Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Oliver Steenbuck

Algorithmisches Routen- und Verzögerungsmanagement für das Logistikframework eines Luftfrachtanbieters

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Oliver Steenbuck

Algorithmisches Routen- und Verzögerungsmanagement für das Logistikframework eines Luftfrachtanbieters

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Thomas Thiel-Clemen

Abgegeben am 22. August 2011

Oliver Steenbuck

Algorithmisches Routen- und Verzögerungsmanagement für das Logistikframework eines Luftfrachtanbieters

Framework, Frameworkdesign, Luftfracht, API, Simulation

Kurzzusammenfassung

Fluggesellschaften stehen heute vor der Herausforderung, bei steigendem Fracht- und Passagieraufkommen sowie zunehmend überfüllten Lufträumen die Effizienz ihrer Routennetzwerke zu halten oder zu steigern. Netzwerk- und Verzögerungsmanagement als Kernbereiche dieser Aufgabe sind zum heutigen Zeitpunkt häufig noch manuelle Prozesse. In dieser Arbeit soll ein Framework zur Evaluation und Entwicklung algorithmischer Lösungen für das Netzwerk- und Verzögerungsmanagement entworfen und prototypisch implementiert werden.

Oliver Steenbuck

Algorithmic routemanagement and delayhandling in the context of an aircargo providers logistic framework

Framework, Frameworkdesign, Aircargo, API, Simulation

Abstract

Today, airlines face the challenge of keeping or even increasing the efficiency of their routenetworks amidst growing airspace congestion. Routemanagement and delayhandling as core competencies of this problem are in many parts still manual processes as of today. This paper will show the design and a prototypical implementation for a framework that can be used for testing and evaluating algorithmic solutions to the problems of routemanagement and delayhandling.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Teil I.

Einführung

1. Einleitung

1.1. Motivation

Das steigende Volumen und die Fokussierung auf Just-in-time-Lieferungen führen in der Logistikbranche zu immer höheren Anforderungen an Durchsatz und Termintreue der Transportnetzwerke (Grandjot u. a., 2007).

Während auf Grund der globalen Rezession 2008/2009 die Wachstumprognosen von ca. 6 % (Grandjot u. a., 2007) nicht erreicht wurden, zeigen aktuelle Statistiken (Cloodt, 2011) wieder ein Wachstum im Segment der Luftfrachtdienstleister. Die oben genannten Anforderungen an Termintreue und Durchsatz treten diese Anforderungen durch die spezifischen Fracht- und Infrastrukturbedingungen noch deutlicher hervor.

Während zwar das absolute Volumen der Luftfracht, gemessen am gesamten Transportaufkommen der Logistikbranche, durch die im Vergleich zu anderen Transportarten (Schiff, Schiene, Straße) hohen Kosten pro Einheit (Tonne, TEU) eher gering ist, werden im Umkehrschluss zumeist terminkritische Waren und/oder solche mit einem hohen Wert pro Gewichtseinheit per Luftfracht transportiert. Bei diesen Gütern legt der Kunde besonders hohen Wert auf die Einhaltung von Lieferterminen. So wird nur ca. 1 % des gesamten weltweiten Warenverkehrs über Luftfracht abgewickelt, der Anteil am kompletten Warenwert liegt jedoch nahe bei 35 % (Potzner, 2008).

Anbieter, die dem Kundenwunsch nach Termintreue entsprechen wollen, werden mit der Herausforderung inkongruenter Planungsziele konfrontiert: Zum einen gilt es für sie, den Durchsatz zu maximieren, was möglichst geringe Stillstände des fliegenden Materials erfordert, und zum anderen müssen sie Termintreue gewährleisten, was operativ schwieriger wird, wenn in den Stops der Transportmittel geringe oder keine Puffer eingeplant sind.

