Auswertung der Ergebnisse zu ermöglichen, muss eine Abfrage realisiert werden, die die Emissionswerte (nach Startregionen und Emissionstyp gruppiert) aufsummiert.

- Der aktuelle Bearbeitungsfortschritt muss für den Benutzer nachvollziehbar sein. Die Rohdaten umfassen eine große Anzahl Datensätze. Somit kann die Laufzeit der Anfragen vom Benutzer als Absturz der Applikation interpretiert werden. Um das zu verhindern, muss dem Benutzer eine Rückmeldung über den Fortschritt der Abfragen gegeben werden.
  Die Bearbeitung der Abfrage findet atomar in der Datenbank statt. Somit ist aus Anwendungssicht kein Status über den Fortschritt der Abfrage zu ermitteln. Als Statushinweise an den Benutzer können demnach drei Zustände angezeigt werden:

(1.) Es wurde noch keine Abfrage gestartet, (2.) eine Abfrage läuft gerade und (3.) die Abfrage wurde abgeschlossen.

- Der Export der ermittelten Daten als Bericht soll möglich sein. Diese Berichte stellen Informationen in Form von Tabellen und graphischen Auswertungen nach einer klar definierten Vorlage dar. Somit können Ergebnisse verschiedener Auswertungen auf einfache Weise miteinander verglichen werden.
- Das Anbinden externer Datenbanken mit gleicher Struktur soll möglich sein. Somit soll die Möglichkeit bestehen, externe Datenquellen zu validieren. Ebenso kann Partnern des DLR die Möglichkeit gegeben werden, auf Basis eigener Rohdaten zu arbeiten, ohne dass die vertraulichen Rohdaten, die im DLR ermittelt wurden, übergeben werden müssen.

2.2. Usecases

Aus den funktionalen Anforderungen werden Usecases abgeleitet. Diese werden zunächst aufgelistet und dann nach einer Notation, die *Balzert* [8, Seite 126 ff.] nachempfunden ist, erläutert. Die Akteure sind der Standard-Benutzer und der Administrator. Es ist nicht vorgesehen, dass ein Benutzer sowohl die Rolle des Standard-Benutzers als auch des Administrators einnimmt.

Übersicht der Usecases

- Starten der Applikation
- Login
- Anlegen eines neuen Standard-Benutzers
- Bearbeiten eines Standard-Benutzers
- Löschen eines Standard-Benutzers
- Datenbankverbindung ändern
- Datenbankverbindung testen

- Berechnung starten
- Darstellen des Emissionskatasters auf der Weltkarte
- Export der Weltkarte
- Darstellung der Zusammenfassung
- Darstellung der detailierten Auswertung
- Export als Excel-Datei
- Beenden der Applikation

Starten der Applikation

- Ziel: Das Programm startet und die Benutzeroberfläche wird sichtbar.
- Akteur: Standard-Benutzer oder Administrator
- Ereignis: Die ausführbare Datei der Anwendung wird geöffnet.
- Vorbedingung: keine
- Nachbedingung (Erfolg): Dem Benutzer öffnet sich eine Oberfläche, um sich bei dem System anzumelden.
- Nachbedingung (Fehlschlag): Eine Fehlermeldung informiert den Benutzer über einen nicht erfolgreichen Programmstart.

Login

- Ziel: Der Benutzer meldet sich am System an, das Fenster für die Abfragedefinition oder die Benutzerverwaltung öffnet sich je nach Rolle des Benutzers.
- Akteur: Standard-Benutzer oder Administrator
- Ereignis: Der Benutzer klickt den Login-Button.
- Vorbedingung: Das Programm wurde erfolgreich gestartet und der Benutzername sowie das Passwort sind eingetragen.
- Nachbedingung (Erfolg): Hat der eingetragene Benutzer die Rolle eines Admi-

nistrators, öffnet sich die Benutzerverwaltung. Hat der Benutzer die Rolle des Standard-Benutzers, öffnet sich das Fenster zur Benutzereingabe.

- Nachbedingung (Fehlschlag): Eine Fehlermeldung weist den Benutzer auf falsche Eingaben hin, das Loginfenster bleibt geöffnet.

Anlegen eines neuen Standard-Benutzers

- Ziel: Ein neuer Standard-Benutzer wird angelegt, unter dessen Namen das Einloggen möglich ist.
- Akteur: Administrator
- Ereignis: Der Benutzer klickt den Button zum Erstellen des Benutzers.
- Vorbedingung: Ein Administrator ist angemeldet, der Bereich zum Erstellen eines Standard-Benutzers ist ausgewählt und ein Benutzername, sowie zwei mal das gleiche Passwort sind eingetragen.
- Nachbedingung (Erfolg): Der Standard-Benutzer wurde erfolgreich angelegt. Außerdem wird eine Statusmeldung darüber ausgegeben.
- Nachbedingung (Fehlschlag): Eine Fehlermeldung weist den Benutzer auf einen Fehlschlag hin. Der Benutzer kann seine Eingaben korrigieren.

Bearbeiten eines Standard-Benutzers

- Ziel: Das Passwort eines bestehenden Standard-Benutzers wird geändert.
- Akteur: Administrator
- Ereignis: Der Benutzer klickt den Button zum Bearbeiten des Standard-Benutzers.
- Vorbedingung: Ein Administrator ist angemeldet und der Bereich zum Bearbeiten eines Standard-Benutzers ist ausgewählt. Außerdem muss ein Benutzer aus einer Liste existierender Standard-Benutzer ausgewählt sein, sowie zwei mal das neue Passwort eingetragen sein.
- Nachbedingung (Erfolg): Der Standard-Benutzer wurde erfolgreich bearbeitet.

Außerdem wird eine Statusmeldung darüber ausgegeben.

- Nachbedingung (Fehlschlag): Eine Fehlermeldung weist den Benutzer auf einen Fehlschlag hin. Der Benutzer kann seine Eingaben korrigieren.

Löschen eines Standard-Benutzers

- Ziel: Ein bestehender Standard-Benutzer wird gelöscht.
- Akteur: Administrator
- Ereignis: Der Benutzer klickt den Button zum Löschen des Standard-Benutzers.
- Vorbedingung: Ein Administrator ist angemeldet und der Bereich zum Löschen eines Standard-Benutzers ist ausgewählt. Außerdem muss ein Benutzer aus einer Liste existierender Standard-Benutzer ausgewählt sein.
- Nachbedingung (Erfolg): Der Standard-Benutzer wurde erfolgreich gelöscht. Außerdem wird eine Statusmeldung darüber ausgegeben.
- Nachbedingung (Fehlschlag): Eine Fehlermeldung weist den Benutzer auf einen Fehlschlag hin.

Datenbankverbindung ändern

- Ziel: Als Zieldatenbank wird eine manuell eingetragene Datenbank ausgewählt.
- Akteur: Standard-Benutzer
- Ereignis: Der Benutzer klickt auf einen Button, um die Datenbankverbindung festzulegen.
- Vorbedingung: Der Benutzer muss eingeloggt sein, die Checkbox, die eine Standard-Datenbankverbindung festlegt, muss deaktiviert sein. Außerdem müssen in allen Feldern Werte eingetragen sein.
- Nachbedingung (Erfolg): Es wird eine manuell gewählte Datenbank verwendet.
- Nachbedingung (Fehlschlag): Eine Fehlermeldung weist den Benutzer auf einen Fehlschlag hin.

Die manuell gewählte Datenbank wird durch eine URL, die die Datenbank festlegt, einen Benutzernamen und ein Passwort definiert. Der Treiber für das Datenbanksystem muss vorliegen.

Datenbankverbindung testen

- Ziel: Die eingetragene Zieldatenbank wird auf Erreichbarkeit geprüft.
- Akteur: Standard-Benutzer
- Ereignis: Der Benutzer klickt auf einen Button, um die Datenbankverbindung zu testen.
- Vorbedingung: Der Benutzer muss eingeloggt sein, die Checkbox, die eine Standard-Datenbankverbindung festlegt, muss deaktiviert sein. Außerdem müssen in allen Feldern Werte eingetragen sein.
- Nachbedingung (Erfolg): Das Feld, das den Status der Datenbank anzeigt, informiert den Benutzer über die Verfügbarkeit der Datenbank.
- Nachbedingung (Fehlschlag): Eine Fehlermeldung und das Statusfeld weisen den Benutzer auf einen Fehlschlag hin.

Berechnung starten

- Ziel: Die Abfrage wird durchgeführt und speichert die in der Spezifikation festgelegten Ergebnisse zwischen.
- Akteur: Standard-Benutzer
- Ereignis: Der Benutzer klickt auf den Button, um die drei festgelegten Abfragen (Zusammenfassung, detailierte Auswertung, Ermittelung der Emissions für die Darstellung als Weltkarte) zu starten.
- Vorbedingung: Der Benutzer ist eingeloggt, die in der Spezifikation geforderten Werte (s.u.) sind in den dafür vorgesehenen Feldern eingetragen.
- Nachbedingung (Erfolg): Die in der Spezifikation festgelegten Werte werden zwi-

schengespeichert, Statusfelder zeigen an, ob die Berechnung läuft oder bereits abgeschlossen ist.

- Nachbedingung (Fehlschlag): Eine Fehlermeldung weist den Benutzer auf einen Fehlschlag hin. Die Statusfelder werden rot gefärbt.

Durch Klicken des Ausführungsbuttons startet der Benutzer die Abfragen. Es gibt mehrere Anwendungsfälle, die im Rahmen des Projekts *Clean Sky* realisiert werden. Diese Anwendungsfälle werden als *ATS-Usecases*[4] bezeichnet. Im Rahmen dieser Arbeit wird nur der Usecase ATS-Emission betrachtet. Dabei wird zwischen Flugzeugen und Hubschraubern als Emissionsverursuacher unterschieden. Diese Unterscheidung fließt als Vehicle Type in die Abfrageeinstellungen mit ein, wobei im Rahmen der Arbeit nur Flugzeuge betrachtet werden.
Wird das Jahr 2020 als Datenbasis ausgewählt, kann der Benutzer wählen, in welchem Maße alte Flugzeuge durch moderne (sog. *JTI-Flugzeuge*) ersetzt werden sollen. Dazu kann der Benutzer einen Ersetzungsfaktor wählen, durch den die Datengrundlage bestimmt wird.
Es werden folgende drei Prozesse durchgeführt:

1. Berechnung der Emissionen, die für die Darstellung als Weltkarte benötigt werden
2. Erstellung einer Zusammenfassung der eingegebenen Werte
3. Detailierte Analyse der Rohdaten anhand der Benutzereingabe

Für die Berechnung der Emissionen der Weltkarte wurden folgende Aktionen spezifiziert:

1. Statusmeldung an den Benutzer, dass die Berechnung gestartet wurde
2. Auslesen der ausgewählten Höhenklasse
3. Auslesen des ausgewählten Schadstoffes
4. Auslesen der ausgewählten Sitzklasse

[4]Ein *ATS-Usecase* ist nicht mit einem Usecase zu verwechseln, der in der Anforderungsanalyse betrachtet wird. Ist im weiteren Verlauf der Arbeit ein Usecase im Sinne der ATS-Ebene gemeint, wird dieser als *ATS-Usecase* bezeichnet.

5. Auslesen der ausgewählten Departure- und Arrivaleinstellungen
6. Ausführen einer Datenbankabfrage je nach Benutzereingabe
7. Zwischenspeichern der Ergebnisse
8. Statusmeldung an den Benutzer, dass die Berechnung abgeschlossen wurde

Mögliche Abfragekriterien für Departure und Arrival sind z.B. bestimmte Regionen (Nordamerika, Mitteleuropa etc.) oder bestimmte Länder. Grundlage der Einteilung in Regionen und Länder sind die monatlichen Flugpläne von *OAG Aviation Solution*.
Die zwischengespeicherten Ergebnisse beinhalten eine Hash Map, drei Objekte vom Typ File, sowie einige Zeichenketten. Die Hash Map besitzt als Key den Längen- und Breitengrad, sowie die zugehörige Emissionssumme als Value. Die drei File-Objekte, werden benötigt, um die Hintergrundkarte darzustellen. Als Zeichenketten sind der ausgewählte Schadstoff, die Höhenklasse, die Sitzklasse sowie Start- und Zieleinstellungen zur Information auf der Karte angegeben.
Für das Erstellen der Zusammenfassung werden einige Felder erneut ausgelesen und deren Inhalt zwischengespeichert. Dies begründet sich darin, dass zu jedem Ausgabefenster die zugehörigen Daten vollständig vorliegen sollen. Dadurch Daten redundant werden. Allerdings fördert es die Kapselung der Module. Des Weiteren werden einige zusätzliche Felder ausgelesen, um bereits durchgeführte Abfragen anhand dieser Angaben ausfindig machen zu können.
Im Rahmen weiterer Entwicklungen ist es vorgesehen, das exportierte Microsoft Excel-Dokument mit diesen Metadaten in eine Datenbank zu schreiben. Somit kann das mehrmalige Durchführen der gleichen Abfrage verhindert werden. Ein Zugriff auf die gesamten Ergebnisse ist über entsprechende Datenbankzugriffstools oder eine noch zu entwickelnde Hilfsapplikation möglich.

Zur Berechnung werden diese Schritte spezifiziert:

1. Statusmeldung an den Benutzer, dass die Erstellung gestartet wurde
2. Auslesen des ausgewählten *ATS-Usecases* (ATS-Emission)
3. Auslesen des eingetragenen Benutzers

4. Auslesen der eingetragenen Firma
5. Auslesen des Datums
6. Auslesen der eingetragenen Beschreibung
7. Auslesen des ausgewählten Jahres
8. Auslesen des ausgewählten Fahrzeugtypen
9. Auslesen der ausgewählten Höhenklasse
10. Auslesen des ausgewählten Schadstoffes
11. Auslesen der ausgewählten Sitzklasse
12. Auslesen des ausgewählten Ersetzungsfaktors
13. Auslesen der Departure- und Arrivaleinstellungen
14. Zwischenspeichern der Ergebnisse
15. Statusmeldung an den Benutzer, dass die Erstellung der Zusammenfassung abgeschlossen wurde

Alle ausgelesenen Werte werden zwischengespeichert und können weiterverarbeitet werden.
Die Ausführung einer detailierten Emissionsanalyse wird im Folgenden aufgezeigt:

1. Statusmeldung an den Benutzer, dass die Berechnung gestartet wurde
2. Auslesen der ausgewählten Höhenklasse
3. Auslesen der ausgewählten Sitzklasse
4. Ausführen einer Datenbankabfrage je nach Benutzereingabe
5. Zwischenspeichern der Ergebnisse
6. Statusmeldung an den Benutzer, dass die Berechnung abgeschlossen wurde

Im Rahmen der Entwicklung dieses Prototyps ist bei den Auswahlfeldern nur die Auswahl eines einzelnen Elements möglich. Nach erfolgreicher Berechnung werden die Buttons zur Darstellung der Ergebnisse aktiviert.

Darstellen des Emissionskatasters auf der Weltkarte

- Ziel: In einem separaten Fenster der Benutzeroberfläche wird eine Weltkarte mit aufgetragenen Emissionen dargestellt.
- Akteur: Standard-Benutzer
- Ereignis: Der Benutzer klickt auf einen Button mit der Aufschrift *show*, um die Weltkarte darstellen zu lassen.
- Vorbedingung: Die Abfrage, welche die Summe der Emissionswerte pro Zelle zwischenspeichert, wurde erfolgreich durchgeführt.
- Nachbedingung (Erfolg): Ein neues Fenster, das eine Weltkarte mit farblich aufgetragenen Emissionen beinhaltet, öffnet sich.
- Nachbedingung (Fehlschlag): Eine Fehlermeldung weist den Benutzer auf einen Fehlschlag hin.

Die durch die Abfrage ermittelten Ergebnisse sollen dargestellt werden. Die Darstellung soll in einem Fenster der Anwendung in Form einer Weltkarte erfolgen. Die Weltkarte soll die Kontinente erkennbar machen, sowie die Emissionen als entsprechend gefärbte Zellen der Ausmaße 1°x1° darstellen. Ebenfalls sollen die zuvor gewählten Abfragekriterien und eine Zuordnung der Farben zu den Emissionsmengen auf der Karte als Legende vermerkt sein.

Export der Weltkarte

- Ziel: Die erstellte Grafik der Weltkarte wird als Bilddatei exportiert.
- Akteur: Standard-Benutzer
- Ereignis: Der Benutzer klickt auf einen Button mit der Aufschrift *export*, um die Weltkarte zu exportieren.
- Vorbedingung: Die Berechnung wurde erfolgreich durchgeführt und die Weltkarte erfolgreich dargestellt.
- Nachbedingung (Erfolg): Eine Bilddatei der erstellten Grafik liegt auf dem Datei-

system vor.

- Nachbedingung (Fehlschlag): Eine Fehlermeldung weist den Benutzer auf einen Fehlschlag hin.

Der Export der Weltkarte ist möglich, wenn der Benutzer die Darstellung der Weltkarte aufgerufen hat. Dadurch ist ein Export nur nach vorheriger Plausibilitätskontrolle möglich, sodass versehentliche Fehleingaben erkannt werden. Es Exportformat ist JPEG vorgesehen.

Darstellen der Zusammenfassung

- Ziel: Es wird ein separates Fenster geöffnet, das eine Zusammenfassung beinhaltet.
- Akteur: Standard-Benutzer
- Ereignis: Der Benutzer klickt auf einen Button, um die Zusammenfassung darstellen zu lassen.
- Vorbedingung: Die Berechnung wurde erfolgreich durchgeführt.
- Nachbedingung (Erfolg): Ein neues Fenster, das eine Zusammenfassung beinhaltet öffnet sich und der Button für den Export als Microsoft Excel-Datei wird aktiviert.
- Nachbedingung (Fehlschlag): Eine Fehlermeldung weist den Benutzer auf einen Fehlschlag hin.

Darstellung der detailierten Auswertung

- Ziel: Es wird ein Fenster geöffnet, das detailierte Informationen zu den Emissionen in Form einer Tabelle beinhaltet. Diese Tabelle besitzt 5 Spalten. Die erste Spalte ist mit den Startregionen des OAG-Flugplans gefüllt. Die restlichen 4 Spalten beinhalten die jeweilige Summe der 4 Emissionen aller Flüge aus den jeweiligen Startregionen.
- Akteur: Standard-Benutzer

- Ereignis: Der Benutzer klickt auf einen Button, um die detailierte Auswertung darstellen zu lassen.
- Vorbedingung: Die Berechnung wurde erfolgreich durchgeführt.
- Nachbedingung (Erfolg): Ein separates Fenster, das eine detailierte Auswertung beinhaltet öffnet sich.
- Nachbedingung (Fehlschlag): Eine Fehlermeldung weist den Benutzer auf einen Fehlschlag hin.

Export als Excel-Datei

- Ziel: Eine Zusammenfassung der Abfrage, eine detailierte Auswertung sowie die Weltkarte werden in einem gemeinsamen Excel-Dokument exportiert.
- Akteur: Standard-Benutzer
- Ereignis: Der Benutzer klickt den Button mit der Aufschrift *Export als xls*, um die Ergebnisse zu exportieren.
- Vorbedingung: Die Abfragen wurden erfolgreich durchgeführt, die Weltkarte wurde erfolgreich dargestellt und exportiert, die Darstellung der Zusammenfassung und der detailierten Auswertung wurden zur Überprüfung aufgerufen.
- Nachbedingung (Erfolg): Eine Microsoft Excel-Datei mit den Ergebnisse liegt im Dateisystem vor.
- Nachbedingung (Fehlschlag): Eine Fehlermeldung weist den Benutzer auf einen Fehlschlag hin.