Als zusätzlich die Komplexität erhöhender Faktor im operativen Ablauf stellen sich die in der Praxis stark synchronisierten Prozesse zwischen Flughäfen, Fluglinien und Streckenkontrolle (Air Route Traffic Control Center) dar, die zu nicht offensichtlichen Abhängigkeiten führen (Grandjot u. a., 2007), (Ball u. a., 2006). So können zum Beispiel Probleme in der Abfertigung in London Heathrow (LHR) dazu führen, dass eine Maschine in Frankfurt keine Starterlaubnis erhält und Kapazitäten der Flugabfertigung bindet, die eine bereits in der Luft befindliche

Maschine aus Singapur benötigt. Diese wird als Reaktion ihre Geschwindigkeit reduzieren, um ihren Treibstoffverbrauch zu minimieren und ihre Ankunft in Frankfurt an die Verfügbarkeit von dortigen Kapazitäten anpassen (Ball u. a., 2006).

Diesen Ansprüchen begegnet die Branche im Moment primär durch manuelle Prozesse in Planung und operativem Betrieb. Netzwerkplanung und die Reaktion auf Störungen im operativen Betrieb werden hier primär durch Mitarbeiter sichergestellt. Die Unterstützung dieser Prozesse durch Computer beschränkt sich auf die Bereithaltung und Darstellung von Informationen. Planungsaufgaben sind teilweise noch nicht durch automatisierte Systeme abgebildet (Erdmann u. a., 2001).

1.2. Ziele

Die vorliegende Arbeit hat zum Ziel, ein Framework zu entwerfen und prototypisch zu implementieren, das die Entwicklung und Evaluierung von Algorithmen aus dem Teilbereich Transportlogistik (Luftfracht) vereinfacht.

Es ist ein Framework zu entwerfen, das durch eine standardisierte Definition der sich nicht ändernden fachlichen Domäne des Problems den Entwicklungs- und Evaluierungsaufwand für neue Algorithmen signifikant senkt. Bestandteil dieses Frameworks sind neben den fachlichen auch technische Komponenten, die die Entwicklung unterstützen. Im Groben wird das Framework aus einem Planungsteil und einem Simulationsteil bestehen, wobei der erste der Planung eines möglichst idealen Netzwerkes dient und der zweite der Simulation des Planes, unter Einbeziehung von Störungen und abschließender Generierung von Auswertungen.

Um diese Ziele zu erreichen, sind durch eine Analyse der fachlichen Domäne Komponenten zu isolieren, die über verschiedene Algorithmen hinweg konstant sind (Hot Spot Analyse[1]) und diese in einem Modell auszudrücken, das die wesentlichen Punkte der Fachlichkeit abbildet. Neben dieser Betrachtung der fachlichen Dimension des Problems ist auch eine Betrachtung der technischen Dimension durchzuführen, um technische Komponenten zu identifizieren, deren Aufnahme in das Framework die Entwicklung neuer Algorithmen unterstützen kann.

Aufgabe des Frameworkentwurfes ist es dann, basierend auf den Ergebnissen der Analyse die Schnittstellen der Komponenten so anzulegen, dass sie mit keinen oder minimalen Änderungen durch verschiedene Implementierungen des Hot Spots genutzt werden können.

Die prototypische Entwicklung des Frameworks dient der Absicherung und Vervollständigung der Analyseergebnisse durch empirische Methoden.

[1] siehe Grundlagen, Kapitel 2.2

1.3. Abgrenzung

Ziel der Arbeit ist nicht eine vollständige Implementierung der Transportlogistik eines Luftfrachtdienstleisters, da dieses Vorhaben durch die diversen Problemfelder in diesem Bereich zu komplex wäre. So ist beispielsweise die *Weight and Balance* Analyse ein Problem, das in seiner Komplexität den in dieser Arbeit behandelten nicht nachsteht. Ebenso ist die Zuweisung von Crews zu Flugzeugen aufgrund des regulatorischen Umfeldes eine nicht zu unterschätzende Aufgabe (vgl. Ball u. a., 2006, S. 2). Das Ziel dieser Arbeit liegt auch nicht darauf, einen Vergleich aller möglicher fachlicher Algorithmen zur Lösung der unter Ziele beschrieben Probleme zu bieten.

Aufgrund des relativ engen Zeitplanes soll das zu entwerfende Framework nicht vollständig implementiert werden, sondern nur die technische Machbarkeit durch eine prototypische Implementierung kritischer Pfade gezeigt werden.

1.4. Gliederung

Die Arbeit gliedert sich in sechs Kapitel. In der Einführung (Kapitel 2) werden erst die fachlichen Grundlagen der Logistik mit Fokus auf die Transportlogistik betrachtet, um anschließend auf die technischen Grundlagen des Frameworkdesigns einzugehen.

Aufbauend auf den fachlichen Grundlagen wird im ersten Kapitel des Hauptteils Anforderungsanalyse (Kapitel 3) die fachliche Problemstellung beleuchtet. In diesem werden zuerst Anwendungsszenarien beschrieben, um dann aus diesen Szenarien zu einer Beschreibung der an ihnen beteiligten Stakeholdern zu kommen. Abschließend wird an einem Fallbeispiel der konkrete Ablauf einer Planung gezeigt.

In Kapitel 4 (Spezifikation) werden die in der Analyse gewonnenen Erkenntnisse mit den Mitteln des Software Engineering formalisiert. Aus den sich aus dem Fallbeispiel ergebenden Anwendungsfällen werden Anforderungen, Prämissen und Ausgrenzungen generiert. Die Entitäten der Domäne werden durch ein fachliches Datenmodell abgebildet und es wird ein Komponentenschnitt vorgenommen. Aus den Komponenten wird eine fachliche Architektur entwickelt und beschrieben. Das Kapitel schließt mit einem Oberflächenentwurf.

Im Kapitel Entwurf und Implementierung (Kapitel 5) wird das zum Entwurf genutzte iterative Vorgehensmodell beschrieben und gegenüber den zur Verfügung stehenden Alternativen abgegrenzt. Danach werden die wesentlichen Eigenschaften des entworfenen Frameworks beschrieben und diskutiert. Unter anderem sind die Details zur technische Architektur, den definierten Erweiterungspunkten (Hooks), dem im Framework verwendeten Eventsystem und der API.

Die Arbeit schließt mit einer Zusammenfassung der wesentlichen Aspekte und einem Ausblick auf mögliche Weiterentwicklungen für zukünftige Arbeiten und die Forschung.

2. Grundlagen

In diesem Kapitel werden mit Blick auf die Fachliteratur Aspekte beleuchtet, die das zu entwickelnde Framework beeinflussen. In Abschnitt 2.1 wird auf die fachlichen Grundlagen eingegangen. Zuerst wird eine Übersicht über die Logistik und ihre Funktionen im Allgemeinen gegeben, anschließend wird die für diese Arbeit relevante Transportfunktion der Logistik und ihre konkrete Ausprägung des Luftfrachttransportes näher beleuchtet. Abschnitt 2.2 beschreibt die Grundlagen des Frameworkdesigns. Zuerst wird eine Definition dafür, was ein Framework ist, entwickelt, um dann die Vor- und Nachteile bei Nutzung eines Frameworks zu diskutieren. Abschließend wird auf bewährte Methoden und Muster zur Frameworkentwicklung eingegangen.

2.1. Logistik

> „Logistik ist marktorientierte, integrierte Planung, Gestaltung, Abwicklung und Kontrolle des gesamten Material- und dazugehörigen Informationsflusses zwischen einem Unternehmen und seinen Lieferanten, innerhalb eines Unternehmens sowie zwischen einem Unternehmen und seinen Kunden."

(Schulte, 2008)

2.1.1. Übersicht

Die fachlichen Aspekte dieser Arbeit sind auf die Transportfunktion der Distributionslogistik fokussiert, hier soll trotzdem ein kurzer Überblick über alle Teile der Logistik gegeben werden, um später eine Einordnung in die Systematik der Logistik zu ermöglichen.

Aus dem oben stehenden Zitat lassen sich als Fachdisziplinen der Logistik **Beschaffungslogistik**, **Produktionslogistik** und **Distributionslogistik** formulieren (Dieter u. a., 2004). In neuerer Literatur kommt als vierte Disziplin die **Entsorgungslogistik** hinzu (Schulte, 2008). Die Fachdisziplinen stellen hierbei einen horizontalen Schnitt durch die Logistikkette dar (Logistikkette siehe Anhang D) (Schulte, 2008).