Das erstellte Dokument enthält für jede Darstellungsart (Zusammenfassung, detailierte Auswertung, Weltkartendarstellung) ein separates Tabellenblatt.

Beenden der Applikation

- Ziel: Das Programm wird beendet und die Benutzeroberfläche geschlossen.
- Akteur: Standard-Benutzer oder Administrator

- Ereignis: Der Schließen-Button wird gedrückt.
- Vorbedingung: keine
- Nachbedingung (Erfolg): Die Anwendung wird erfolgreich beendet.
- Nachbedingung (Fehlschlag): Eine Fehlermeldung informiert den Benutzer über ein fehlerhaftes Beenden.

Schließt der Benutzer das Hauptfenster der Anwendung, wird er erinnert, eventuell noch keine Daten exportiert zu haben. Durch eine Bestätigung der Beendigung des Programms werden alle Fenster geschlossen.

2.3. Nichtfunktionale Anforderungen

In diesem Abschnitt werden die nichtfunktionalen Anforderungen beschrieben.

2.3.1. Intuitive Anwendbarkeit

Da zum Benutzerkreis auch wissenschaftliche Mitarbeiter gehören, denen Erfahrung im Umgang mit konsolenbasierten oder komplexen Benutzeroberflächen fehlt, ist es eine Anforderung, eine intuitiv bedienbare Benutzeroberfläche bereitzustellen. Durch den Einsatz einer graphischen Benutzeroberfläche kann man die Probleme einer konsolenbasierten Anwendung umgehen. Weitere Hilfe bietet das Ausgrauen inaktiver Fenster. Die Installation der Anwendung soll möglichst einfach durchführbar sein.

2.3.2. Minimierung der Reaktionszeiten

Es ist erstrebenswert, dem Benutzer die Ergebnisse ohne langen zeitlichen Verzug präsentieren zu können. Obwohl die Anwendung nicht als zeitkritisch zu sehen ist, können sich die Abfragezeiten negativ auf die Wahrnehmung des Benutzers auswirken. Die große Menge an Rohdaten macht Optimierungen an den Abfragen unumgänglich, da ansonsten inakzeptable Ausführungszeiten auftreten würden. Auf die durchgeführten Maßnahmen zur Optimierung wird in Abschnitt 4.6 genauer eingegangen.

2.3.3. Erweiterbarkeit

Die Anwendung soll leicht erweiterbar sein. Um dies zu gewährleisten, müssen die Abhängigkeiten der Module genau definiert sein. Somit werden Seiteneffekte bei Erweiterungen ausgeschlossen. Ein Problem, das auftreten kann, sind zyklische Abhängigkeiten.
Ist Modul A von Modul B abhängig, kann Modul A erst nach der Instanziierung von Modul B aufgerufen werden. Wäre Modul B gleichzeitig von Modul A abhängig, könnte keines der Module aufgerufen werden, da eine gegenseitige Abhängigkeit besteht. An diesem simplen Beispiel mit zwei Modulen lässt sich eine zyklische Abhängigkeit noch leicht erkennen und bei der Erweiterung der Anwendung entsprechend agieren.
Bei steigender Anzahl der Module können zyklische Abhängigkeiten über mehrere Module hinweg nicht so offensichtlich zu erkennen sein, sodass hier ein besonderes Augenmerk liegt.

2.3.4. Wartbarkeit

Für das System sind anerkannte Standards und Vorschriften einzuhalten. Es sollen bewährte Programmiersprachen und Datenbankschnittstellen berücksichtigt werden. Außerdem sollen zur Erleichterung der Wartbarkeit Softwarekomponenten und deren Bestandteile dokumentiert und innerhalb der Software kommentiert werden.

3. Konzeptionierung und Architekturmodell

Dieses Kapitel beschreibt die Konzeptionierung der Anwendung, einschließlich eines Architekturmodells mit Beschreibung.

3.1. NetBeans-Module-Suite

Die Anwendung wurde mit Hilfe der NetBeans IDE (Version 6.9) entwickelt. Die verschiedenen Module sind als *NetBeans Module-Suite* zusammengestellt. NetBeans ist eine weit verbreitete Entwicklungsumgebung, besonders im Bereich der Anwendungsentwicklung in Java. Da NetBeans von *Sun Microsystems*, einer Tochter der *Oracle Corporation*, entwickelt wird, kann von einer professionellen und fortlaufenden Weiterentwicklung der Entwicklungsumgebung ausgegangen werden.
Es ist vorgesehen, zur Datensicherung den Quellcode und weitere Resourcen in regelmäßigen Abständen auf einem Versionskontrollsystem-Server zu sichern. Die NetBeans IDE bietet die Möglichkeit, ohne die Installation weiterer Plugins Projekte mit einem Repository zu verbinden. Ein Repository dient dazu, Quellcode oder andere Dokumente zu speichern.
Ein weiteres Argument für den Einsatz der NetBeans IDE ist der integrierte GUI-Builder, der früher als Projekt *Matisse* bekannt war. Die zu entwickelnde Anwendung basiert auf der Java-Swing-Technologie. Die intuitive Anwendbarkeit des GUI-Builders erleichtert das Erstellen einer solchen Oberfläche und kann um Komponenten externer Bibliotheken erweitert werden. [9] Andere Entwicklungsumgebungen wie *Eclipse* bieten keinen vergleichbaren GUI-Builder.
Weitere Gründe, die zwar nicht im Rahmen dieser Arbeit zum Tragen kommen werden, aber in Folgearbeiten von Bedeutung sein können, sind die OSGi Interoperabilität[10] und die Unterstützung von REST-Webservices. [11]
Die zu entwickelnde Anwendung setzt sich aus mehreren Modulen zusammen. Die Verwaltung der Module übernimmt das *NetBeans Module System*. [12, Seite 35] Die

Grundidee der Module wurde vom *Java Extension-Mechanismus*[5] übernommen, der die Kern-API um weitere Funktionalitäten erweitert. Diese *Package Versioning Specification*[6] wird dazu benutzt, um Abhängigkeiten zwischen Anwendungsmodulen und Abhängigkeiten von Systemmodulen zu beschreiben und zu verwalten.
Eine Beschreibung eines Moduls, sowie die Abhängigkeiten von anderen Modulen werden in einer Manifest-Datei (Module Manifest) festgehalten. Außer dieser Manifest-Datei benötigen Module keinen speziellen Installationscode. Eine weitere Konfigurationsdatei (layer.xml) stellt anwendungsspezifische Informationen bereit und definiert die Integration eines Moduls in die Plattform. Somit wird spezifiziert, was das Modul der Plattform hinzufügt (Menüeinträge, Services etc.). Die Beschreibung der Module erfolgt deklarativ, es ist kein Eingriff in die Applikationslogik nötig. Aus Sicht der Entwicklungsumgebung ist ein Modul ein eigenständiges Projekt, aus dem beim Build ein JAR-Archiv generiert wird. Dieses Archiv enthält die Manifest- und Layerdatei, die kompilierten Klassen sowie andere Ressourcen (z.B. Icons). [12]
Der maßgebliche Grund, das Projekt auf Modulbasis zu entwickeln, ist die Kapselung, wodurch eine leichte Erweiterbarkeit erreicht wird. Wenn man z.B. die abgefragten Daten mit Hilfe verschiedener Bibliotheken visualisieren will, kann man dafür jeweils ein Modul erstellen. Diese Module können unabhängig voneinander aufgerufen und weiterentwickelt werden. Das ist insbesondere bei großen Projekten und Projekten mit hohem Erweiterungspotential ein Vorteil.

3.2. MVP-Pattern

Es wurde eine Architektur gewählt, die an das Designpattern Model-View-Presenter (MVP) angelehnt ist. Model-View-Presenter ist aus dem Pattern Model-View-Controller (MVC) hervorgegangen.
Das MVC-Pattern hat drei Hauptkomponenten: Model, View und Controller. Das Mo-

[5] Der *Java Extension-Mechanismus* erlaubt es der Laufzeitumgebung Erweiterungsklassen zu finden und zu laden, ohne dass sie im Classpath genannt werden müssen. [13]

[6] Durch die B*Package Versioning Specification* können Inkompatibilitäten zwischen Resourcen erkannt werden. [14]

del repräsentiert die darzustellenden Daten und Werte. Die View beinhaltet die Darstellung der Daten, also die für den Benutzer sichtbare Oberfläche. Der Controller nimmt Interaktionen des Benutzers mit der Oberfläche entgegen und agiert dementsprechend. Durch eine Benutzereingabe kann der Controller z.B. veranlasst werden, Abfragen zu starten und die Ergebnisse an das Model zu übergeben. Im MVC-Pattern wird der View eine Instanz des Models übergeben. Dem Controller werden das Model und die View übergeben. Somit kann die View direkt auf das Model zugreifen. [15, Seite 4 ff.] Hier liegt der Unterschied zum MVP-Pattern. Im MVP-Pattern fungiert der Controller als sogenannter *Presenter*[7] und entscheidet wie die Daten präsentiert werden. Dem Controller sind sowohl Model, als auch View bekannt. Bei Änderungen am Model kann der Controller diese Daten auslesen und in die View schreiben. Ein Zugriff der View auf das Model ist somit nicht möglich. Diese Vorgehensweise bietet dem Controller die Möglichkeit auf externe Events zu reagieren, indem direkt die View angesprochen wird, ohne dass diese das Model aufrufen muss.
Folgende Grafik veranschaulicht den Zusammenhang der Module:

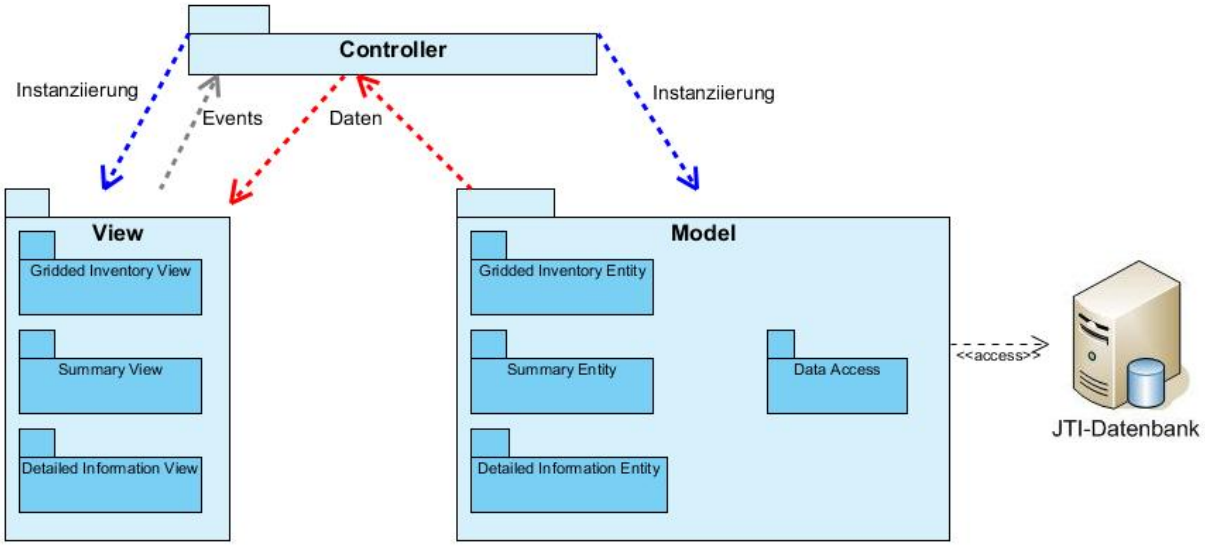


Abbildung 2: *Architekturmodell der Anwendung*

Die Funktionalitäten der einzelnen Module sind im folgenden Abschnitt erklärt.

[7] Aufgrund der ähnlichen Funktionsweise werden die Begriffe *Presenter* und *Controller* im Folgenden synonym verwendet.

3.2.1. Model-Komponenten

Das Model wird durch das Modul DataModel umgesetzt und beinhaltet zwei Packages. Zum einen ein Package dao, in dem sich Klassen zum Datenbankzugriff befinden und zum anderen ein Package entity, das reine Datenklassen (Entities) enthält. Das Modul ist von keinem anderen Modul abhängig.
Durch die Datenbankzugriffsklasse (DatabaseAccessImpl) werden sämtliche Zugriffe auf die Datenbank ausgeführt. Dabei wird das Pattern des Data Access Object (DAO) umgesetzt. Dieses Pattern ist sinnvoll, wenn man den Datenzugriff in eine separate Klasse auslagern möchte. Man stellt der Anwendung eine API zur Verfügung, die alle Zugriffe auf externe Daten umsetzt. Aus Sicht der Anwendung wird auf ein Interface (DatabaseAccessIF) zugegriffen. Die dahinterstehende Implementierung ist gekapselt. Somit besteht die Möglichkeit, die Implementierung des Datenbankzugriffs zu ändern, ohne den Client anzupassen. Dieses Package beinhaltet genau ein Interface und genau eine Implementierung dieses Interfaces. Aus Gründen der Übersichtlichkeit wurden keine getrennten Module für Implementierung und Interface erstellt. Die Wahrscheinlichkeit, dass die Implementierung eines Interfaces ausgetauscht wird, ist als gering zu betrachten. Aufgrund der Zuordnung von Interface und Implementierung, die im Rahmen der Arbeit als fix anzunehmen ist, wurde die Umsetzung in einem Modul gewählt. [16, Seite 462 ff.]
Das Entity-Package enthält für die drei Ausgabetypen (Weltkarte, tabellarische Detailansicht, Abfragezusammenfassung) jeweils die dafür benötigten Daten. Für die Weltkartendarstellung werden folgende Objekte benötigt:

1. Eine Liste der Emissionswerte vom Typ HashMap⟨ String, Double⟩
2. Shape-Files zur Darstellung der Hintergrundkarte vom Typ File
3. Abfrageparameter zur Angabe auf der Karte (Jahr, Schadstoff, Sitzklasse, Höhenklasse, Abflug- und Ankunftseinstellungen) vom Typ String

Zur Darstellung der Tabelle mit detailierten Emissionsinformationen werden folgende Objekt benötigt:

1. Eine Liste der Startregionen vom Typ Vector⟨String⟩
2. Eine Liste der anzuzeigenden Emissionswerte vom Typ double[ ][ ]

Um eine Zusammenfassung über die Abfrage zu liefern werden folgende Objekte benötigt, die allesamt vom Typ String sind:

1. Metadaten über die Abfrage: UseCase, Name, Firma, Datum und Beschreibung
2. Gewählte Abfrageparameter: Jahr, Schadstoff, Sitzklasse, Höhe, Abflug- und Ankunftseinstellungen, Fahrzeugtyp, Ersetzungsfaktor

In der Zusammenfassung werden abweichend von der Weltkarte noch die Daten *Fahrzeugtyp* und *Ersetzungsfaktor* benötigt. Diese sind aus den aktuellen Rohdaten noch nicht abfragbar, sollen aber bereits in der Zusammenfassung enthalten sein.

3.2.2. Controller-Komponenten

Der Controller wird durch das Modul Controller umgesetzt und besteht aus einem Package (controller). Dieses beinhaltet ein Interface (ControllerIF) für den Controller, sowie dessen Implementierung (ControllerImpl). Der Controller wurde nach dem Designpattern *Frontcontroller* entworfen (vgl. 3.3.1).
Durch einen Lookup des Moduls Login wird der Controller instanziiert. Es wurde keine zufriedenstellende Möglichkeit gefunden, den Controller anderweitig zu instanziieren. Dieser besitzt Abhängigkeiten zum Modul DataModel, sowie zu vier weiteren Modulen, die Views repräsentieren. Diese vier Module sind: das Modul zur Benutzereingabe(InputView), das Modul zur Darstellung der Weltkarte(InventoryGridOutputViewer), das Modul für die detailierte Emissionsausgabe(DetailedInformationOutputViewer) sowie das Modul für die Zusammenfassung(SummaryOutputViewer).
Der Controller implementiert das Interface ActionListener. Bei Programmstart ist er als Action Listener beim Login-Button registriert. Nach erfolgreichem Login wird er als Action Listener für alle weiteren relevanten Buttons registriert. Somit wird das Eventhandling alleine vom Controller übernommen. Wie auf die einzelnen Events reagiert wird, beschreibt Abschnitt 5.7.

Des Weiteren beinhaltet die Controller-Klasse innere Klassen. Diese implementieren das Interface Runnable und überschreiben somit dessen Methode run. Es wurden drei zeitintensive Aktionen identifiziert, die in eigenen Threads ausgeführt werden. Somit wird verhindert, dass die Benutzeroberfläche blockiert. Ein Blockieren der Benutzeroberfläche würde das Anzeigen von Statusmeldungen über den Fortschritt der Ausführung verhindern. Das könnte vom Benutzer als ein Programmabsturz interpretiert werden. Durch das Erstellen folgender Threads wird das umgangen:

- CalculationThread: Diese innere Klasse führt die Berechnungen aus und setzt die Statusfelder entsprechend des Fortschritts der Berechnungen.
- PrintingMapThread: Der Export der Weltkarte wird in diesem Thread ausgeführt.
- PrintingSummaryThread: Der Exportvorgang des Microsoft Excel-Dokuments, welches die Zusammenfassung, die detailierten Ergebnisse und die Weltkarte enthält, wird in diesem Thread ausgeführt.

3.2.3. View-Komponenten

Es gibt fünf Module die als View fungieren, also eine grafische Oberfläche haben (Login, InputView, InventoryGridOutputViewer, DetailedInformationOutputViewer, SummaryOutputViewer). Die Oberflächen leiten sich von der Klasse TopComponent, einer Kernkomponente der *Window System API* von NetBeans, ab.
Eine besondere Stellung nimmt dabei das Login-Modul ein. Es besitzt als einziges Modul eine Abhängigkeit vom Controller-Modul und ist die einzige View-Komponente, die bei Programmstart aufgerufen wird. Das Login-Modul instanziiert durch einen Lookup den Controller, übergibt ihm die GUI-Elemente und registriert den Controller als Action Listener.
Bei allen anderen View-Komponenten geht das Lookup vom Controller aus, sodass dieser eine Instanz der GUI beinhaltet. Da zyklische Abhängigkeiten nicht auftreten dürfen, und der Controller als "Einstiegspunkt" für das Programm technisch nicht realisiert werden konnte, wurde es auf diese Weise durch das Login-Modul gelöst. Die Oberfläche des Login-Moduls besteht im wesentlichen aus einem Textfeld zur

Namenseingabe, einem Passwortfeld, einem Button zum Einloggen, einem Statusfeld zur Fehlerausgabe, sowie einem versteckten Button zum Schließen des Fensters. Auf die Funktion des versteckten Buttons wird in Abschnitt 5.2 genauer eingegangen. Der Ablauf des Login-Vorgangs ist in Abschnitt 5.7 beschrieben.
Folgende Grafik zeigt eine Skizze der Aufteilung der Benutzeroberfläche:

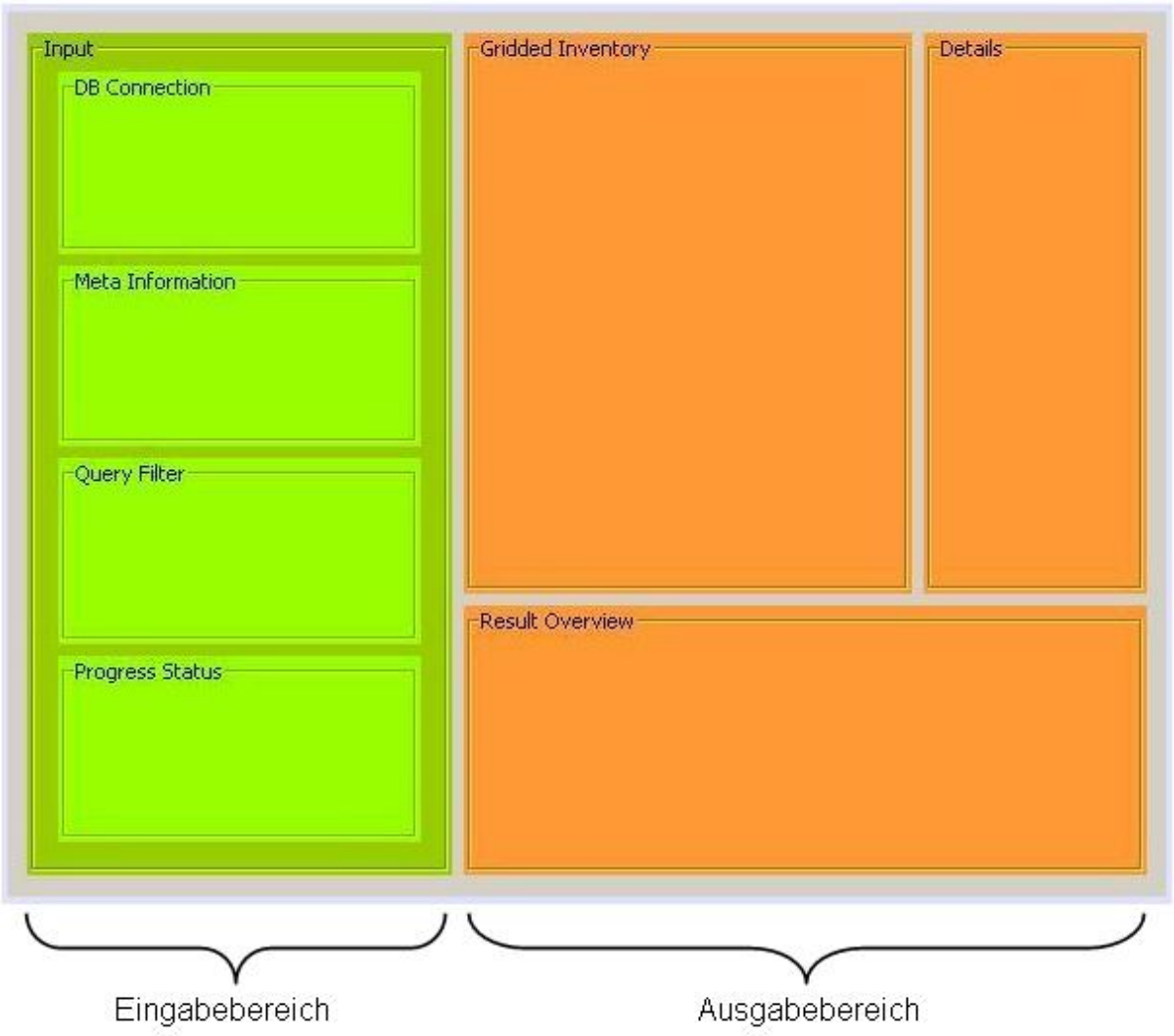


Abbildung 3: *Skizze der Benutzeroberfläche*

Auf der linken Seite in grünlichem Farbton ist das Modul InputView für die Benutzereingabe dargestellt. Die verschiedenen Module zur Ergebnisausgabe (InventoryGridOutputViewer, DetailedInformationOutputViewer, SummaryOutputViewer) sind auf der rechten

Seite in orangenem Farbton skizziert. Die einzelnen Komponenten der Module sind im Folgenden beschrieben.

Bei erfolgreichem Login öffnet sich das Modul, in dem die Benutzereingabe erfolgt (InputView). Die Oberfläche dieses Moduls lässt sich in vier Bereiche einteilen:

- Einstellung der Datenbankverbindung (DB Connection)
- Definition der Meta-Informationen (Metadata Information)
- Definition der Abfragekriterien (Query Filter)
- Prozess-Monitoring (Progress Status)

Der Bereich, in dem die Datenbankverbindung festgelegt wird, beinhaltet im wesentlichen Textfelder zur Eingabe von URL und Benutzername, ein Passwortfeld zur Passworteingabe und Buttons zum Übernehmen und Testen der Datenbankverbindung. Der Controller ist für den Button als Action Listener registriert.
Die Metadatendefiniton besteht aus einem Auswahlfeld für einen *ATS-Usecase*[8], sowie Textfeldern für den Namen und die Firma des Benutzers, das aktuelle Datum und einer Beschreibung der durchzuführenden Abfrage. Das Datumsfeld ist nicht veränderbar und zeigt die aktuelle Systemuhrzeit an.
Die Abfragedefinition enthält Auswahlfelder für Jahr, Fahrzeugtyp, Höhenklasse, Schadstoff, Sitzklasse und Ersetzungsfaktor. Dabei sind im Rahmen der Arbeit Rohdaten für das Jahr 2000 mit Fahrzeugtyp Aircraft und ohne Ersetzungsfaktor vorhanden. Des Weiteren gibt es bei der Auswahl von Start und Ziel drei Möglichkeiten:

- keine Einschränkung
- Einschränkung auf eine bestimmte Region
- Einschränkung auf ein bestimmtes Land

Bei dem Button zum Starten der Berechnung wurde der Controller als Action Listener registriert.
Der Monitoring Bereich informatiert den Benutzer über den Fortschritt der Berech-

[8] vgl. Abschnitt 2.2, Usecase: Berechnung starten

nungen, und erlaubt es dem Benutzer, die Ergebnisse darzustellen und zu exportieren. Außerdem wird der Benutzer über den Fortschritt des Exports informiert. Dazu sind für die Weltkartendarstellung, die Zusammenfassung und die detailierte Auflistung jeweils ein Statusfeld für die Berechnung und ein Button zur Darstellung vorhanden. Des Weiteren sind ein Button zum Exportieren, sowie ein Statusfeld zum Verfolgen des Fortschritts vorhanden. Der Controller ist bei diesen Buttons ebenso als Action Listener registriert.
Die folgenden drei Module dienen der Ausgabe der Ergebnisse. Zur Trennung der reinen GUI-Elemente von der Aufbereitung der Daten wurde das Modul in zwei Packages (util, view) unterteilt. Die Aufbereitung wurde in Hilfsklassen im Package util ausgelagert.

Das Modul für die Darstellung der Weltkarte enthält drei Packages. Ein Package (layer) enthält die verschiedenen Schichten[9], die für die Darstellung der Weltkarte benötigt werden. Ein weiteres Package (util) enthält eine Hilfsklasse, in der verschiedene Methoden zum Erstellen der Schichten enthalten sind. Das Package view beinhaltet die von TopComponent abgeleitete Klasse, welche die Benutzeroberfläche beinhaltet. Auf die einzelnen Schichten, sowie den Inhalt der Hilfsklasse wird in Abschnitt 5.4 genauer eingegangen.

Das Modul SummaryOutputViewer zur Darstellung der Zusammenfassung besteht aus zwei Packages. Ein Package (util) enthält eine Hilfsklasse mit Methoden, die zum Befüllen der Benutzeroberfläche benötigt werden. Das zweite Package (view) beinhaltet die Klasse, die die Oberfläche selbst darstellt. Als Oberflächenelemente beinhaltet das Modul für die Zusammenfassung unveränderliche Textfelder, die folgende Werte darstellen: Usecase, Benutzer, Firma, Datum, Beschreibung, Jahr, Fahrzeugtyp, Höhe, Schadstoff, Sitzklasse, Ersetzungsfaktor, Start- und Zielwahl. Auf die Implementierung des Moduls wird in Abschnitt 5.5 eingegangen.

[9]Diese Schichten sind nicht mit den Schichten einer Systemarchitektur zu verwechseln. An dieser Stelle sind Schichten als Bestandteil der Bibliothek OpenMap gemeint.

Des Weiteren besteht das Modul DetailedInformationOutputViewer zur Ausgabe detailierter Emissionsanalyse-Ergebnisse aus zwei Packages. Diese sind ähnlich der anderen Ausgabemodule. Eines enthält eine eine Hilfsklasse (util) und ein weiteres die Oberfläche (view). Die Hilfsklasse DetailedInformationUtility beinhaltet Methoden, um die Kopfzeile und den Inhalt der Tabelle darzustellen. Die Oberfläche selbst beinhaltet nur eine Tabelle. Eine Einsicht in die Implementierung ist in Abschnitt 5.6 gegeben.
Zusätzlich zu den hier genannten Packages und Klassen enthält jedes Modul ein Package exchange, das zum Austausch mit dem Controller dient. Es wurde programmiertechnisch keine Möglichkeit gefunden, ein Lookup auf eine von TopComponent abgeleitete Klasse auszuführen. Dies wurde durch eine Exchange-Klasse gelöst. Auf diese Klasse kann ein Lookup ausgeführt werden. Sie enthält Methoden zum Erstellen und Öffnen der Oberfläche, sodass faktisch das gleiche wie bei einem direkten Lookup auf die von TopComponent abgeleitete Klasse geschieht.

3.3. Besonderheiten des Controllers

Da der Controller eine zentrale Rolle einnimmt, wurde hier ein besonderes Augenmerk auf das Design gelegt. Die Besonderheiten werden in diesem Abschnitt erläutert.

3.3.1. Controller als Frontcontroller

Der Controller wurde nach dem Entwurfsmuster *Frontcontroller* umgesetzt. Dieses Pattern ist dann geeignet, wenn man einen zentralisierten Zugangspunkt für das Eventhandling der Benutzeroberflächen haben möchte. Verwendet man mehrere Controller, ist es oftmals der Fall, dass gewisse Funktionalitäten mehrfach oder über mehrere Controller verteilt sind. Auch können sich Änderungen auf mehrere Klassen auswirken, was zu Wartungs- und Erweiterungsproblemen führen kann. Das Verwenden eines Frontcontrollers kann sich ab einer gewissen Größe nachteilig auf die Übersicht und Wartbarkeit auswirken. Die im Rahmen der Arbeit zu erwartende Größe wird diesbezüglich nicht als kritisch angesehen. Zusammengefasst sind die Gründe für den Einsatz eines

Frontcontrollers:

1. Doppelte Kontrolllogik wird vermieden.
2. Eine einheitliche Logik wird für vielseitige Events realisiert.
3. Die Anwendungslogik ist von der graphischen Aufbereitung getrennt.
4. Es gibt einen zentralen Zugriff auf das System.

[16, Seite 166 ff.]

3.3.2. Controller und das Observer-Designpattern

Controller und View-Komponenten setzen in dieser Konstellation das Designpattern *Observer* um. Im Observer-Designpattern kann ein Subjekt mehrere Observer warten, indem deren Methoden aufgerufen werden. Auf die erstellte Anwendung umgesetzt, entspricht der Controller dem Observer und stellt öffentliche Methoden zur Verfügung. [15, Seite 293 ff.]
Durch dieses Pattern hält man sich die Möglichkeit offen, bei künftigen Erweiterungen um weitere Module den Controller als Eventhandler zu benutzen. Dieser nimmt Events externer Subjekte entgegen und kann entscheiden, wie auf diese reagiert wird.

4. Datenbank

In diesem Kapitel wird auf das Thema Datenmodellierung und Datenverwaltung eingegangen. Das umfasst die Wahl eines passenden Datenbankmanagementsystems und das Erstellen eines geeigneten Entity-Relationship-Modells für die Rohdaten, sowie den Entwurf von Datenobjekten. Außerdem werden die Vorbereitungen (Import und Validierung) und Optimierungsmaßnahmen erläutert.

4.1. Verwenden von MySQL

Als Datenbankmanagementsystem (DBMS) wurde MySQL verwendet. MySQL gehört seit Januar 2010 zur *Oracle Corporation*. [17] Gründe für den Einsatz von MySQL sind zum einen die Verfügbarkeit einer kostenfreien Community-Version und zum anderen die existierende Erfahrung der Abteilung mit diesem DBMS.

4.2. Struktur der vorliegenden Daten

Die zu verarbeitenden Daten liegen in unterschiedlichen Formaten vor. Wie die verschiedenen Formate in die Datenbank importiert wurden, ist in Abschnitt 4.4 erläutert. Die vorliegenden Daten sind folgende:

- Eine Textdatei, mit den Emissionswerten, die durch 4D-FATE errechnet wurden
- Eine Microsoft Access-Datenbank mit Flugplan-Informationen

Der Inhalt der Textdatei, die durch das Programm 4D-FATE berechnet wurde, ist in Abschnitt 1.4.2 beschrieben.
Dabei ist zu beachten, dass nur auf Grundlage dieser Datei noch keine Rückschlüsse auf die Flüge gemacht werden können, durch die die Emissionen verursacht wurden. Das wurde durch die Assoziierung jedes Emissionswerts mit einem zugehörigen Eintrag im Flugplan ermöglicht. Die dafür benötigte Microsoft Access-Datenbank besteht aus drei Tabellen:

- Flightplan: Enthält von OAG herausgegebene Flugplandaten

- DepAirportInfo: Enthält Daten zu den Startflughäfen
- ArrAirportInfo: Enthält Daten zu den Zielflughäfen

An dieser Stelle sind alle relevanten Spalten der Tabellen aufgezählt, um eine Übersicht über potentielle Abfragekriterien zu geben. Die Tabelle flightplan enthält folgende Attribute:

- Carrier1 (Fluggesellschaft)
- FlightNo1 (Flugnummer)
- DepAirport (Startflughafen)
- DepTerminal (Startterminal)
- ArrAirport (Zielflughafen)
- ArrTerminal (Zielterminal)
- LocalDepTime (Kodierte lokale Abflugszeit)
- LocalArrTime (Kodierte lokale Ankunftszeit)
- LocalDayOfOp (Kodierte Wochentage, an denen der Flug stattfindet, in lokaler Zeit)
- ArrDaysOfOp (Kodierte Wochentage, an denen der Flug landet, in lokaler Zeit)
- Service (Art der Dienstleistung)
- Seats (Gesamtanzahl der Sitze)
- FstSeats (Anzahl der Sitze der First Class)
- BusSeats (Anzahl der Sitze in der Business Class)
- EcoSeats (Anzahl der Sitze in der Economic Class)
- ElapsedTime (insgesamt vergangene Zeit)
- FlyingTime (Zeit, die sich das Flugzeug in der Luft befand)
- GroundTime (Zeit, die das Flugzeug am Boden verbrachte)
- Stops (Anzahl der Zwischenstopps)

- SpecificAcft (Genaue Flugzeugbezeichnung)
- Km (Zurückgelegte Entfernung in Kilometern)
- Frequency (Anzahl der durchgeführten Flüge im Monat)
- ID (entspricht der Flugnummer aus der Ergebnisdatei von 4D-FATE)

Die Tabellen DepAirportInfo und ArrAirportInfo liefern Informationen über Flughäfen. Diese Tabellen beinhalten redundante Daten. Die Spaltennamen unterscheiden sich lediglich im Präfix *Dep* (engl. departure) für Startflughafen, bzw. *Arr* (engl. arrival) für Zielflughafen. In der weiteren Betrachtung wird auf dieses Präfix verzichtet. Folgende Attribute der Tabelle sind relevant:

- AirportID (entspricht der Spalte DepAirport bzw. ArrAirport)
- AirportName (Name des Flughafens)
- City (Drei-Letter-Code der Stadt)
- CityName (Name der Stadt)
- IATACtryName (Name des Landes)
- RegName (Name der Region, redundant)
- Lat (Breitengrad des Flughafens)
- Long (Längengrad des Flughafens)

Die Region ließe sich aus dem Namen des Landes herleiten, wird jedoch redundant mitgeführt, um das Datenmodell nicht zu komplex zu gestalten. Die Daten bieten vielseitige Abfragemöglichkeiten.
Im Rahmen dieser Arbeit konnte nur ein Auszug der aufgezeigten Abfragekriterien in die Anwendung integriert werden. Das hatte zum einen den zeitlichen Horizont der Arbeit zum Grund. Zum anderen sind im Rahmen des Projekts *CleanSky* nur gewisse Abfragen benötigt.

4.3. Entity-Relationship-Modell

Im nachfolgenden Entity-Relationship-Modell sind nur die Attribute einbezogen, die für die Anwendung von Relevanz sind:

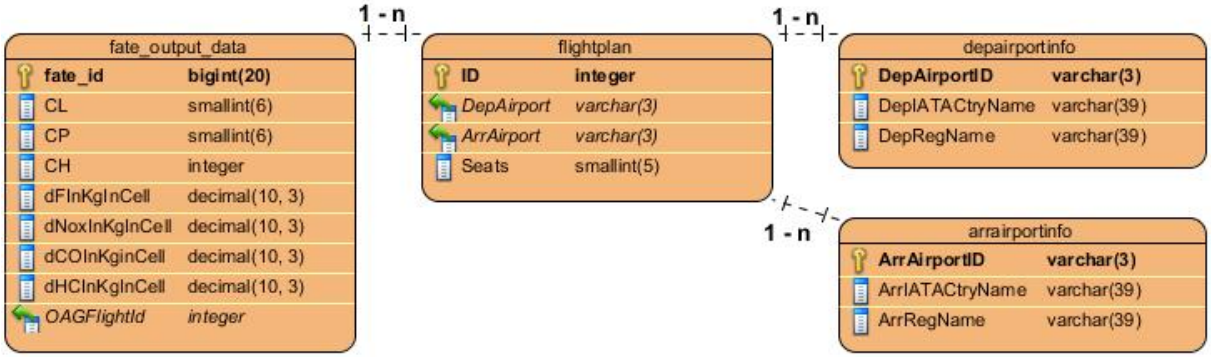


Abbildung 4: *Entity-Relationship-Modell der verwendeten Datenbank*

Für die Validierung der Daten waren Fremdschlüssel nötig. Im Zuge von Optimierungsmaßnahmen wurden Beziehungen durch Fremdschlüssel obsolet, da die Storage-Engine *MyISAM* benutzt wurde. Da nach der Datenvalidierung ausschließlich lesende Zugriffe auf die Datenbank vorgesehen sind, konnte auf den Einsatz einer transaktionssicheren Storage-Engine verzichtet werden (vgl. 4.6.1).
Es ist zu beachten, dass dieses Modell in der Datenbank im aktuellen Stand der Entwicklung einmal enthalten ist und die Daten für das Jahr 2000 enthält. Wenn die entsprechenden Daten vorliegen, werden weitere Tabellen nach diesem Modell eingefügt, die Daten für das Jahr 2020 beinhalten.
Des Weiteren gibt es zwei Tabellen, die davon unabhängig sind. Eine Tabelle beinhaltet die Daten der Benutzerverwaltung und hat folgende Attribute:

- ID (Identifikationsnummer als Primärschlüssel)
- Username (gewählter Benutzername)
- PasswordHash (SHA-1 Hashwert des gewählten Passwortes)
- Role (dem Benutzer zugewiesene Rolle)

Eine weitere Tabelle beinhaltet die Binärdateien, die zum Darstellen der Hintergrund-

weltkarte unabhängig vom lokalen Dateisystem benötigt werden. Diese enthält folgende drei Spalten:

- ID (Identifikationsnummer als Primärschlüssel)
- Name (Name der abgelegten Datei)
- File (Binärobjekt der Datei)

4.4. Importieren der Daten in die Datenbank

Die Datenbanktabelle, die die von 4D-FATE berechneten Ergebnisse beinhaltet, lag bei Beginn dieser Arbeit bereits vor.
Um die Tabellen, die den Flugplan und die Flughafeninformationen enthalten zu befüllen, stand eine Microsoft Access-Datenbank zur Verfügung. Um die Daten aus den Tabellen der Microsoft Access-Datenbank in Tabellen der MySQL-Datenbank zu schreiben, wurde auf Tools des *MySQL Migration Toolkit* zurückgegriffen. Es bietet die Möglichkeiten der unkomplizierten und intuitiven Migration von verschiedenen Datenbanksystemen (MS Access, Oracle Database Server usw.) auf eine Zieldatenbank.
Des Weiteren bietet es die Optionen, mehrere Tabellen in einem Programmdurchlauf, auch unter geänderten Namen zu speichern. Dabei können Spalten gezielt von der Migration ausgeschlossen werden.