Die **Beschaffungslogistik** hat die Aufgabe, die Versorgung mit *Einsatzgütern* externer Lieferanten sicherzustellen. Hier sind unter anderem die Aufgaben der Bedarfsermittlung und Disposition, Warenannahme und Prüfung sowie Lagerhaltung/-verwaltung angesiedelt. Die Entwicklung der Beschaffungslogistik ist über die letzten Jahrzehnte von einer Zunahme der Komplexität und Vernetzung mit den Zulieferern geprägt. Ziel und Anspruch der Beschaffungslogistik haben sich in dieser Zeit von der reinen Versorgung mit Material gelöst, um durch die Steuerung der Einkaufsmärkte sowie einen durchgängigen Informations- und Materialfluss einen größeren Anteil am unternehmerischen Erfolg eines Konzernes zu haben (Schulte, 2008).

Der Übergang in die **Produktionslogistik** wird in der Literatur dort gesehen, wo die Wertschöpfung im Unternehmen durch die Kombination von Produktionsfaktoren beginnt. Die drei großen Aufgabenfelder der Logistik sind hier Fabrikplanung, Produktionsplanung und -steuerung (PPS) sowie interne Materialbereitstellung. Unter Fabrikplanung wird in der mittel- bis langfristigen Strategie eines Unternehmens die technisch einwandfreie und unter wirtschaftlichen Gesichtspunkten getroffene Neu- oder Erweiterungsplanung von Produktionsstätten verstanden. Die PPS ist im Gegensatz hierzu eine eher kurz- bis maximal mittelfristige Tätigkeit, deren Ziel zuerst die Planung der Produktion (Programmplanung, Mengenplanung, Kapazitätsplanung) und danach die Steuerung der laufenden Produktion (Auftragsveranlassung, Auftragsüberwachung) ist. Die kurzfristige Bereitstellung von Produktionsmaterialien aus Wareneingängen bzw. internen Lagern ist schließlich Aufgabe der internen Materialbereitstellung (Schulte, 2008).

Distributionslogistik beschrieb bis in die achtziger Jahre regelmäßig den letzten Schritt in der Logistikkette, in dem fertige Produkte zum Endkunden transportiert werden. Hierfür sind Standorte für Lager auszuwählen, Lagerhaltung zu betreiben, Auftragsabwicklung und Kommissionierung vorzunehmen sowie nicht zuletzt der Transport zu organisieren (Schulte, 2008).

Seit ca. 1990 geht die Logistikplanung über den absatzbezogenen Güterfluss hinaus und plant mit der **Entsorgungslogistik** auch die Rückführung von beim Endkunden entstandenen Abfällen mit ein. Hier ist unter den zumeist gesetzlichen Rahmenbedingungen der Aufbereitung und Entsorgung primär ein logistischer Prozess zu etablieren, durch den Produktrückstände/Abfälle vom Endverbraucher zurückgeführt werden können (Schulte, 2008).

Als Weiterentwicklung der Verknüpfung einzelner Unternehmensteile in der Logistikkette wird in Literatur und Praxis das Supply Chain Management eingesetzt. In der Literatur sind unterschiedliche Definitionen verbreitet (Potzner, 2008). Sinngemäß ist hier die Definition aus (Schulte, 2008) wiedergegeben: Die bewusst herbeigeführte Verknüpfung dieser Fachdisziplinen mit Orientierung auf die Bedürfnisse des Endkunden und dem Ziel einer besseren Abstimmung von Angebot und Bedarf führt zum Begriff des Supply Chain Management. Als

Weiterentwicklung der Logistikkette erzielen Unternehmen hierdurch Kosten-, Zeit- und Qualitätsvorteile durch die weitergehende Integration der eigenen Zulieferer und dem Nutzen von Spezialwissen über die eigenen Kunden.

2.1.2. Transportfunktion

> „Die Aufgabe der Transportfunktion in der Distributionslogistik liegt in dem Raumausgleich von Gütern innerhalb des Logistiksystems durch die Verwendung von geeigneten Transportmitteln und Verkehrsträgern."

(Dieter u. a., 2004)

> „Der Transport von Material und Waren dient der Überwindung räumlicher Distanzen und führt damit zu einer Ortsveränderung des Transportguts."

(Schulte, 2008)

In der Transportlogistik wird unterschieden zwischen unternehmensinternem und -externem Transport. Im Folgenden wird der für diese Arbeit relevante Teil des externen Transports beschrieben.

Die erforderliche Transportleistung wird mit einem Verkehrsträger unter Nutzung der zugehörigen Verkehrsinfrastruktur erbracht. Übliche Verkehrsmittel im Güterverkehr sind **Straße**, **Eisenbahn**, **Binnen-/Seeschifffahrt** und **Luftverkehr**. Die Verkehrsträger werden in der Literatur systematisch anhand der Kriterien Kosten und Verkehrswertigkeit verglichen. Abhängig vom gewählten Ziel kann so der optimale Verkehrsträger ausgewählt werden. Im Folgenden sind zuerst klassische Komponenten der Kosten und dann solche der Verkehrswertigkeit aufgeführt (Schulte, 2008).

Kosten:

- Frachtkosten
- Handlingkosten
- sonstige Logistikkosten

Leistungsfähigkeit:

- Massenleistungsfähigkeit
- Schnelligkeit
- Netzbildungsfähigkeit
- Berechenbarkeit

- Zeitliche Flexibilität
- Räumliche Flexibilität
- Sicherheit
- Umweltbeeinflussung

Ein Vergleich der einzelnen Verkehrsträger anhand dieser Kriterien zeigt, dass größere Flexibilität (Straße) bzw. Geschwindigkeit (Luft) mit höheren Kosten verbunden ist und in der Regel eine niedrigere Massenleistungsfähigkeit besitzt als der Transport per Schiff oder Bahn. In modernen Logistiknetzwerken findet daher häufig ein intermodaler Transport statt, bei dem für einzelne Streckenabschnitte unterschiedliche Verkehrsträger genutzt werden. Es entstehen zwar durch Umladevorgänge zusätzliche Wartezeiten, diese treten jedoch speziell bei zunehmender Strecke in den Hintergrund (Schulte, 2008), (Dieter u. a., 2004).

Als eine weitere mögliche Differenzierung der Verkehrsträger im intermodalen Transport hat sich ihre Nutzung in *Vor-*, *Haupt-* und *Nachlauf* bewährt. Während in Vor- und Nachlauf regelmäßig nur die Straße genutzt wird, bieten sich aufgrund der größeren Kapazitäten, regelmäßigeren Fahrpläne und größeren Reichweiten die anderen Träger fast nur für den Sammelladungsverkehr im Hauptlauf an (Dieter u. a., 2004).

2.1.3. Luftfracht

Im folgenden Abschnitt wird zuerst die Systematik des Luftfrachtverkehrs vorgestellt, um dann die abgrenzenden Eigenschaften der Luftfracht gegenüber anderen Verkehrsträgern zu definieren und ihre speziellen Vor- und Nachteile aufzuzeigen. In den letzten beiden Teilen des Abschnittes wird auf die für diese Arbeit besonders relevanten Probleme der Routenplanung und des Verzögerungsmanagements eingegangen.

Eine Systematik der Luftfracht wird in der Literatur anhand verschiedener Kriterien aufgebaut, unter anderem **Zweck**, **Flugplan**, **eingesetztes Material** und wahrgenommene **geschäftliche Aktivität**. Eine Unterscheidung nach dem **Zweck** teilt die Luftfrachtdienstleister in militärische und zivile. Die Unterscheidung erklärt sich aus der ursprünglichen Herkunft der Logistik aus dem militärischen Nachschubwesen. Die Teilung nach einem vorhandenen **Flugplan** spaltet die Dienstleister in solche, die ein Netzwerk mit regelmäßigen Verbindungen zwischen Flughäfen betreiben und solche, die individuelle Charterflüge mit Frachtmaschinen anbieten. Die Klassifizierung nach dem **eingesetzten Material** unterscheidet Frachdienstleister, die reine Frachtmaschinen einsetzen und solche, die ihren Transport im Frachtraum von Passagiermaschinen durchführen. Diese Unterscheidung wird vorgenommen, da im ersten Fall das Netzwerk der Airline auf die Bedürfnisse des Frachttransportes abgestimmt werden kann, während im zweiten Fall aufgrund der Preisstrukturen die Bedürfnisse

des Passagiertransportes primär sind und erst an zweiter Stelle der Frachttransport steht und die Netzwerke also regelmäßig an Passagier- und nicht an Frachtströmen ausgerichtet sind. Die Unterscheidung nach der wahrgenommenen **geschäftlichen Aktivität** teilt die Dienstleister, je nachdem ob die Fracht von Tür zu Tür oder von Station zu Station transportiert wird. Ob der Luftfrachtdienstleister also nur eine Dienstleistung im Rahmen des *Hauptlaufes* erbringt oder es sich um einen Generalunternehmer handelt, der eine gesamte Logistikkette anbietet (Schulte, 2008), (Grandjot u. a., 2007).