4.5. Datenvalidierung

Um sicherzustellen, dass auf Grundlage einer korrekten Datenbasis gearbeitet wird, wurde eine Validierung der Daten durchgeführt. Diese Validierung bestand aus einer Abfrage, die feststellt, welche Flugplaneinträge nicht in der Rohdatentabelle vorkommen.
Dadurch lässt sich neben einer Validierung der Datengrundlage für die zu entwickelnde Anwendung, eine Validierung des Tools 4D-FATE vornehmen. Die Datengrundlage bestimmt maßgeblich die Aussagekraft der erzeugten Ergebnisse.
Kommt es bei der Abarbeitung der Inputdaten innerhalb des Programmes 4D-FATE

zu einem Fehler, wird der Datensatz übersprungen. Anhand der prozentualen Menge von übersprungenen Datensätzen lässt sich ein Maß für die Güte der Abarbeitung von 4D-FATE ermitteln.

Eine erste Datengrundlage bestand aus 9 Millionen Datensätzen in der Rohdatentabelle, was einem Datenvolumen von ca. 6 Gigabyte entspricht. Zur Validierung dieser Daten wurden drei SQL-Statements ausgeführt. Zunächst wurde die Gesamtzahl der Flugplaneinträge ermittelt. Folgende Anweisung lieferte als Ergebnis 372.341 Zeilen:

```
SELECT COUNT(*)
FROM flightplan;
```

Listing 1: Abfrage der Anzahl der Flugplaneinträge

Um festzustellen, welche Flugplaneinträge nicht in der Rohdatentabelle vorhanden waren, wurde folgende Abfrage ausgeführt:

```
SELECT f.ID
FROM flightplan f
WHERE f.ID NOT IN
(SELECT DISTINCT e.OAGFlightID
FROM fate_output_data e);
```

Listing 2: Abfrage der nicht berücksichtigten Flugplaneinträge

Als Ergebnis wurden 347.607 Zeilen ausgegeben. Des Weiteren wurde die Anzahl der insgesamt vorkommenden Flugplannummern in den Rohdaten abgefragt. Dazu wurde folgende Anweisung formuliert:

```
SELECT DISTINCT OAGFlightID
FROM fate_output_data;
```

Listing 3: Abfrage der vorkommenden Flugplannummern

Als Ergebnis wurden 24.734 Zeilen ermittelt, was genau der Differenz der beiden vorherigen Abfragen entspricht. Daraus konnten zwei Erkenntnisse gewonnen werden:

1. In den Rohdaten wird auf keine Flüge verwiesen, die nicht im Flugplan stehen.
2. Die Rohdaten beinhalten nur rund 10% der Daten, die laut Flugplan vorhanden sein müssten.

Die Daten der Ergebnisdatei von 4D-FATE waren nur unvollständig in der Datenbank (vgl. Abschnitt 4.4) enthalten. Um die grundsätzliche Funktionalität der Anwendung zu evaluieren, ist dieser Datenbestand ausreichend. Der vollständige Datenbestand wird nach dem Übertragen des Datenbankservers auf einen leistungsstärkeren Rechner zur Verfügung stehen. Dies ist erst nach Abschluss dieser Arbeit vollzogen.

4.6. Optimierungsmaßnahmen

Bei der zunächst angenommenen Datenmenge von 9 Millionen Zeilen als Rohdaten schwankten die Abfragezeiten ohne Optimierungsmaßnahmen zwischen 30 und 200 Sekunden. Da diese Abfragezeiten unangenehme Wartezeiten für den Benutzer bedeuten, wurden einige Optimierungsmaßnahmen durchgeführt. Mit der Erkenntnis, dass die angenommenen 9 Millionen Zeilen nur 10% der tatsächlichen Daten entsprechen, wurde klar, dass weitaus mehr Optimierungsarbeit betrieben werden muss.
An gleicher Stelle sei allerdings darauf hingewiesen, dass aufgrund des zeitlichen Rahmens nur grundlegende Performanceverbesserungen durchgeführt werden konnten. Weitere Maßnahmen zur Minimierung der Abfragezeiten sind im Kapitel "Zusammenfassung und Ausblick" beschrieben.

4.6.1. Anpassen der Storage-Engine

Aus der Tatsache, dass nach dem Erstellen der Datenbank nur lesend auf die Daten zugegriffen wird, wurde eine entsprechende Storage-Engine gewählt. In MySQL wird standardmäßig die Storage-Engine *InnoDB* verwendet. Diese unterstützt Fremdschlüssel und Transaktionen nach dem *ACID*-Prinzip. ACID beschreibt folgende vier Eigenschaften von Transaktionen:

- **A**tomicity: Transaktionen werden entweder komplett oder garnicht ausgeführt.
- **C**onsistency: Eine Transaktion führt eine Datenbank immer von einem konsistenten Zustand in einen anderen konsistenten Zustand.
- **I**solation: Transaktionen werden in ihrer Ausführung nicht durch parallele aus-

geführte Aktionen beeinflusst.

- **D**urability: Das Ergebnis einer Transaktion wird dauerhaft in der Datenbank gespeichert.

Diese Eigenschaften sind bei konkurrierendem, schreibendem Zugriff wichtig. Da im Rahmen der Anwendung nur lesender Zugriff auf die Rohdaten erforderlich ist, kann darauf verzichtet werden. Eine weitere Storage-Engine, die weder Fremdschlüssel, noch Transaktionssicherheit anbietet, ist die Engine *MyISAM*. Diese zeichnet sich durch höhere Performance als die *InnoDB*-Engine aus. Es konnte eine Performancesteigerung von 50% erreicht werden. [18, Seite 189 ff.]

4.6.2. Indizierung

Eine weitere Möglichkeit zur Performancesteigerung ist das Erstellen von Indizes. Indizes beschleunigen das Auffinden von Informationen bei Abfragen. [18, Seite 88 ff.] Es wurden aufgrund der zeitlichen Beschränkung der Arbeit und der großen Datenmenge nur grundlegende Betrachtungen bezüglich des Indexdesign angestellt. Dabei wurden Indizes auf die Fremdschlüsselattribute gelegt, um das Ausführen der Joins zu beschleunigen.
Auf weitere Möglichkeiten, die durch das Anlegen von Indizes bestehen, wird im Ausblick eingegangen.

4.6.3. Datentypänderungen

Die Emissionswerte in der Rohdatentabelle waren zu Beginn der Arbeit im Format MEDIUMTEXT gespeichert. Dies erforderte beim Aufruf der Summenfunktion eine Umwandlung vom Textformat in ein mathematisch verwertbares Zahlenformat. Abgesehen von der potentiellen Gefahr, dass hier Werte stehen könnten, die sich nicht als Zahlenwert interpretieren lassen, wird an dieser Stelle Performance eingebüßt. Daher wurde eine Umwandlung des Datentyps durchgeführt. Eine Umwandlung zum Datentyp DECIMAL(10,3) führte zu einer Performancesteigerung. Eine Referenzabfrage, die

alle CO_2-Emissionen unabhängig von sonstigen Parametern aufsummiert, konnte um ca. 13% beschleunigt werden.

4.6.4. Anlegen von Caches

Es besteht die Möglichkeit, diverse Zwischenspeicher (Caches) einzurichten. Wird eine Angabe benötigt, die sich noch im Zwischenspeicher befindet, wird sie direkt aus dem Zwischenspeicher bezogen. Das ist insbesondere in der Entwicklungszeit ein großer Vorteil, da somit Wartezeiten reduziert werden können.
Es sind zwei Caches zu betrachten:

- Query-Cache: Ein Query-Cache speichert lesende Abfragen mit dem zugehörigen Ergebnis ab.
- Key-Cache: Die Storage-Engine *MyISAM* bietet eine spezielle Struktur für Indexblöcke an. Diese Struktur wird Key-Cache genannt und enthält die meist verwendeten Indexblöcke.

5. Implementierung der Module

In diesem Kapitel wird die Implementierung der einzelnen Module beschrieben. Dabei wird an ausgewählten Stellen anhand von Quellcode-Ausschnitten und Screenshots genauer auf Besonderheiten eingegangen.
Die Oberflächen wurden auf Basis von Java Swing entwickelt. Abschnitt 5.1 beschreibt, welche Module Abhängigkeiten zu anderen haben. Die folgenden Abschnitte 5.2 bis 5.6 beschreiben die Implementierung der Module mit grafischer Oberfläche. Abschnitt 5.7 beschreibt die Funktionen des Controllers mit seinen Besonderheiten.
Der Abschnitt "Das DataModel-Modul" beschreibt die Umsetzung der Datenklassen und des DataAccess-Objekts. Bis die Funktionalität des DataAccess-Objektes in Abschnitt 5.8.2 genauer beschrieben wird, ist dieses als Blackbox anzusehen, welches die Ergebnisse von Datenbankabfragen liefert. Abschließend wird auf das Testen der Anwendung eingegangen.

5.1. Abhängigkeiten der Module

An dieser Stelle wird erläutert, wie sich die in Abschnitt 3.2 beschriebenen Abhängigkeiten mit Hilfe der NetBeans IDE umsetzen lassen. Dabei wird beispielhaft die Abhängigkeit zwischen Controller und Model betrachtet, wobei der Controller abhängig vom Model ist. Zunächst muss das Modul, zu dem eine Abhängigkeit definiert wird, einen Service anbieten. Dazu sind drei Aktionen erforderlich:

1. Einrichten der Lookup API[10]: Um einen Service anzubieten, muss für das Modul die Lookup API verfügbar sein. Diese lässt sich in den Projekteigenschaften im Unterpunkt *Libraries* einfügen, wie folgender Screenshot zeigt:

[10] Das Lookup ist eine zentrale Komponente der NetBeans Platform, deren Aufgabe es ist, Instanzen von Objekten zu verwalten. Module können über das Lookup sowohl Objekte bereitstellen, als auch Objekte aufsuchen.[12, Seite 127]

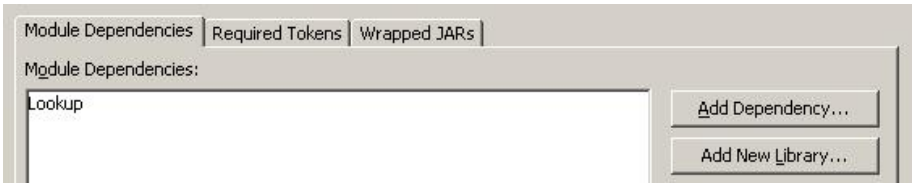


Abbildung 5: *Screenshot vom Einrichten der Lookup API*

2. Veröffentlichen der Packages: Um zu definieren, welche Packages für andere Module sichtbar sind, gibt es die Möglichkeit öffentliche Packages als *Public Packages* auszuwählen. Dies lässt sich im Unterpunkt *API Versioning* ausführen. Der nachfolgende Screenshot veranschaulicht dies:

Abbildung 6: *Screenshot vom Einrichten der öffentlichen Packages*

3. Zuletzt muss eine Klasse definiert werden, die durch einen Lookup aufgerufen werden kann. Diese Klasse wird über die Annotation ServiceProvider festgelegt, wie folgender Quellcode-Ausschnitt verdeutlicht:

```
@ServiceProvider(service = DatabaseAccessIF.class)
public class DatabaseAccessImpl implements DatabaseAccessIF {

  // Klasseninhalt ...

}
```

Listing 4: Definieren eines Service-Providers

Um die Abhängigkeit eines anderen Moduls zu definieren und auf den angebotenen Service zuzugreifen, müssen folgende Aktionen durchgeführt werden:

1. Einrichten der Lookup API: Um auf den angebotenen Service zuzugreifen, muss ein Lookup durchgeführt werden. Dazu muss die Lookup API verfügbar sein, wie in Abbildung 5 dargestellt.

2. Definieren der Abhängigkeit: Um ein anderes Modul bei einem Lookup finden zu können, muss es als Abhängigkeit (engl. Dependency) eingetragen sein. Dies kann ebenso wie die Lookup API im Unterpunkt *Libraries* der Projekteigenschaften eingestellt werden. Der nachfolgende Screenshot verdeutlicht dies:

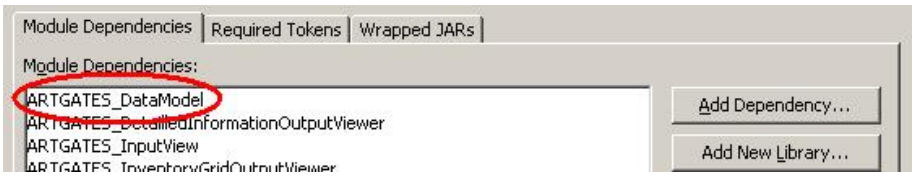


Abbildung 7: *Screenshot der Abhängigkeitsdefinition*

3. Ausführen des Lookups: Beim Lookup wird die Klasse, die einen Service anbietet anhand ihres Namens gesucht und instanziiert. Mit dieser Instanz kann weitergearbeitet werden. Folgender Quellcodeausschnitt verdeutlicht die Instanziierung und Verwendung der Klasse:

```
DatabaseAccessIF dao =
    Lookup.getDefault().lookup(DatabaseAccessIF.class);
dao.establishDBConnection();
```

Listing 5: Lookup und Verwenden eines Services

In gleicher Weise werden alle weiteren Abhängigkeiten von Modulen umgesetzt.

5.2. Login-Modul

Das Login-Modul (Login)ist vom Controller abhängig und instanziiert diesen. Im Konstruktor werden die Oberflächenelemente übergeben, sodass der Controller diese Auslesen kann. Des Weiteren ist der Controller für den Login-Button als Action Listener registriert.
Eine Besonderheit stellt ein versteckter Button dar, der für das Schließen dieses Fensters verantwortlich ist. Es wurde keine Möglichkeit gefunden auf eine Methode der Oberfläche direkt zuzugreifen, sodass sich diese Lösung anbietet. Die Methode actionPerformed() des versteckten Buttons sieht wie folgt aus:

```
private void loginHiddenCloseButtonActionPerformed
(java.awt.event.ActionEventevt){
    this.close();
  }
```

Listing 6: Methode actionPerformed() im Login-Modul

Der Aufruf der Methode doClick des versteckten Buttons im Controller bewirkt das Schließen des Fensters. Eine Vorschau des Login-Moduls, wie es im GUI-Builder erstellt wurde, sei an dieser Stelle gezeigt:

Abbildung 8: *Screenshot vom Login in der Vorschau*

Dabei ist zu beachten, dass der Button mit der Aufschrift *hidden* zur Laufzeit unsichtbar ist.

Zu dem eingetragenen Benutzer wird bei korrektem Passwort die zugewiesene Rolle abgefragt. Ist die Rolle des Standard-Benutzers (*user*) zugewiesen, verschwindet das Login-Fenster und die Oberfläche zur Abfragedefinition erscheint. Ist dem eingetragenen Benutzer die Rolle des Administrators (*admin*) zugewiesen, öffnet sich die Benutzerverwaltung. Diese bietet Funktionen zum Erstellen, Bearbeiten und Löschen von Benutzern an. Einem Benutzer können nicht beide Rollen zugewiesen werden. Die Oberfläche zum Erstellen eines neuen Benutzers ist hier als Beispiel dargestellt:

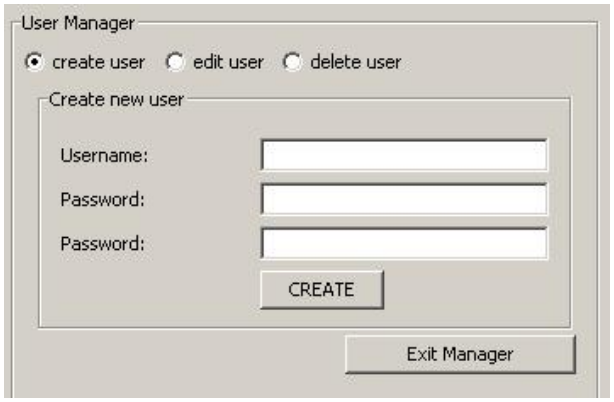


Abbildung 9: *Screenshot der Benutzerverwaltung*

Die Funktionen, die aufgerufen werden, wenn der Button geklickt wird, sind in Abschnitt 5.7 beschrieben.

5.3. Modul für die Benutzereingabe

Das Modul für die Benutzereingabe (InputView) beinhaltet, wie in Abschnitt 3.2.3 beschrieben, mehrere Felder, um die Abfrage zu definieren. Es setzt sich aus mehreren Panels zusammen, die jeweils einen Aufgabenbereich abdecken. Wofür diese zuständig sind, ist aus der Beschriftung und der Beschreibung in Abschnitt 3.2.3 ersichtlich. Um sich das bildlich vorstellen zu können, seien hier Screenshots der Anwendung dargestellt. Der Übersicht wegen wird im Folgenden jedes Panel einzeln betrachtet.

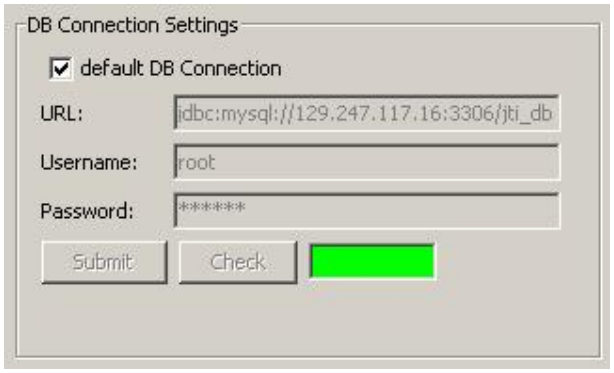


Abbildung 10: *Screenshot der Datenbankeinstellungen*

Eine Aktivierung bzw. Deaktivierung der Checkbox wirkt sich direkt auf die anderen

GUI-Elemente aus. Aufgrund der geringen Komplexität der Aktionen, die ausschließlich die Oberfläche betreffen, wurde festgelegt, das Eventhandling nicht durch den Controller ausführen zu lassen.
Die Metadaten-Definition besteht lediglich aus Eingabefeldern mit Beschriftung. Eine vollständige Eingabe von Werten ist erforderlich. Das Panel, das die Eingabeelemente beinhaltet, ist der Vollständigkeit halber hier abgebildet:

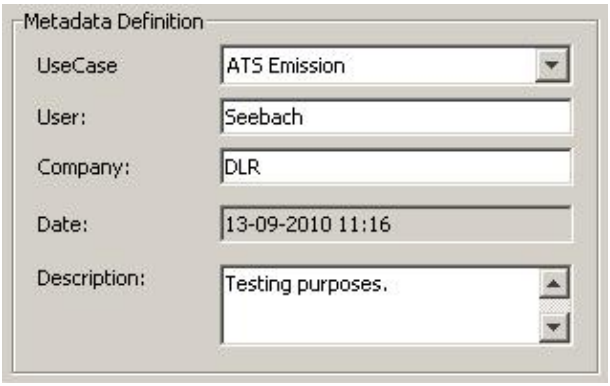


Abbildung 11: *Screenshot der Metadatendefinition*

Bei der Einstellung der Abfragedefinitionen kann der Benutzer aus mehreren Auswahlfelder seine Abfrage definieren. Die Departure- und Arrivaleinstellungen sind gesondert zu betrachten. Über drei Radio-Buttons, die zu einer Button-Group zusammengefasst sind, kann der Benutzer zwischen allen Flughäfen oder denen einer bestimmten Region oder eines bestimmten Landes wählen. Die Auswahlboxen für eine bestimmte Region oder ein bestimmtes Land sind nur bei Wahl des entsprechenden Radio-Buttons aktiviert.
Das Eventhandling wird an dieser Stelle vom Controller übernommen. Für den Grad an Komplexität, der im Rahmen dieser Arbeit erreicht wird, wäre es auch vertretbar, das in der GUI-Klasse selbst auszuführen. Da es jedoch nicht absehbar ist, wie sich diese Komplexität im Zuge weiterer Entwicklungen ändert, wurde das Eventhandling hier vom Controller übernommen. Die Oberfläche stellt sich wie folgt dar:

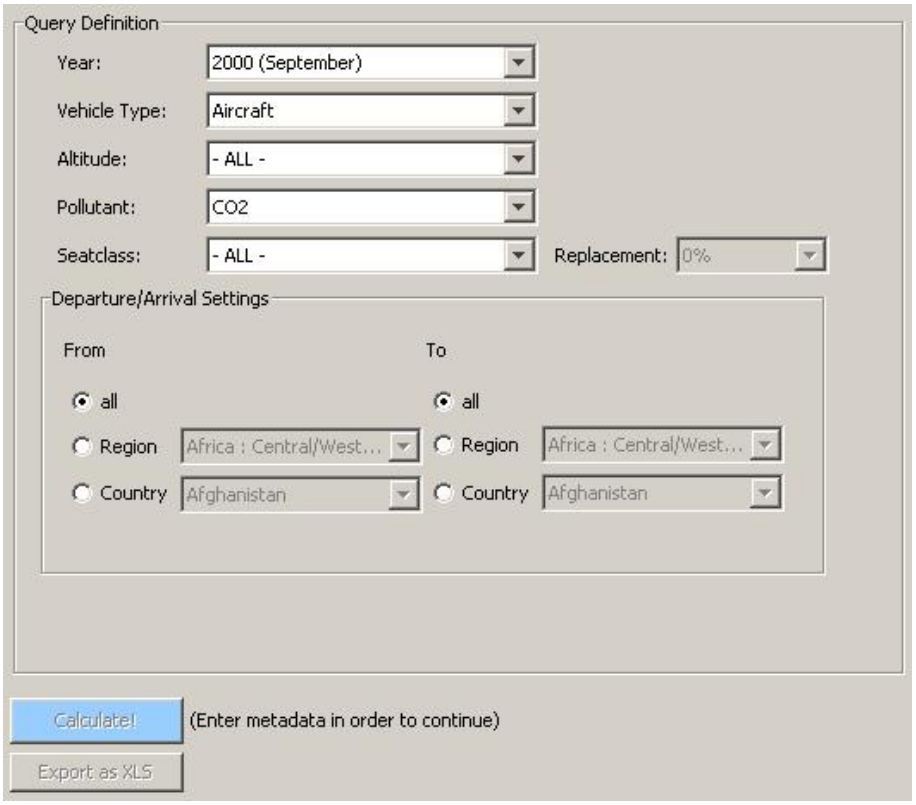


Abbildung 12: *Screenshot der Abfragedefinition*

Beim Klicken des Buttons mit der Aufschrift *calculate* wird die Berechnung angestoßen. Dieser Vorgang wird in Abschnitt 5.7 genauer beschrieben. Die Statusfelder des Progress-Panels informieren den Benutzer über den aktuellen Status der Abfrage. Folgender Screenshot zeigt es:

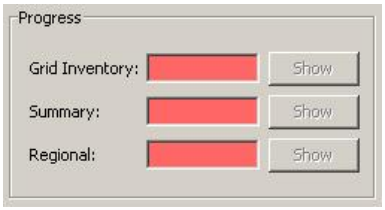


Abbildung 13: *Screenshot des Progress-Panels*

Die Buttons werden vom Controller nach erfolgreicher Berechnung aktiviert. Das Eventhandling dieser Buttons wird vom Controller übernommen. Bei erfolgreicher Darstellung der Ergebnisse wird der Export-Button aktiviert. Über den Fortschritt des Exports wird der Benutzer durch die Statusfenster informiert.

5.4. Modul zur Erstellung einer Weltkarte

Die Darstellung einer Weltkarte wurde mit Hilfe der OpenSource Bibliothek OpenMap implementiert. OpenMap beinhaltet zahlreiche Klassen, die von Java-Swing Komponenten abgeleitet sind. Diese können auf Basis geografischer Koordinaten angesprochen werden. Sie können Karten darstellen und je nach Benutzereingabe Daten und Darstellung verändern. Dabei lassen sich einer Weltkarte verschiedene Schichten hinzufügen und übereinander darstellen. [19]
Weitere Gründe für den Einsatz von OpenMap sind die Quelloffenheit, sowie die vorhandene Erfahrung im Umgang mit OpenMap. Somit konnten lange Einarbeitungszeiten vermieden werden.
Die darzustellenden Informationen wurden auf drei Schichten, die jeweils von einer Klasse repräsentiert werden, verteilt:

- MapLayer: Als Hintergrund wird ein Shape-File[11], das die Umrisse der Kontinente beinhaltet, eingebunden. Damit eine Klasse ein Shape-File darstellen kann, muss sie von der Klasse PlugInLayer von OpenMap abgeleitet sein. Einer solchen Plugin-Schicht lassen sich verschiedene Plugin-Arten hinzufügen.
  Ein Shape-File kann über eine Instanz der Klasse EsriPlugIn der Plugin-Schicht hinzugefügt werden. Das Esri-Plugin wird während der Berechnung der Emissionen erstellt und im Model zwischengespeichert. Ein detailierter Einblick ist im Abschnitt “Datenklassen” gegeben.

- EmissionLayer: Die Weltkarte wird um eine Schicht erweitert, welche die Emissionen darstellt. Die entsprechende Klasse ist von Layer, einer Klasse von OpenMap, abgeleitet. Die Emissionswerte werden in einer Hash Map im Konstruktor der Klasse übergeben. Diese Hash Map enthält Objekte vom Typ OMRect[12]. Diese repräsentieren Rechtecke der Größe 1°x1° dar und werden entsprechend des Emissionswertes eingefärbt.

[11] Das Dateiformat Shapefile ist ein von *Environmental Systems Research Institute* (ESRI)) entwickeltes Format für Geodaten

[12] OMRect ist ein Datentyp der Bibliothek OpenMap, mit dessen Hilfe vierseitige Polygone erzeugt werden können.

Auf die Erstellung der Hash Map wird nach der Beschreibung der Schichten genauer eingegangen. Erzeugten Rechtecke werden der Reihe nach ausgelesen und dargestellt.

- MetadataLayer: Als oberste Schicht werden Informationen über die durchgeführte Abfrage, sowie eine Legende dargestellt. Die dazugehörige Klasse ist von Layer abgeleitet. Ihr wird im Konstruktor eine Hash Map mit Meta-Daten übergeben. Diese wird ausgelesen und daraus Objekte vom Typ OMText[13] erstellt. Eine Legende, die eine Zuordnung von Farbe und Schadstoffmenge ermöglicht, wird ebenfalls durch Objekte vom Typ OMText umgesetzt.

Sowohl die Erstellung der einzelnen Schichten, als auch deren Zusammensetzung ist mit einigen Operationen verbunden. Um die eigentliche Darstellung von diesen vorbereitenden Operationen zu kapseln, wurde eine Hilfsklasse GridUtilities erstellt. Die Methoden dieser Klasse führen im wesentlichen folgende Operationen durch:

1. initColors(): Diese Methode füllt eine Hash Map, die bei der Einfärbung der Rechtecke aufgerufen wird mit Farben.
2. loadEmissionHashMap(): Aus dem Model wird die Hash Map mit Emissionswerten ausgelesen.
3. loadEsriPlugIn(): Aus dem Model werden die drei Dateien, die das Shape-File ergeben ausgelesen. Daraus wird ein Esri-Plugin erstellt.
4. processHashMap(): Diese Methode ist essentiell für die Darstellung der Weltkarte. Die in Schritt 2 eingelesene Hash Map wird verarbeitet. Aus den reinen Zahlenwerten werden Objekte erstellt, die sich geografisch zuordnen lassen. Dazu wird über jeden Längen- und jeden Breitengrad iteriert und ein Objekt der Klasse OMRect mit diesen Koordinaten instanziiert.
   Der Rahmen des Rechtecks ist transparent, sonst würde ein störendes Gitter entstehen. Anschließend wird das Rechteck in der Farbe eingefärbt, die dem Emissionswert zugeordnet ist und in einer Hash Map zur Weiterverarbeitung gespeichert.

[13] OMText ist ein Datentyp der Bibliothek OpenMap, der das Darstellen von Texten ermöglicht.

5. createMapLayer(): Es wird eine Instanz der Schicht zur Hintergrundkartendarstellung (MapLayer), wie oben beschrieben, erstellt. An diese wird das Esri-Plugin übergeben.

6. createEmissionLayer(): Eine Instanz der Emissions-Schicht (EmissionLayer) wird erstellt. An diese wird die Hash Map mit Objekten vom Typ OMRect übergeben.

7. createMetadataLayer(): Zunächst wird eine HashMap mit darzustellenden Metadaten erstellt. Als Key wird das Attribut benutzt, als Value wird ein Wert aus dem Model ausgelesen. Anschließend wird eine Instanz der Metadaten-Schicht (MetadataLayer) erstellt, der diese Hash Map übergeben wird.

8. createMapBean(): Diese Methode erstellt eine Instanz der Klasse MapBean von OpenMap. Diese ist eine Hauptkomponente von OpenMap und übernimmt die Verwaltung und Darstellung der Karten. Eine Karte besteht aus einer Projektion und einer Liste von Schichten.
   Die Klasse MapBean beinhaltet Methoden, um Projektionsparameter anzupassen und Schichten hinzuzufügen oder zu entfernen. Diese sind größtenteils auf Java AWT / Swing basierend, sodass das Zusammenspiel mit der Swing Oberfläche problemlos erfolgt. [20]

9. addLayersToBean(): Mit dem Aufruf dieser Methode werden die drei erstellten Schichten der Map Bean hinzugefügt, sodass die Map Bean die Verwaltung der Schichten übernimmt.

10. addMapBeanToPanel(): Diese Methode fügt die Map Bean einer Instanz der Klasse BasicMapPanel hinzu. Diese ist von JPanel abgeleitet und eine weitere Hauptkomponente von OpenMap. Sie eignet sich, um Daten, die mit OpenMap visualisiert werden in eine bestehende Applikation zu integrieren. [21] Es besteht die Möglichkeit, der Komponentenpalette des GUI-Builders eigene Komponenten hinzuzufügen. So ließ sich die Klasse BasicMapPanel der Palette hinzufügen und als Oberfläche des Moduls darstellen.

Diese Methoden werden vom Controller beim Erstellen dieses Moduls aufgerufen, sodass eine Weltkarte mit aufgetragenen Emissionen dargestellt wird.

Folgender Screenshot zeigt eine beispielhafte Ausgabe:

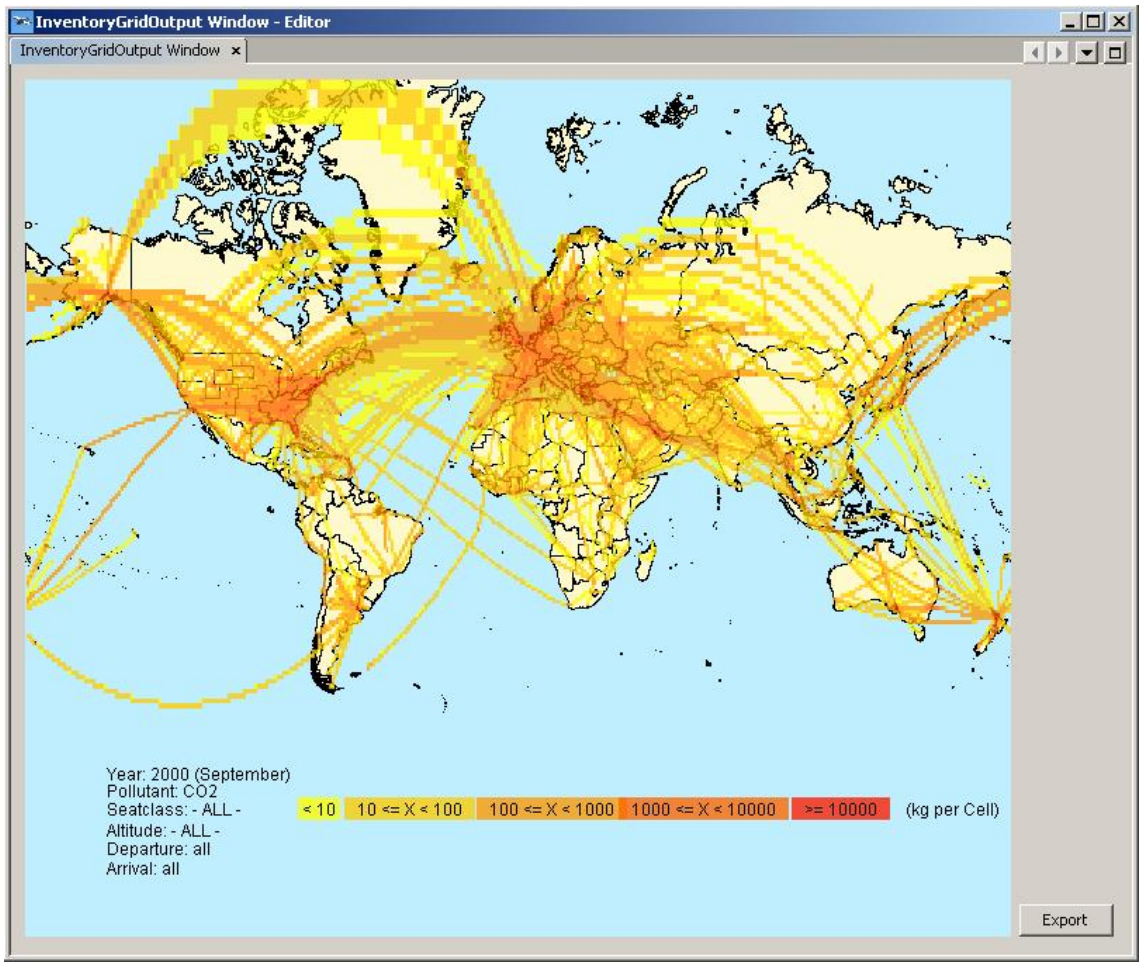


Abbildung 14: *Beispielhafte Ausgabe einer Weltkarte mit aufgetragenen Emissionen*

5.5. Modul zur Erstellung einer Zusammenfassung

Um die Zusammenfassung darzustellen, wurde eine grafische Oberfläche erstellt. Diese besteht ausschließlich aus Swing Komponenten vom Typ JPanel, JTextField, JTextArea und JLabel. Folgender Screenshot veranschaulicht den Aufbau der Oberfläche:

Das Befüllen der Oberfläche wird von einer Hilfsklasse ausgeführt. Diese liest das Model für die Zusammenfassung aus und trägt die Werte in die Textfelder ein.

Abbildung 15: *Beispielhafte Ausgabe einer Zusammenfassung*

5.6. Modul zur tabellarischen Darstellung der Ergebnisse

In ähnlicher Weise ist das Modul zur tabellarischen Darstellung der Emissionen (DetailedInformationOutputViewer) aufgebaut. Die Oberfläche besteht aus einer Tabelle vom Typ JTable. Dieser kann man im Konstruktor zwei Instanzen übergeben. Eine Instanz ist vom Typ String[ ] und beinhaltet die Kopfzeile der Tabelle, die zweite Instanz ist ein zweidimensionales Array vom Typ Object[ ][ ], welches die Zelleninhalte der Tabelle repräsentiert.

Ebenso wie in den anderen Ausgabemodulen, gibt es auch für die tabellarische Darstellung eine Hilfsklasse. Diese besitzt zwei Methoden, die jeweils eine der Instanzen liefert, die dem Konstruktor der Tabelle übergeben werden. Das Array, das die Kopfzeile der Tabelle beinhaltet, wird hart kodiert in der Methode eingebettet. Aufgrund der geringen Komplexität des Moduls ist das vertretbar.

Die Daten werden aus dem Datenmodel ausgelesen und in ein Array vom Typ Object[ ][ ] geschrieben. Dabei wird über ein Array mit Werten vom Typ Double iteriert und jeweils in ein in eine Instanz der Klasse Object umgewandelt.

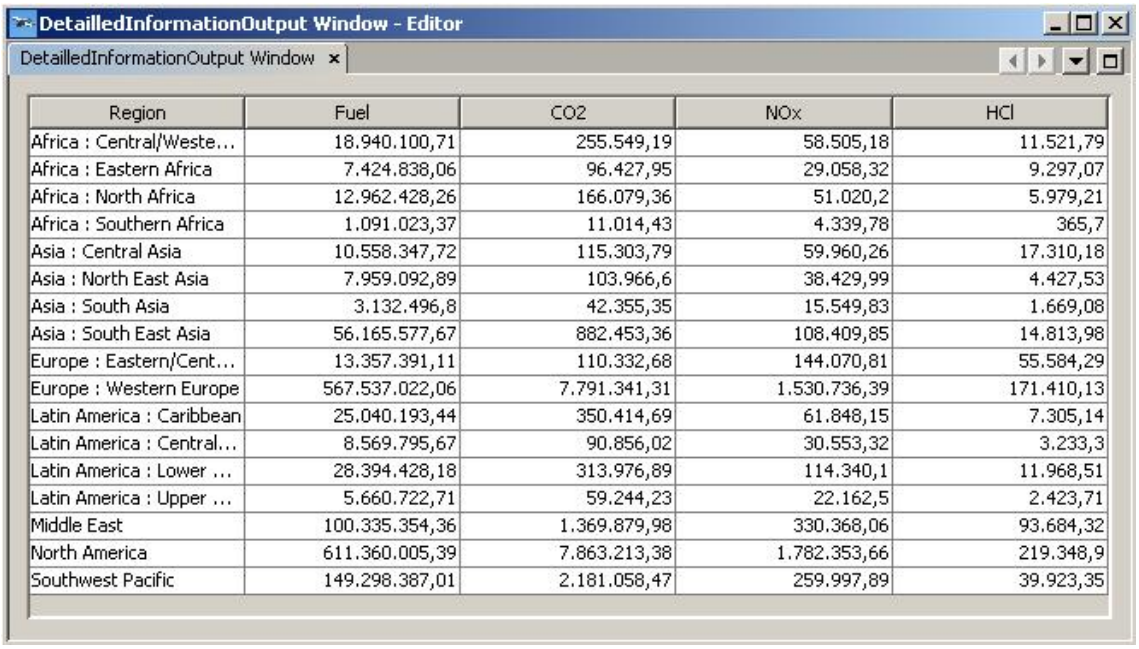
Bei diesem Modul wird die Methode zur Initialisierung der Oberfläche erst nach diesen einleitenden Funktionen ausgeführt.