Während das Ergebnis dieser Arbeit später auch für andere Szenarien nutzbar sein wird, liegt der Schwerpunkt während der Entwicklung auf einem zivilen Luftfrachtdienstleister, der nach Flugplan Frachttransporte mit speziellen Maschinen durchführt, ein Beispiel für einen solchen Anbieter wäre das Unternehmen FedEx Express.

2.1.3.1. Vor- und Nachteile

Im Vergleich zu anderen Verkehrsträgern weisen Luftfrachtdienstleister spezifische **Nachteile** auf. So sind die Preisstrukturen häufig hochkomplex und individuell, um den Vergleich zwischen verschiedenen Anbietern im immer weiter liberalisierten Markt zu erschweren. Die Preise sind auch regelmäßig höher als bei gleicher Entfernung mit anderen Verkehrsträgern (Grandjot u. a., 2007), (Potzner, 2008). Die baulich bedingten Beschränkungen des Laderaumes in Flugzeugen wirken sich in Form von Beschränkungen des Gewichts und der Ausmaße von möglichen Transportgütern nieder. Der Transport von Gefahrgütern ist teilweise komplett ausgeschlossen (IATA, 2011). Die abschließende Beschränkung ist, dass am Zielort ein Flughafen vorhanden sein muss. Ist dies nicht der Fall, ist ein mehr oder weniger großer Nachlauf notwendig (Grandjot u. a., 2007).

Demgegenüber stehen die **Vorteile** des Luftfrachttransportes. Es muss zwar ein Flughafen vorhanden sein, im Gegensatz zu z. B. einem Straßennetz benötigt ein Flughafen jedoch relativ wenig unterstützende Infrastruktur und kann technisch fast überall gebaut werden, politisch stellt dieser Bau möglicherweise ein deutlich größeres Problem dar. Die Sicherheit von Gütern gegenüber Diebstahl und Verlust ist im Luftfrachtverkehr signifikant höher als bei anderen Verkehrsträgern, dies führt zu niedrigeren Versicherungsprämien. Die Integration, von Dienstleistungen, die die Nachverfolgung von Paketen in Echtzeit ermöglichen, ist bei den Luftfrachtdienstleistern weit fortgeschritten. Für einige Klassen von Gütern (verderbliche Produkte, lebende Tiere, Pharmaprodukte) ist die Geschwindigkeit im Luftfrachtbereich das ausschlaggebende Argument, ab einer gewissen Entfernung ist der Transport per Flugzeug die schnellstmögliche Option. Für kapitalintensive Güter senkt der schnelle Transport auch die Zinskosten durch Verringerung der zeitlichen Kapitalbindung (Schulte, 2008), (Grandjot u. a., 2007).

2.1.3.2. Routenplanung

Ziel der Routenplanung (Airline Schedule Generation) ist es, ein Netzwerk von Flughäfen, die durch die Flugzeuge einer Airline verbunden werden, so zu erstellen, dass der zu erwartende Gewinn der Airline maximiert wird (Cheng-Lung, 2010). Aufgrund der Komplexität dieser Aufgabe und der Probleme, gewisse Eingangsgrößen mit ausreichend Vorlauf festzulegen, wird die Routenplanung gewöhnlich in sequenziell durchzuführende Schritte geteilt. In der **Marktanalyse** wird der zu erwartende Bedarf geschätzt. Aufgrund dieser Schätzung wird ein **Flugplan** erstellt, der zeigt, welche Flughäfen verbunden werden sollen. In der **Flottenzuweisung** wird den im Flugplan bestehenden Routen ein *Flugzeugmuster* zugewiesen, aus dem in der **Flugzeugzuweisung** ein konkretes Flugzeug wird. Für diese *Umläufe* werden in der **Crewgenerierung** noch unpersonalisierte Besatzungen erstellt, die später in der **Crewzuweisung** durch konkretes Personal gefüllt werden (Erdmann u. a., 2001), (Cheng-Lung, 2010).