Dazu ruft der Controller folgende Methode auf:

```
public void createAndFillTable(){
        util = new DetailledInformationUtility();
        dataO = util.getData();
        header = util.getHeader();
        initComponents();
    }
```

Listing 7: Laden und Erstellen der Tabelle zur detailierten Ansicht

Eine Abfrage der Emissionen auf alle Höhen- und Sitzklassen liefert folgendes Ergebnis:



Region	Fuel	CO2	NOx	HCl
Africa : Central/Weste...	18.940.100,71	255.549,19	58.505,18	11.521,79
Africa : Eastern Africa	7.424.838,06	96.427,95	29.058,32	9.297,07
Africa : North Africa	12.962.428,26	166.079,36	51.020,2	5.979,21
Africa : Southern Africa	1.091.023,37	11.014,43	4.339,78	365,7
Asia : Central Asia	10.558.347,72	115.303,79	59.960,26	17.310,18
Asia : North East Asia	7.959.092,89	103.966,6	38.429,99	4.427,53
Asia : South Asia	3.132.496,8	42.355,35	15.549,83	1.669,08
Asia : South East Asia	56.165.577,67	882.453,36	108.409,85	14.813,98
Europe : Eastern/Cent...	13.357.391,11	110.332,68	144.070,81	55.584,29
Europe : Western Europe	567.537.022,06	7.791.341,31	1.530.736,39	171.410,13
Latin America : Caribbean	25.040.193,44	350.414,69	61.848,15	7.305,14
Latin America : Central...	8.569.795,67	90.856,02	30.553,32	3.233,3
Latin America : Lower ...	28.394.428,18	313.976,89	114.340,1	11.968,51
Latin America : Upper ...	5.660.722,71	59.244,23	22.162,5	2.423,71
Middle East	100.335.354,36	1.369.879,98	330.368,06	93.684,32
North America	611.360.005,39	7.863.213,38	1.782.353,66	219.348,9
Southwest Pacific	149.298.387,01	2.181.058,47	259.997,89	39.923,35

Abbildung 16: *Beispielhafte Ausgabe einer tabellarischen Ansicht*

Das Bild zeigt eine Tabelle, das Emissionen aus den jeweiligen Startregionen aufsummiert als Zahlenwert darstellt.

5.7. Funktionalitäten des Controllers

In diesem Abschnitt wird erklärt, welche Funktionalitäten der Controller bereitstellt. Das beinhaltet das Eventhandling der Benutzeroberfläche, das Erzeugen von Threads, die Übergabe von Oberflächenelementen aus dem Login-Modul und das Starten von

Datenbankabfragen.

Da die Datenbankabfragen eine Kernfunktionalität der Anwendung sind, kann davon ausgegangen werden, dass eine Datenbankverbindung benötigt wird. Deshalb wird bereits im Konstruktor des Controller eine Datenbankverbindung hergestellt. Dazu wird über einen Lookup das DataAccess-Objekt instanziiert und die entsprechende Methode zum Einrichten einer Datenbankverbindung aufgerufen. Der genaue Methodeninhalt ist im Abschnitt "Umsetzung der DataAccessObjekt-Klasse" beschrieben. Des Weiteren wird bereits im Konstruktor das Systemdatum ausgelesen. Der Konstruktor des Controllers sieht damit wie folgt aus:

```
public ControllerImpl() {
        Date dt = new Date();
        SimpleDateFormat df = new SimpleDateFormat("dd-MM-yyyy HH:mm");
        df.setTimeZone(TimeZone.getDefault());
        dateString = df.format(dt);
        dao = Lookup.getDefault().lookup(DatabaseAccessIF.class);
        dao.establishDBConnection();
    }
```

Listing 8: Konstruktor des Controllers

In der Darstellung wurde aus Gründen der Übersichtlichkeit die Ausnahmenbehandlung nicht aufgeführt.

Der Controller implementiert das Interface ActionListener und muss somit die Methode zum Abfangen von Action-Events implementieren. In dieser wird je nach ActionCommand[14] des Events agiert. Die einzelnen Kommandos mit ihren Aktionen sind hier erläutert:

- calculate: Bei diesem Action-Command wird ein Thread für die Abfrage der Emissionswerte erstellt und gestartet. Eine Erläuterung dieses Threads ist weiter unten in diesem Kapitel zu finden.
- setconnection: Der Button zum Festlegen einer manuell eingetragenen Daten-

[14] Über das Attribut ActionCommand vom Typ String der Oberklasse AbstractButton lässt sich ermitteln, durch welchen Button ein Event ausgelöst wurde. Diese Attribut wird im Folgenden als *Action-Command* oder *Kommando* bezeichnet.

bankverbindung übergibt dieses Action-Command. Dabei werden die Textfelder, die die Datenbankverbindung spezifizieren ausgelesen und deren Werte an das DataAccess-Objekt übergeben, sodass für folgende Abfragen diese Datenbankverbindung genutzt wird.

- showgridinventory: Bei der Übergabe dieses Action-Commands wird das Modul InventoryGridOutputViewer über einen Lookup erstellt und mit geöffnet. Da der Code in ähnlicher Weise für die nächsten beiden Kommandos gilt, sei er an dieser Stelle aufgeführt:

```
if(e.getActionCommand().equals("showgridinventory")) {
            inventoryGridExchange = Lookup.getDefault().lookup(
            InventoryGridOutputExchangeIF.class);
            inventoryGridExchange.getTopComponent();
            inventoryGridExchange.loadAndCreateMap();
        }
```

Listing 9: Eventhandling durch Abfrage des Action-Commands

- showsummary: Bei diesem Action-Command wird analog zum vorher genannten Kommando das Modul SummaryOutputViewer erstellt und geöffnet.
- showregional: In gleicher Weise wird bei diesem Action-Command das Modul DetailedInformationOutputViewer erstellt und geöffnet.
- login: Dieses Action-Command wird vom Login-Button des Moduls Login aufgerufen. Dabei werden zunächst der eingegebene Benutzername und das eingegebene Passwort ausgelesen. Anschließend wird der in der Datenbank gespeicherte Hashwert zum eingetragenen Benutzernamen ermittelt. Über eine Hashfunktion wird aus dem eingetragenen Passwort der Hashwert ermittelt. Der im *RFC 3174* [22] beschriebene Hashalgorithmus SHA-1 wurde gewählt.
  Dazu wird eine Instanz der Klasse MessageDigest aus dem Package java.security verwendet. Dieser übergibt man eine Instanz vom Typ ByteArray, das die eingegebenene Zeichenkette repräsentiert, und ruft die Methode digest() auf. So lässt sich der Hashwert des eingegebenen Passwortes ermitteln.

Der zugehörige Quellcode ist hier aufgeführt:

```
StringBuffer enteredPWHash = new StringBuffer();
String enteredUsername = loginUsernameTextfield.getText();
char[] loginPWChar = loginPasswordField.getPassword();
String loginPWStr = "";
for (int i = 0; i < loginPWChar.length; i++) {
        loginPWStr += loginPWChar[i];
}
MessageDigest digest;
try {
    digest = MessageDigest.getInstance("SHA-1");
    digest.update(loginPWStr.getBytes());
    byte digestedPW[] = digest.digest();
    for (int i = 0; i < digestedPW.length; i++) {
            enteredPWHash.append(Integer.toHexString(0xFF &
                digestedPW[i]));
    }
}
```

Listing 10: Hashwertbildung zur Passwortüberprüfung

Ist der Hashwert des eingegebenen Passwortes verschieden von dem Hashwert des in der Datenbank gefundenen Passwortes, oder wurde kein Passwort zu dem eingegebenen Benutzernamen gefunden, werden die Textfelder zurückgesetzt und das Statuslabel informiert den Benutzer über eine fehlerhafte Anmeldung.
Bei korrekter Anmeldung wird überprüft, welche Rolle dem eingetragenen Benutzernamen zugeordnet ist. Ist der eingetragene Benutzer mit der Rolle des Standard-Benutzers assoziiert, wird das Modul InputView über einen Lookup instanziiert. Die Oberflächenelemente dieses Moduls werden dem Controller übergeben, ebenso wird der Controller bei allen relevanten Buttons als Action Listener registriert.
Außerdem werden einige der Oberflächenelemente mit Daten gefüllt. Die Auswahlfelder für die Departure- und Arrivaldefinition werden gefüllt.

Folgender Quellcode-Ausschnitt verdeutlicht das am Beispiel der Startregionen:

```
Vector<String> depRegions = dao.getAllDepRegions();
for(String depRegion : depRegions){
    depRegionComboBox.addItem(depRegion);
}
```

Listing 11: Hinzufügen von Elementen einer Auswahlbox

Zuletzt wird die Methode doClick() des versteckten Buttons des Login-Moduls aufgerufen. Dieses Event wird im Login-Modul selbst abgefangen und ruft eine Methode auf, die das Login-Modul schließt.
Ist der Benutzername mit der Rolle des Administrators assoziiert, öffnet sich die Benutzerverwaltung und die Elemente des Login-Bereichs werden deaktiviert. Auf die weitere Funktionalität der Benutzerverwaltung wird im Folgenden eingegangen.

- createUserRadio: Beim Aktivieren des Radio-Buttons mit diesem Kommando wird das Panel zum Erstellen eines Benutzers sichtbar. Die Panels zum Bearbeiten oder Löschen eines Benutzers werden unsichtbar.
- editUserRadio, deleteUserRadio: In gleicher Weise werden die entsprechenden Panels der Benutzerverwaltung sichtbar bzw. unsichtbar.
- createButton: Zum Erstellen eines neuen Benutzers wird dieses Kommando übergeben. Zunächst wird überprüft, ob das gleiche Passwort in den dafür vorgesehenen Feldern eingetragen wurde. Ist dies der Fall, wird überprüft, ob der Benutzername bereits exisitert und gegebenenfalls ein Hinweis ausgegeben. Bei korrekter Eingabe wird der Hashwert des Passwortes gebildet und gemeinsam mit dem neuen Benutzernamen an das DAO übergeben.
- editButton: In ähnlicher Weise kann ein Benutzername editiert werden. Dabei werden ebenso die eingetragenen Passwörter verglichen. Bei Gleichheit wird der Hashwert des Passwortes gebildet und dieser anschließend gemeinsam mit dem Namen an das DAO übergeben.
- deleteButton: Aus einer Liste existierender Benutzer wird der ausgewählte Benut-

zer an das DAO übergeben, das den entsprechenden Eintrag aus der Datenbank entfernt.

- exitManager: Durch das Klicken des Buttons mit diesem Action-Command wird das Panel der Benutzerverwaltung geschlossen. Die Elemente des Login-Fensters werden wieder aktiviert.

Im Folgenden werden die Funktionalitäten der Benutzereingabe anhand der Action-Commands beschrieben:

- checkConnection: Um zu testen, ob eine Verbindung zur Datenbank hergestellt werden kann, wird die entsprechende Methode aufgerufen. Tritt ein Fehler auf, wird das Statusfeld rot, ansonsten grün, gefärbt.
- depAll: Durch dieses Action-Command werden die beiden Auswahlfelder für Startland und -region deaktiviert.
- depRegion: Bei diesem Action-Command wird das Auswahlfeld für die Startregionen aktiviert und das für das Startland deaktiviert.
- depCountry: Ähnlich zum vorherigen Kommando werden die Auswahlfelder umgekehrt aktiviert bzw. deaktiviert.
- arrAll, arrRegion, arrCountry: Diese drei Action-Commands werden analog zu den vorherigen drei Action-Commands behandelt, jedoch betreffen sie die Zielregionen bzw. -länder.
- exportMap, exportSummary, exportDetails: Beim Aufruf dieser Action-Commands wird der entsprechende Thread zum Exportieren der Weltkarte (PrintinMapThread) oder als Microsoft Excel-Dokument (PrintingSummaryThread) erstellt und gestartet.

Der weiterer wesentlicher Bestandteil des Controllers sind die genannten Threads, die als innere Klassen umgesetzt wurden. Diese implementieren das Interface Runnable und müssen somit die Methode run() überschreiben.

PrintingMapThread

Dieses Beispiel veranschaulicht den Export der Weltkarte:

```
public class PrintingMapThread implements Runnable{
    @Override
    public void run() {
        exportMapStatusfield.setBackground(Color.ORANGE);
        exportMapStatusfield.setText("exporting...");
        File file = new File("C://temp//imgExport.jpg");
        BufferedImage img = new BufferedImage(892, 872,
            BufferedImage.TYPE_INT_RGB);
        basicmappanel.paint(img.createGraphics());
        ImageIO.write(img, "jpg", file);
        exportMapStatusfield.setBackground(Color.GREEN);
        exportMapStatusfield.setText("");
    }
}
```

Listing 12: Thread zum Exportieren der Weltkarte

Der Benutzer wird durch Manipulation des Statusfeldes über den Start des Exports informiert. Anschließend wird eine Instanz eines File-Objektes erstellt. Da die im Rahmen der Arbeit entwickelte Anwendung ein Prototyp ist, wurde an dieser Stelle der Dateipfad hart kodiert. Die Grafiken des Panels werden einem Objekt der Klasse BufferedImage übergeben. Die Klasse ImageIO schreibt diese in das File-Objekt.

PrintingExcelThread

Der Export als Microsoft Excel-Datei wurde ebenfalls in einen eigenen Thread ausgelagert. Um aus einer Java-Anwendung heraus auf Excel-Dokumente zuzugreifen, wurde sich für den Einsatz der *Java Excel API* (Jexcel) entschieden. Diese zeichnet sich durch intuitive Anwendbarkeit und Unterstützung aller benötigten Funktionalitäten aus. [23] An dieser Stelle werden die verwendeten Klassen von Jexcel mit kurzer Erläuterung vorgestellt, anschließend wird ein Quellcode-Beispiel aufgeführt.

- WritableWorkbook: Diese Klasse repräsentiert ein Excel-Dokument, auf das schreibend zugegriffen werden darf.
- WritableFont: Mit dieser Klasse wird eine Schriftart definiert.
- WritableCellFormat: Diese Klasse repräsentiert die Formatierung einer Zelle. Dazu kann u.a. eine Schriftart übergeben werden.
- WritableSheet: Durch diese Klasse wird ein einzelnes Tabellenblatt eines Excel-Dokuments beschrieben. Auf dieses Tabellenblatt kann schreibend zugegriffen werden.
- WritableImage: Um Bilder in ein Excel-Dokument einzufügen stellt Jexcel diese Klasse zur Verfügung.

Am folgenden Beispiel ist die Anwendung dieser Klassen zu sehen:

```
Workbook workbook = Workbook.createWorkbook(new
    File("C://temp//export.xls"));
WritableFont fontCaption = new WritableFont(WritableFont.ARIAL, 14,
    WritableFont.BOLD);
WritableCellFormat formatCaption = new WritableCellFormat(fontCaption);
WritableSheet sheetSummary = workbook.createSheet("Summary", 1);
sheetSummary.addCell(new Label(1, 2, "Summary", formatCaption));
workbook.write();
workbook.close();
```

Listing 13: Erstellen einer Microsoft Excel-Datei mit der Java Excel API

Diese Zeilen erzeugen ein Microsoft Excel-Dokument mit einem Blatt und einer in fetter Schrift dargestellter Zelle mit dem Inhalt "Summary". Auf die Darstellung der Ausnahmebehandlung wurde verzichtet. Außerdem informiert ein Statusfeld den Benutzer über den Fortschritt des Prozesses. Der Dateipfad des zu erstellenden Dokuments wurde hart kodiert, da es sich im Rahmen der Arbeit um die Entwicklung eines Prototyps handelt.

CalculationThread Die Methode run() des CalculationThread führt die Emissionsabfrage durch.

Beim Starten des Threads wird der Benutzer zunächst über das Statusfeld vom Start der Abfrage informiert. Es werden nun alle benötigten Schritte durchgeführt, um das Model für die Weltkartendarstellung (GriddedInventoryEntityImpl) zu füllen. Anschließend werden die ausgewählten Abfrageparameter ausgelesen.
Um die Start- und Zieldefinitionen abzufragen, wird ausgelesen, welcher Radio-Button ausgewählt ist. Ist der Radio-Button mit der Aufschrift *Region* oder *Country* gewählt, wird außerdem die ausgewählte Region bzw. das ausgewählte Land angefügt. Anschließend wird das Model über einen Lookup initialisiert. Das Datenfeld für die Emissionen wird gefüllt, indem die Methode getGenericEmissionPerCell() der DAO-Klasse aufgerufen wird. Dieser werden die Abfrageparameter als Zeichenketten übergeben. Im Rahmen der Arbeit sind das: Höhenklasse, Emissionstyp, Sitzklasse, Startparameter und Zielparameter. Wie die Methode die übergebenen Parameter weiterverarbeitet ist im Abschnitt "Umsetzung der DataAccessObject-Klasse" genauer beschrieben.
Des Weiteren werden über das DataAccess-Objekt die drei Dateien des Shape-Files der Hintergrundkarte ausgelesen und dem Model übergeben.
Zuletzt werden die für die Metadaten-Schicht erforderlichen Informationen aus den Oberflächenelementen ausgelesen und in das Model geschrieben. Bei erfolgreichem Durchführen der Aktionen wird der Benutzer durch das Statusfeld informiert und der Button zur Darstellung wird aktiviert.
Es folgen die Schritte, um das Model SummaryEntityImpl zu befüllen. Wieder wird der Benutzer über ein Statusfeld über den Fortschritt der Ausführung informiert.
Anschließend werden die in den Anforderungen beschriebenen Daten aus den Oberflächenelementen ausgelesen und in das Model eingetragen. Der Button für die Darstellung wird aktiviert. Darauf folgend werden die entsprechenden Schritte durchgeführt, um das Model für die tabellarische Darstellung (DetailedInformationEntityImpl) zu füllen.
Über das DataAccess-Objekt werden die Regionen, sowie die Emissionswerte je Region abgefragt und in die dafür im Model vorgesehene Variable geschrieben. Ein Statusfeld informiert den Benutzer über den Fortschritt, der Button zum Darstellen wird analog zu den vorherigen Schritten aktiviert.

5.8. Das Datamodel-Modul

Das Modul Datamodel beinhaltet zum einen je Ausgabetyp eine Datenklasse und zum anderen die DataAccessObject-Klasse (DataAccessImpl), welche die Datenbankzugriffe durchführt.

5.8.1. Datenklassen

Es wurde für jeden Ausgabetypen eine Datenklasse erstellt. Diese enthalten jeweils die für die Darstellung benötigten Attribute, sowie Methoden zum Zugriff darauf. Im Folgenden werden die Klassen und deren zugehörigen Attribute genannt:

SummaryEntity Für die Zusammenfassung sind folgende Attribute als Zeichenketten enthalten:
Usecase, Name, Firma, Datum, Beschreibung, Jahr, Fahrzeugtyp, Höhenklasse, Emissionstyp, Sitzklasse, Ersetzungsfaktor, Departure- und Arrivalinformationen.

GriddedInventoryEntity Die Datenklasse für die Darstellung der Weltkarte beinhaltet mehrere Attribute als Zeichenketten. Diese geben einige Metadaten wieder, namentlich Jahr, Emissionstyp, Sitzklasse, Höhenklasse, Departure- und Arrivalinformationen. Für die Hintergrundkarte werden drei Objekte vom Typ File benötigt. Die Emissionswerte werden in einer Hash Map übergeben. Diese enthält als Key eine Kodierung der Koordinaten und als Value den Emissionswert als Fließkommazahl.

DetailedInformationEntity Um eine tabellarische Ansicht der Emissionswerte zu ermöglichen, benötigt die zugehörige Datenklasse zwei Attribute. Eines davon ist ein Vektor, der die existierenden Regionen beinhaltet. Diese befüllen eine Spalte der Tabelle. Das andere Attribut ist ein zweidimensionales Array, das die restlichen Spalten der Tabelle mit den Emissionswerten füllt.

5.8.2. Umsetzung der DataAccessObject-Klasse

Der Zugriff auf die Datenbank wurde durch eine DataAccessObject-Klasse umgesetzt. Dieser Abschnitt beschreibt die wesentlichen Methoden, sowie den Konstruktor der Klasse.

Im Konstruktor des DataAccess-Objekts werden drei Hash Maps initialisiert und gefüllt. Diese Hash Maps beinhalten als Key die Werte, die auf der Benutzeroberfläche als Sitzklasse, Emission und Höhenklasse ausgewählt werden können. Als Value ist die zugehörige Zeichenkette, die für die Umsetzung in eine SQL-Anweisung benötigt wird, eingetragen. Folgender Quellcode-Ausschnitt veranschaulicht das:

```
seatClasses = new HashMap<String, String>();
seatClasses.put(" 200 <= x < 250 ", "200 AND 250 ");
emissionClasses = new HashMap<String, String>();
emissionClasses.put("CO2", "dCOInKginCell");
heightClasses = new HashMap<String, String>();
heightClasses.put("3000 m - 4000 m", "4");
```

Listing 14: Initialisieren und Füllen der Hash Maps im DAO-Konstruktor

Des Weiteren stellt die DAO-Klasse folgende Methoden bereit:

- establishDBConnection: Damit nicht für jeden Datenbankzugriff eine neue Verbindung zur Datenbank hergestellt werden muss, gibt es eine Methode, die die Verbindung herstellt. Über deren Attribut Statement lassen sich die SQL-Abfragen ausführen. Dabei wird zunächst der Treiber geladen, dann eine Verbindung hergestellt und eine Instanz der Klasse Statement aus dem Package java.sql erstellt. Der zugehörige Quellcode sieht wiefolgt aus:

```
public void establishDBConnection() {
        Class.forName(driver);
        connection = DriverManager.getConnection(url, username,
            password);
        statement = connection.createStatement();
 }
```

Listing 15: Quellcode-Ausschnitt der Verbindungsherstellung mit der Datenbank

Die erforderliche Ausnahmenbehandlung wurde aus Platzgründen nicht aufgeführt. Dabei ist zu beachten, dass driver, url, username und password Variablen vom Typ String sind. Diese sind für die Standardverbindung im Rahmen der Entwicklung dieses Prototyps hart kodiert. Es besteht die Möglichkeit in der Benutzeroberfläche eine manuell definierte Verbindung zu wählen, wodurch diese Variablen überschrieben werden.

- getGenericEmissionPerCell: Diese Methode führt eine SQL-Abfrage durch und gibt eine Hash Map mit Zellenkodierung als Key und Emissionswert als Value zurück. Als Übergabeparameter nimmt die Methode fünf Zeichenketten für Höhenklasse, Emissionstyp, Sitzklasse, Start- und Zieleinstellungen entgegen. Aus diesen Angaben wird eine SQL-Abfrage generiert.
  Das Grundgerüst der Abfragen ist immer gleich, es ändern sich lediglich die JOIN- und WHERE-Klauseln. Folgendes Grundgerüst liegt den Abfragen der Emissionen zugrunde:

```
String query = "SELECT e.CL, e.CP, sum(e." + emissionTable + ")
FROM " + fateOutputTable + " e " + joins + wheres + " GROUP BY
    e.CL, e.CP";
```

Listing 16: Grundgerüst der Abfrage von Emissionen

Die Variable fateOutputTable legt den Namen der Tabelle mit den von 4D-FATE berechneten Rohdaten fest.
Durch die Variable emissionTable wird festgelegt, auf welchen Schadstoff abgefragt wird. Dazu wird aus der Hash Map der Emissionsklassen der entsprechende Spaltenname zu dem in der Oberfläche gewählten Schadstoff ausgelesen.
Die Variablen joins und wheres sind vom Typ String und enhalten die JOIN- bzw. WHERE-Klauseln. Deren Erstellung ist im Folgenden beschrieben: Ist eine spezielle Sitzeklasse ausgewählt, muss die Flugplantabelle mit der Rohdatentabelle über einen Join verknüpft werden, da sie die Anzahl der Sitze enthält. Hat der Benutzer bei den Departure- und Arrivaleinstellungen etwas anderes als *all* gewählt, müssen auch hier Tabellen verknüpft werden. Die Flugplantabelle wird über einen Join mit der Start- bzw. Zielflughafentabelle verknüpft.

Am Beispiel der Sitzklasse ist an dieser Stelle aufgeführt, wie die JOIN-Klausel zusammengesetzt wird:

```
if ((!seats.equals("- ALL -")) && (!seats.equals("-----"))) {
            joins += "INNER JOIN flightplan f ON e.OAGFlightId =
                f.ID ";
        }
```

Listing 17: Ausschnitt aus dem Zusammensetzen der JOIN-Klausel

In ähnlicher Weise wird die WHERE-Klausel zusammengesetzt. Diese wird mit der Zeichenkette *WHERE* initialisiert. Ist ein Parameter verschieden von *all* ausgewählt, wird zunächst überprüft, ob die Klausel schon eine Bedingung enthält. In diesem Fall wird die Klausel um die Zeichenkette *AND* erweitert.
Außerdem wird der für die Abfrage benötigte Wert aus der Hash Map ausgelesen. Diesem Wert wurde das in der Oberfläche ausgewählte Element als Key zugewiesen. Am Beispiel der Sitzklasse lässt sich dies verdeutlichen:

```
String seatBorders = seatClasses.get(seats);
if (!wheres.equals("WHERE ")) {
                wheres += "AND ";
  }
  wheres += "f.Seats " + seatBorders;
```

Listing 18: Ausschnitt aus dem Zusammensetzen der WHERE-Klausel

In ähnlicher Weise wird die WHERE-Klausel gegebenenfalls durch weitere Abfrageparameter erweitert, was jedoch an dieser Stelle nicht weiter beleuchtet wird. Wurde die WHERE-Klausel um keinen Parameter erweitert, wird sie durch einen leeren String ersetzt, sodass die SQL-Abfrage syntaktisch korrekt bleibt. Als Ergebnis liefert die SQL-Abfrage eine Instanz vom Typ ResultSet[15]. Für die Weiterverarbeitung ist es jedoch wünschenswert, auf allgemeinere Datentypen zugreifen zu können. Dazu wird das Result Set ausgewertet und die Ergebnisse in eine Hash Map übertragen.

[15]Das Interface ResultSet ist im Package java.sql enthalten und repräsentiert die Ergebnismenge einer Datenbankabfrage.

Die Hash Map beinhaltet als Key eine Kodierung der Koordinaten und als Value den Emissionswert. Die Kodierung der Koordinaten besteht aus dem Breitengrad als String, einem Semikolon als Trennzeichen, sowie dem Längengrad als String. Somit kann über die Methode split() der Klasse String später darauf zugegriffen werden.

Der zugehörige Quellcode veranschaulicht das:

```
HashMap<String, Double> emissions = new HashMap<String,Double>();
ResultSet rs = statement.executeQuery(query);
while (rs.next()) {
                String breite = rs.getString("CP");
                String laenge = rs.getString("CL");
                double emissionD = rs.getDouble("sum(e." +
                    emissionTable + ")");
                emissions.put(breite + ";" + laenge, emissionD);
}
return emissions;
```

Listing 19: Verarbeitung der Ergebnismenge einer SQL-Abfrage

Auf die Darstellung der Ausnahmenbehandlung wurde aus Gründen der Übersicht verzichtet. Die Variable query ist eine Zeichenkette und beinhaltet die Abfrage, die - wie oben beschrieben - zusammengesetzt wurde.

- getAllDepCountries: Um die Auswahlbox der Startländer zu füllen, werden alle vorkommenden Länder aus der Tabelle der Startflughäfen abfragt. Die zugehörige SQL-Abfrage lautet:

```
SELECT DISTINCT DepIATACtryName
FROM depairportinfo d
ORDER BY DepIATACtryName ASC;
```

Listing 20: SQL-Anweisung zur Abfrage aller Startländer

Über die Ergebnismenge wird iteriert und als Ergebnis ein Objekt vom Typ Vector zurückgegeben.

- getAllDepRegions, getAllArrCountries, getAllArrRegions: In gleicher Weise wie

die Startländer werden die Startregionen, sowie Zielländer und -regionen ermittelt.

- getWorldDBFFile: Die Darstellung der Umrisse der Kontinente wird durch ein Shape-File umgesetzt. Um unabhängig vom jeweiligen Rechner Zugriff auf das Shape-File zu haben, wurde es in binär als Binary Large Object (BLOB) in der Datenbank gespeichert.
  Durch die Methode getWorldDBFFile() wird eine der drei Dateien, die zum Shape-File gehören, aus der Datenbank ausgelesen. Dazu wird eine Instanz vom Typ File erzeugt. Diese benötigt als Übergabeparameter einen Pfad. Als Dateipfad wurde *C:/temp/world.dbf* gewählt, welcher auf Windows-Systemen im Rahmen der Entwicklung eines Prototyps als verfügbar angenommen wird. Das Auslesen aus der Datenbank erfolgt über eine Instanz der Klasse FileOutputStream, die Binärdaten in das File-Objekt schreibt.
  Im folgenden Quellcode-Ausschnitt wurde die Ausnahmenbehandlung aus Gründen der Übersichtlichkeit weggelassen:

```
File file = new File("C://temp//world.dbf");
ResultSet rs = statement.executeQuery("SELECT File FROM
   shapefile_blobs WHERE ID = 1");
FileOutputStream fileOutputstream = new FileOutputStream(file);
    while(rs.next()){
       byte[] fileBytes = rs.getBytes(1);
       fileOutputstream.write(fileBytes);
    }
```

Listing 21: Auslesen einer Binärdatei aus der Datenbank

- getWorldSHPFile, getWorldSHXFile: Das Shape-File besteht aus drei Dateien. Ebenso wie in der vorherigen Methode werden die anderen Dateien, die das Shape-File bilden, über eine Instanz der Klasse FileOutputStream aus der Datenbank gelesen. Anschließend werden sie einer Instanz der Klasse File übergeben.
- getPasswordToUser: Für den Login-Vorgang ist es erforderlich, zu dem eingegebenen Benutzernamen den Hashwert des Passwortes zu ermitteln. Dazu wird

der Benutzername als Zeichenkette übergeben. Dieser Benutzername wird in die SQL-Abfrage integriert und durchgeführt. Der Abfragestring ist wie folgt umgesetzt:

```
"SELECT Password FROM `user` u WHERE Username = '"+ user +"'"
```

Listing 22: Abfragestring zum Auslesen eines Passworthashes aus der Datenbank

Aus der Ergebnismenge wird das Passwort als String extrahiert und zurückgegeben.

- getDetailedEmissions: Diese Methode setzt eine SQL-Abfrage ab und wandelt das Ergebnis in ein zweidimensionale Array vom Typ double[ ][ ] um, welches zurückgegeben wird. Dieses Array hat 17 Zeilen mit jeweils 4 Spalten. Somit lässt sich von jeder der 17 Regionen des OAG-Flugplans die Gesamtsumme pro Emissionstyp ausgeben. Folgende SQL-Abfrage führt zur benötigten Ergebnismenge:

```
SELECT SUM(dFInKgInCell),SUM(dNoxInKgInCell),
SUM(dCOInKginCell),SUM(dHCInKgInCell)
FROM fate_output_data e
INNER JOIN flightplan f ON e.OAGFlightId = f.ID
INNER JOIN depairportinfo d ON f.DepAirport = d.DepAirportID
GROUP BY d.DepRegName;
```

Listing 23: SQL-Anweisung zur Summierung der Emissionen

Über die erzeugte Ergebnismenge wird iteriert, die Emissionen werden ausgelesen und in das Array eingefügt.

5.9. Testen der Anwendung

Aufgrund der großen Datenmenge und der vielen Kombinationsmöglichkeiten der Abfragen, ist es nicht möglich, in einem angemessenen Zeitrahmen alle auftretenden Abfragekombinationen zu testen.
Um dennoch in einem gewissen Maß die Funktionalität nachzuweisen, wurde für jede Kombination von Abfrageparametern eine beispielhafte Abfrage durchgeführt.
Die dabei erzeugte SQL-Abfrage wurde mit einer Abfrage verglichen, wie man sie

händisch ausgeführt hätte. Außerdem wurde das von der Anwendung erzeugte Bild der Weltkarte einer Plausibilitätskontrolle unterzogen. Beim Durchführen dieser Überprüfungen sind keine Unregelmäßigkeiten aufgefallen.
Durch regelmäßig durchgeführte Regressionstests bei Änderungen an der Anwendung wurde sichergestellt, dass keine neuen Fehler aufgetreten sind. Außerdem wurde dadurch ausgeschlossen, dass die durchgeführten Änderungen Auswirkungen auf andere Teile der Anwendung haben.
Zusätzlich wird die Funktionalität im Rahmen des Abnahmetests verifiziert.

6. Zusammenfassung und Ausblick

Das Ziel der Arbeit, eine Anwendung zum Abfragen, Visualisieren und Reporting von Emissionsstudiendaten zu entwickeln, wurde erreicht. Es wurde eine Anwendung entwickelt, die dem Benutzer eine dynamische Generierung von Abfragen erlaubt. Die dadurch erzeugten Ergebnisse sind sowohl auf einer Weltkarte, als auch in Tabellenform darstellbar. Das Exportieren der Ergebnisse als Bilddatei im Format *JPEG*, sowie in einem standardisierten Format als Microsoft Excel-Dokument ist umgesetzt. Eine Erweiterung um zusätzliche Export-Formate, wie PDF oder XML ist möglich.
Die Datengrundlage bildet eine MySQL-Datenbank. Diese enthält Emissionsdaten, die durch das Tool 4D-FATE berechnet wurden. Außerdem beinhaltet die Datenbank Flugplandaten und Informationen zu Flughäfen, welche aus den OAG-Flugplandaten entnommen sind. Eine Validierung der Daten wurde sowohl durch eine Plausibilitätskontrolle der erzeugten Datenmengen, als auch durch stichprobenartige Überprüfung ausgewählter Verbindungen erreicht. Dabei wurde erkannt, dass zunächst auf einer nicht vollständigen Datengrundlage gearbeitet wurde. Der Grund für diesen Missstand wurde identifiziert und das Problem inzwischen behoben.
Es besteht die Möglichkeit Abfragen auf externe Datenbanken, welche die gleiche Struktur aufweisen, durchzuführen. Somit besteht die grundsätzliche Möglichkeit, die Anwendung an andere Firmen weiterzugeben, ohne die vertraulichen Rohdaten weiterzugeben. Außerdem ist es dadurch möglich, fremde Datenquellen nach einer Anpassung an die Datenstruktur anhand der erzeugten Bilder und Tabellen zu evaluieren.
Eine passwortgeschützte, rollenbasierte Zugriffsbeschränkung verhindert den unerlaubten Zugriff auf die Anwendung. Es wurden die Rollen des (Standard-)Benutzers und des Administrators umgesetzt. Das Hinzufügen weiterer Rollen ermöglicht es, für jede Rolle eine differenzierte Auswahl an Abfragekriterien bereitzustellen. Denkbar sind ein weniger priviligierter Benutzer als Gastzugang, der nur allgemeine Daten sehen kann, sowie ein abteilungsinterner Benutzer mit erweitertem Zugriff.
Die Umsetzung als modulbasierte Anwendung bietet die Möglichkeit der flexiblen Erweiterung um weitere Module, die andere Darstellungsmöglichkeiten oder sonsti-

ge Funktionalitäten bereitstellen.

Um die Darstellung einer Weltkarte zu realisieren, wurde die in Java geschriebene Bibliothek OpenMap verwendet. Der modulbasierte Aufbau bietet die Möglichkeit diese Darstellung durch ein anderes, geeignetes Werkzeug umzusetzen. Beispielhaft seien hier Visualisierungen auf Basis von *ArcView* oder *Google-Earth* genannt.

Der Datenbestand beinhaltet die Emissionsdaten, sowie einen Flugplan und Informationen zu den Flughäfen. Aufgrund der großen Datenmengen sind Optimierungsmaßnahmen unumgänglich. Wenngleich das Anpassen der Storage-Engine, eine Indizierung der Fremdschlüssel, eine Anpassung der verwendeten Datentypen und das Konfigurieren von Caches eine Performancesteigerung bewirkte, gibt es weitere Möglichkeiten, die aus dem Bereich *Data Warehousing* stammen.

Eine Möglichkeit ist die Verwendung materialisierter Sichten. Materialisierte Sichten werden wie Views durch eine SQL-Anweisung definiert. Das Resultat dieser Anweisung wird gespeichert (und bei Bedarf aktualisiert), um es wiederzuverwenden. Eine weitere Voraussetzung ist der überwiegend lesende Zugriff auf eine stabile Datenbasis.[24] Da diese Voraussetzungen zutreffen, ist davon auszugehen, dass die Verwendung materialsierter Sichten die Performance steigern wird. Sie werden nicht direkt von der Anwendung, sondern vom Datenbanksystem verwendet und sind transparent, wenn sie von einen Datenbanksystem oder einem Applikationsserver verwaltet werden. Da MySQL das Prinzip materialisierter Sichten nicht unterstützt und kein Applikationsserver zum Einsatz kommt, müssten sie in der Applikationsebene umgesetzt werden. Somit wäre deren Verwendung nicht transparent.

Des Weiteren kann bei sinnvoller Einteilung durch Partitionierung der Datenbanktabellen die Leistung verbessert werden. Durch die Einteilung der logischen Einheiten auf kleinere physische Einheiten, kann der sequenzielle Zugriff effizienter durchgeführt werden. [25, Seite 146 f.] Denkbar ist das Erstellen einer Partition je Start- und je Zielregion. Bei Abfragen auf eine bestimmte Start- oder Zielregion kann das Datenvolumen eingeschränkt werden, das durchsucht werden muss. Dazu muss das Partitionierungskriterium (z.B. die Region) in die Rohdatentabelle aufgenommen werden. Um das Risiko einer Performance-Einbuße durch schlechte Wahl der Partitionierung zu minimieren,

ist eine Analyse des Zugriffsverhaltens notwendig.
Es wurde bereits ein Index auf die Fremdschlüsselspalten angelegt. Aufgrund der Komplexität des Themas, wurde die Anlegung von Indizes nicht weiter verfolgt. Es ist mit einem enormen Zeitaufwand verbunden, das volle Potential des Indexdesigns auszuschöpfen. Bei entsprechend angelegten Indizes können Abfragen durch *Index only*-Zugriff enorm beschleunigt werden, da nur noch die Inidizes, nicht aber die kompletten Daten durchsucht werden müssen.
Da ein Großteil der Informationen, die in der Datenbank gehalten werden, (noch) nicht abfragbar ist, bietet es sich an, sogenannte Mini-Dimensionen zu bilden. Diese enthalten nur die Attribute die häufig abgefragt werden (z.B. Sitzklasse, sowie Start- und Zielregion). Anschließend wird jedem Eintrag in der Rohdatentabelle eine Kombination dieser Attribute zugewiesen. Dies erfolgt über die ID der Mini-Dimension. Somit lassen sich die zu durchsuchenden Dimensionsdaten erheblich verkleinern.
Diese Maßnahmen sind insbesondere interessant, weil der Datenbestand im Rahmen des Projektes um die prognostizierten Werte für das Jahr 2020 erweitert wird.
Abschließend kann festgehalten werden, dass die Anforderungen an die Anwendung erfüllt wurden. Die geforderten Funktionalitäten sind prototypisch implementiert. Durch den gewählten modularen Aufbau ist es möglich, die Anwendung um weitere Funktionalitäten zu erweitern und anzupassen.