In der Literatur wird das Problem der Routenplanung häufig als Problem des Netzwerkdesigns verallgemeinert und mit den Mitteln der Graphentheorie gelöst. Als spezifische Lösungen für den Bereich der Luftfahrt werden nur Varianten der *Lagrangian relaxation* und ganzzahlige lineare Optimierung genannt (Erdmann u. a., 2001).

Während durch die Erweiterung des Resultates dieser Arbeit theoretisch alle oben genannten Schritte in die Simulation mit einbezogen werden könnten, konzentriert sich diese Arbeit auf die Schritte der Flugplanerstellung und Flugzeugzuweisung.

2.1.3.3. Verzögerungsmanagement

Das Netzwerk einer Airline unterliegt zum Zeitpunkt der Nutzung diversen Einflüssen, die zu Verzögerungen führen können. Diese Faktoren sind zum Teil integraler Bestandteil des Geschäftsbetriebes, wie etwa überraschend erhöhtes Passagier- oder Frachtaufkommen, und teilweise Störungen im üblichen Sinn des Wortes wie zum Beispiel technische Fehler am Flugmaterial oder Krankstände der Besatzungen. Beiden Arten von Störungen ist gemeinsam, dass sie stochastisch auftreten und schwer planbar sind (Cheng-Lung, 2010).

Die Auswirkungen dieser Störungen auf den operativen Ablauf innerhalb einer Airline zu minimieren, ist Aufgabe des Verzögerungsmanagements. Hier werden sowohl Störungen, die durch Pufferzeiten im Flugplan aufgeholt werden können, beobachtet, um zu vermeiden, plötzlich von einer Akkumulation kleinerer Störungen überrascht zu werden, die in ihrer Gesamtheit die eingeplante Toleranz des Netzwerkes überfordern. Im Verzögerungsmanagement werden auch geeignete Maßnahmen ergriffen, um nicht mehr durch Puffer zu beseitigende Störungen abzufangen. Dem Verzögerungsmanagement stehen hierzu verschiedene Möglichkeiten zur Verfügung, unter anderem der Einsatz von

Standbycrews, die Verzögerung von Anschlussflügen, damit Passagiere oder Fracht den Anschlussflug noch erreichen oder das komplette Absagen eines Fluges. Diese Maßnahmen befinden sich in einem Spannungsfeld, in dem durch die operative Leitung beurteilt werden muss, welchen Zielen Priorität gegeben wird. Die Disponenten müssen zwischen der Gewinnmaximierung der Airline und der Zufriedenheit der Kunden ein Gleichgewicht finden (Cheng-Lung, 2010), (Ball u. a., 2006).

2.2. Framework

Frameworks sind vorgefertigte Teile einer Anwendung/Anwendungsarchitektur, mit deren Hilfe ein Entwickler eine Anwendung realisiert. Die verwendete Terminologie ist dabei unterschiedlich. Fayad u. a. (1999) definiert ein Framework als das Skelett einer Anwendung, das vom Anwendungsentwickler ausgefüllt wird. In Schmid (1997) wird ein Framework als eine generische Anwendung beschrieben, die es erlaubt, verschiedene Anwendungen aus einer fachlichen Domäne zu erstellen.

Für diese Arbeit wurde die Definition aus Schmid (1997) übernommen:

> „A framework is a generic application that allows the creation of different applications from an application (sub)domain."

Hier soll ein Framework für die Entwicklung und Evaluation von Algorithmen aus dem Bereich der Luftfrachtlogistik entwickelt werden.

Allen Definition gemeinsam ist, dass ein Framework im Detail beschreibt, wie sich ein System aus Objekten bzw. Modulen und ihren Interaktionen zusammensetzt. Hierzu wurden und werden formale Methoden der Schnittstellenbeschreibung entwickelt (Fayad u. a., 1999).