Literatur

[1] Schäfer, Martin: *Emailkommunikation*

[2] *Das DLR im Überblick*. http://www.dlr.de/desktopdefault.aspx/tabid-636, Abruf: 15. September 2010

[3] *Institut für Flughafenwesen und Luftverkehr*. http://www.dlr.de/fw/desktopdefault.aspx/tabid-2939, Abruf: 15. September 2010

[4] *The 'Clean Sky' JTI*. http://www.cleansky.eu/index.php?arbo_id=50&set_language=en, Abruf: 15. September 2010

[5] Henry Pak, Katrin Dahlmann Marc Gelhausen Wolfgang Grimme Erik Grunewald Michael Hepting Eike Hoffmann Alexandra Leipold Robert Sausen Martin Schaefer Tobias S. Peter Berster B. Peter Berster: *Integrierte Modellierung des Luftverkehrssystems 1 - Abschlussbericht*. Mai 2010

[6] Holger Pabst, Brigitte B.: *Das 4D-Berechnungstool FATE*, Juni 2003

[7] Balzert, Helmut: *Lehrbuch der Software-Technik - Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Heidelberg, Berlin : Spektrum Akademischer Verlag, 1998

[8] Balzert, Helmut: *Lehrbuch der Software-Technik - Software-Entwicklung*. Heidelberg, Berlin : Spektrum Akademischer Verlag, 2000 (2. Auflage)

[9] *Swing GUI Builder (Matisse) Features*. http://netbeans.org/features/java/swing.html, Abruf: 15. September 2010

[10] *OSGiAndNetBeans - NetBeans Wiki*. http://wiki.netbeans.org/OSGiAndNetBeans, Abruf: 15. September 2010

[11] *Video: NetBeans 6.9 Overview*. http://netbeans.org/kb/docs/ide/overview-screencast.html, Abruf: 15. September 2010

[12] Böck, Heiko: *NetBean Platform 6 - Rich-Client-Entwicklung mit Java*. Bonn : Galileo Press, 2008

[13] *Trail: The Extension Mechanism.* http://download.oracle.com/javase/tutorial/ext/index.html, Abruf: 15. September 2010

[14] *Java Product Versioning.* http://download-llnw.oracle.com/javase/1.5.0/docs/guide/versioning/spec/versioning2.html#wp89936, Abruf: 15. September 2010

[15] Erich Gamma, Ralph Johnson John V. Richard Helm H. Richard Helm: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 2000

[16] Deepak Alur, Dan M. John Crupi C. John Crupi: *Core J2EE Patterns - Best Practices and Design Strategies.* Sun Microsystems Press, 2003

[17] *Oracle Completes Acquisition of Sun.* http://www.oracle.com/us/corporate/press/044428, Abruf: 15. September 2010

[18] Marcus Throll, Oliver B.: *Einstieg in SQL.* Bonn : Galileo Press, 2010

[19] *What is OpenMap(tm)?* http://openmap.bbn.com/whatis.html, Abruf: 15. September 2010

[20] *MapBean (OpenMap API).* http://openmap.bbn.com/doc/api/com/bbn/openmap/MapBean.html, Abruf: 15. September 2010

[21] *BasicMapPanel (OpenMap API).* http://openmap.bbn.com/doc/api/com/bbn/openmap/gui/BasicMapPanel.html, Abruf: 15. September 2010

[22] D. Eastlake, P. J.: *US Secure Hash Algorithm 1 (SHA1).* Network Working Group, 2001. – RFC 3174

[23] *Excel-Tools unter Java im Vergleich.* http://www.ordix.de/ORDIXNews/1_2008/Java_J2EE_JEE/jexcelapi_jakarta.html, Abruf: 15. September 2010

[24] *Materialisierte Sichten.* http://wwwiti.cs.uni-magdeburg.de/iti_db/lehre/dw/dw0001/dw07.pdf, Abruf: 15. September 2010

[25] Klaus Küspert, Jan N.: *Informatik Spektrum.* Springer Verlag, 1999

A. 4D-FATE Input und Output

A.1. 4D-FATE Flugverbindungsdatei (Auszug)

```
97/09/18-97/09/18 1 1 1
BKK--LHR- 744 000 0 10 0 45 1 100.7489 13.68583 1.524 -0.45 51.46667 0.0
BUF--FLL- 320 001 0 10 1065 835 1 -78.73333 42.93333 220.67520002 -80.16667 26.06667 0.0
BUF--JFK- 320 002 0 10 850 775 1 -78.73333 42.93333 220.67520002 -73.78333 40.63333 3.9624
DAL--HOU- 733 003 0 10 775 720 1 -96.85 32.85278 148.4376 -95.27833 29.645 14.0208001
DCA--SEA- 733 004 0 10 878 960 1 -77.03333 38.85278 4.572 -122.3 47.45 0.0
GOJ--DME- TU3 005 0 10 245 245 1 43.78333 56.23333 78.0288 37.88333 55.4 0.0
GVA--LUG- AT7 006 0 10 445 395 1 6.11667 46.23333 430.07280003 8.91667 46.0 278.892
HKG--SIN- 744 007 0 10 220 60 1 113.9025 22.30139 0.0 103.9833 1.36667 0.0
HLP--SIN- 722 008 0 9 1230 1050 1 106.8833 -6.26667 25.6032 103.9833 1.36667 0.0
IAH--HNL- 763 009 0 10 405 1235 1 -95.33333 29.98333 0.0 -157.9333 21.33333 0.0
IST--JFK- 343 010 0 10 755 920 1 28.81667 40.98333 49.6824 -73.78333 40.63333 3.9624
JFK--FLL- 320 011 0 10 1065 950 1 -73.78333 40.63333 3.9624 -80.16667 26.06667 0.0
JFK--LAX- 763 012 0 10 953 1020 1 -73.78333 40.63333 3.9624 -118.4 33.93333 0.0
JNB--CPT- 733 013 0 10 505 385 1 28.25 -26.13333 1694.0784 18.6 -33.96667 46.0248
KUL--SYD- 772 014 0 10 485 1155 1 101.685 2.735 0.0 151.1667 -33.93333 0.0
LAX--HNL- 763 015 0 10 1145 1175 1 -118.4 33.93333 0.0 -157.9333 21.33333 0.0
MJI--BEN- 733 016 0 10 610 480 1 13.26667 32.88333 0.0 20.26667 32.1 131.9784
MKE--LGA- 717 017 0 10 939 760 1 -87.9 42.95 220.37040002 -73.86667 40.76667 0.0
NRT--HKG- 744 018 0 10 780 630 1 140.3833 35.76667 42.9768004 113.9025 22.30139 0.0
OMA--LGA- 717 019 0 10 939 650 1 -95.9 41.3 299.9232 -73.86667 40.76667 0.0
OMA--MKE- 717 020 0 10 725 650 1 -95.9 41.3 299.9232 -87.9 42.95 220.37040002
ORD--HNL- 772 021 0 10 1141 1270 1 -87.9 41.98333 203.6064 -157.9333 21.33333 0.0
PHX--ORD- 320 022 0 10 479 750 1 -112.01 33.43667 345.948004 -87.9 41.98333 203.6064
PIT--LAX- 320 023 0 10 875 1000 1 -80.23333 40.49167 366.9792 -118.4 33.93333 0.0
SIN--HLP- 722 024 0 9 1290 1260 1 103.9833 1.36667 0.0 106.8833 -6.26667 25.6032
SYD--BKK- 744 025 0 10 795 600 1 151.1667 -33.93333 0.0 100.7489 13.68583 1.524
BEN--MJI- 722 026 0 10 435 420 1 20.26667 32.1 131.9784 13.26667 32.88333 0.0
CPT--JNB- 733 027 0 10 980 860 1 18.6 -33.96667 46.0248 28.25 -26.13333 1694.0784
DAL--LIT- 733 028 0 10 840 835 1 -96.85 32.85278 148.4376 -92.23333 34.73333 0.0
DME--GOJ- TU3 029 0 10 460 335 1 37.88333 55.4 0.0 43.78333 56.23333 78.0288
FLL--PHL- 752 030 0 10 1282 1120 1 -80.16667 26.06667 0.0 -75.23333 39.88333 0.0
GUM--HNL- 763 031 0 10 425 1020 1 144.7333 13.48333 0.0 -157.9333 21.33333 0.0
GUM--IAH- 763 032 0 9 1265 720 1 144.7333 13.48333 0.0 -95.33333 29.98333 0.0
HKG--LAX- 744 033 0 10 105 990 1 113.9025 22.30139 0.0 -118.4 33.93333 0.0
HKG--NRT- 744 034 0 10 355 1410 1 113.9025 22.30139 0.0 140.3833 35.76667 42.9768004
HKG--SFO- 744 035 0 10 690 355 1 113.9025 22.30139 0.0 -122.3833 37.61667 0.0
HNL--LAX- 763 036 0 11 485 530 1 -157.9333 21.33333 0.0 -118.4 33.93333 0.0
HOU--DAL- 733 037 0 10 745 690 1 -95.27833 29.645 14.0208001 -96.85 32.85278 148.4376
HOU--LIT- 733 038 0 10 840 750 1 -95.27833 29.645 14.0208001 -92.23333 34.73333 0.0
IST--DLM- CR1 039 0 9 1260 530 1 28.81667 40.98333 49.6824 28.78333 36.7 0.0
JFK--BUF- 320 040 0 10 740 660 1 -73.78333 40.63333 3.9624 -78.73333 42.93333 220.6752002
KUL--VIE- 772 041 0 9 1390 1415 1 101.685 2.735 0.0 16.55 48.11667 0.0
LAX--JFK- 763 042 0 11 119 810 1 -118.4 33.93333 0.0 -73.78333 40.63333 3.9624
LGA--MKE- 717 043 0 10 780 705 1 -73.86667 40.76667 0.0 -87.9 42.95 220.37040002
LPX--RIX- F50 044 0 10 340 180 1 21.1 56.51667 0.0 23.96667 56.91667 10.3632
LUG--GVA- AT7 045 0 10 360 315 1 8.91667 46.0 278.892 6.11667 46.23333 430.0728003
```

A.2. 4D-FATE Emissionsprofildatei

```
H[m]   dS[m] dT[sec] dF[kg]   dNOx[kg]   dSOOT[kg] dPART[-]
10058.5 11555463.45 12001187.81
  239841.45 1096.69
0.0 1.00   660.00   681.12   4.08   0.0076   3.64E+16
0.0 1.00   108.00   1339.26 66.95 0.0562   2.71E+17
304.8 3007.82 23.29 264.53   12.08 0.0103   5.13E+16
609.6 3118.56 23.69 265.81   12.32 0.0099   5.08E+16
914.4 2926.10 21.93 244.42   11.57 0.0087   4.62E+16
1219.2   3036.70 22.42 246.54   11.78 0.0084   4.57E+16
1524.0   3154.88 22.96 248.65   11.96 0.0081   4.53E+16
3048.0   17873.84   124.26   1288.50 63.13 0.0364   2.24E+17
4572.1   23661.92   113.03   1261.75 74.31 0.0369   2.69E+17
6096.1   31315.21   139.06   1430.24 83.85 0.0331   2.87E+17
7620.1   44650.90   184.15   1708.47 96.44 0.0310   3.22E+17
9144.1   56513.67   221.38   1786.13 92.08 0.0248   3.12E+17
10058.5 50581.87   200.51   1376.56 63.17 0.0136   2.01E+17
10058.5 11555463.45 45976.34   170043.70 3407.52 1.1541   1.77E+19
9144.1   14453.66   57.12 41.79 0.19   0.0003   3.98E+15
7620.1   28730.27   113.25   99.91 0.53   0.0007   8.96E+15
6096.1   31627.72   132.72   135.02   0.79   0.0011   1.11E+16
4572.1   30696.77   138.53   161.24   1.06   0.0015   1.24E+16
3048.0   29991.77   145.40   192.56   1.40   0.0020   1.41E+16
1524.0   33781.51   244.52   427.34   3.61   0.0043   2.55E+16
1219.2   6899.94 57.63 123.52   1.13   0.0013   7.14E+15
914.4 7006.40 59.37 130.45   1.21   0.0014   7.68E+15
609.6 7110.92 61.97 138.54   1.29   0.0016   8.14E+15
304.8 7568.81 82.05 307.67   4.14   0.0045   2.26E+16
0.0 8015.13 90.74 344.63   4.61   0.0052   2.52E+16
0.0 1.00   300.00   309.60   1.86   0.0034   1.66E+16
```

A.3. 4D-FATE Ergebnisdatei (Auszug)

```
101 77 0 1 10 7 10 0.002 12.80 1827.770 43.340 0.056 268000000000000000.000 744-81
101 77 1 1 10 7 10 0.710 1.20 608.322 15.652 0.022 106055380577428880.000 744-81
101 77 2 1 10 7 10 8.774 1.10 540.942 14.660 0.019 106261889774719790.000 744-81
101 77 3 1 10 7 10 11.126 1.14 552.291 15.409 0.018 114123771367411620.000 744-81
101 77 4 1 10 7 10 12.551 1.16 565.482 16.214 0.018 122628681127871340.000 744-81
101 77 5 1 10 7 10 11.050 0.85 408.327 11.884 0.011 89640880853904080.000 744-81
101 76 5 1 10 8 10 5.934 0.46 219.283 6.382 0.006 48139628218342576.000 744-81
101 76 6 1 10 8 10 20.062 1.49 678.351 19.226 0.017 140812448781497120.000 744-81
101 76 7 1 10 8 10 22.978 1.65 726.438 20.136 0.015 143689216074253090.000 744-81
101 76 8 1 10 8 10 33.817 2.26 891.099 22.962 0.015 163178775806058850.000 744-81
101 76 9 1 10 8 10 4.556 0.30 119.891 3.087 0.002 21939424560097012.000 744-81
100 76 9 1 10 8 10 29.233 1.93 712.341 17.468 0.010 118523014816573540.000 744-81
100 76 10 1 10 8 10 24.395 1.60 553.307 12.884 0.006 82816084670421744.000 744-81
100 75 10 1 10 8 10 9.233 0.60 209.422 4.877 0.002 31345249362569568.000 744-81
100 75 11 1 10 8 10 111.073 7.42 1481.589 22.702 0.011 175822509110142400.000 744-81
99 75 11 1 10 8 10 21.072 1.41 229.718 2.712 0.002 26287766433443564.000 744-81
99 74 11 1 10 8 10 141.820 9.49 1546.041 18.250 0.011 176921314156636220.000 744-81
99 73 11 1 10 8 10 8.950 0.60 97.566 1.152 0.001 11164993910462470.000 744-81
98 73 11 1 10 8 10 133.349 8.92 1453.699 17.160 0.010 166354227745896580.000 744-81
98 72 11 1 10 8 10 36.295 2.43 395.667 4.671 0.003 45278169498642952.000 744-81
97 72 11 2 10 8 10 106.524 7.13 1161.273 13.708 0.008 132890329294347820.000 744-81
97 71 11 2 10 8 10 60.829 4.07 663.126 7.828 0.005 75884856453060048.000 744-81
96 71 11 2 10 8 10 82.555 5.52 899.967 10.624 0.006 102987766769808140.000 744-81
96 70 11 2 10 8 10 82.428 5.52 898.589 10.608 0.006 102830093179973380.000 744-81
95 70 11 2 10 8 10 61.567 4.12 671.170 7.923 0.005 76805411979388128.000 744-81
95 69 11 2 10 8 10 100.977 6.76 1100.800 12.995 0.008 125970122524504740.000 744-81
94 69 11 2 10 8 10 43.680 2.92 476.178 5.621 0.003 54491424523184480.000 744-81
94 68 11 2 10 8 10 116.369 7.79 1268.592 14.975 0.009 145171456607694750.000 744-81
93 68 11 2 10 8 10 29.005 1.94 316.199 3.733 0.002 36184257599995832.000 744-81
93 67 11 2 10 8 10 128.504 8.60 1400.879 16.537 0.010 160309754088742020.000 744-81
92 67 11 2 10 8 10 17.646 1.18 192.370 2.271 0.001 22013864722439800.000 744-81
92 66 11 2 10 9 10 137.288 9.19 1496.643 17.667 0.011 171268517365205250.000 744-81
91 66 11 3 10 9 10 9.702 0.65 105.764 1.249 0.001 12103102817673788.000 744-81
91 65 11 3 10 9 10 142.634 9.54 1554.922 18.355 0.011 177937613728802400.000 744-81
90 65 11 3 10 9 10 5.266 0.35 57.407 0.678 0.000 6569331492041893.000 744-81
90 64 11 3 10 9 10 144.457 9.67 1574.793 18.590 0.011 180211619726591710.000 744-81
89 64 11 3 10 9 10 4.430 0.30 48.291 0.570 0.000 5526214053991371.000 744-81
89 63 11 3 10 9 10 142.675 9.55 1555.362 18.360 0.011 177987981754191070.000 744-81
88 63 11 3 10 9 10 7.283 0.49 79.396 0.937 0.001 9085733526518464.000 744-81
88 62 11 3 10 9 10 137.205 9.18 1495.739 17.657 0.011 171164978011876900.000 744-81
87 62 11 3 10 9 10 13.916 0.93 151.706 1.791 0.001 17360446638445646.000 744-81
87 61 11 3 10 9 10 127.966 8.56 1395.022 16.468 0.010 159639457010378590.000 744-81
86 61 11 3 10 9 10 24.421 1.63 266.230 3.143 0.002 30466011289920288.000 744-81
86 60 11 3 10 9 10 114.872 7.69 1252.276 14.783 0.009 143304315233013760.000 744-81
85 60 11 4 10 9 10 38.897 2.60 424.031 5.006 0.003 48524039034356464.000 744-81
85 59 11 4 10 9 10 97.831 6.55 1066.506 12.590 0.008 122045660507302210.000 744-81
84 59 11 4 10 9 10 57.447 3.84 626.253 7.393 0.004 71665344986843560.000 744-81
```

B. Inhalt der beiliegenden CD-ROM

Der Arbeit ist eine CD-ROM beigelegt. Im Folgenden wird der Inhalt der Ordner erklärt. Zum Betrachten der PDF-Datei wird der frei verfügbare Adobe Reader oder eine vergleichbare Anwendung benötigt.

- **Abbildungen**:
  Dieser Ordner beinhaltet die in der Arbeit enthaltenen Abbildungen im JPEG-Format.
- **NetBeans Projekt**:
  Dieser Ordner enthält das NetBeans-Projekt und eine ausführbare Version des im Rahmen der Arbeit entwickelten Prototyps.
- **Bachelorarbeit**:
  Dieser Ordner beinhaltet die Bachelorarbeit im PDF-Format.
- **Literatur**:
  Dieser Ordner enthält die Online-Literatur, auf die in der Arbeit verwiesen wird, als offline verfügbare HTML-Seiten.