Mit diesen Formalismen werden im Framework die möglichen Stellen der Erweiterung beschrieben, an denen Anpassungen für die konkrete Funktionalität vorgenommen werden können (Buschmann u. a., 2004). An diesen Stellen eingesetzte, formal beschrieben Entwurfsmuster erlauben es, die sonst steile Lernkurve bei Verwendung eines neuen Frameworks abzuflachen (Flores und Aguiar, 2008).

2.2.1. Klassifikation

Frameworks werden in verschiedenen Dimensionen klassifiziert. In der Literatur sind unter anderem Klassifikationen nach Art der Erweiterung des Frameworks, der genutzten Technologie (Fayad u. a., 1999) und dem Anwendungsbereich verbreitet (Shan und Hua, 2006), (Fayad u. a., 1999).

Die Klassifikation nach Art der Erweiterung unterscheidet zwischen den alternativen Whitebox- und Blackbox-Frameworkdesigns sowie verschiedenen Abstufungen dazwischen. In der Literatur gebräuchlich sind:

- Whitebox
- Glassbox
- Graybox
- Blackbox

In einem **Whitebox-Framework** (auch vererbungsbasiert) wird vom Anwender des Frameworks erwartet, dass er das Verhalten des Frameworks durch Spezialisierung von Frameworkklassen anpasst, indem er vordefinierte Hook-Methoden überschreibt. Der Entwickler muss hier Kenntnisse über Design und Implementierung des Frameworks besitzen (Fayad u. a., 1999), (Schmid, 1997).

Im Gegensatz hierzu wird das Verhalten eines **Blackbox-Framework** durch Komposition verändert. Das Framework bringt für die variablen Aspekte der Problemdomäne verschiedene fertige Lösungen mit und wird durch die konkrete Auswahl von Lösungen parametrisiert.

Bei dieser Unterteilung handelt es sich nicht um eine binäre Unterscheidung, ein Framework enthält oft Komponenten, die durch Vererbung erweitert werden und solche, die durch Komposition verändert werden können (Fayad u. a., 1999). Abbildung 2.1 verdeutlicht das Kontinuum, auf dem sich diese Klassifikation bewegt.

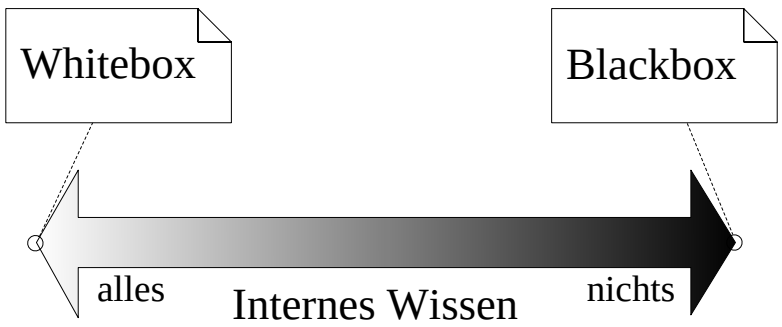

Abbildung 2.1.: Kontinuum von Whitebox- zu Blackbox-Framework

Als weitere Klassifikation existieren deshalb Glassbox und Greybox. Ein **Glassbox-Framework** verbirgt seine interne Struktur nicht und erlaubt dem Entwickler dadurch, sie zu studieren, um den Umgang mit dem Framework zu erlernen. Ein **Graybox-Framework** zeigt nur kontrollierte Aspekte seiner Implementierung (Szyperski u. a., 2002). In neuerer Literatur bezeichnet der Begriff der Greybox auch ein Framework, das Elemente von Whitebox und Blackbox vermischt (Flores und Aguiar, 2008).

Im Zusammenhang mit Blackbox- und Whitebox-Frameworks führt die Literatur den Begriff des „Hot Spots“ als Bezeichnung für einen bekannten Ort bekannter Variabilität im Framework ein. Abbildung 2.2 zeigt, wie die Variabilität für eine Blackbox im Framework enthalten ist und ausgewählt wird, während sie für eine Whitebox außerhalb des Frameworks implementiert wird. Für das konkrete Design eines Hot Spots bieten sich bekannte Design Pattern als bewährte und bekannte Lösungen an (Schmid, 1997).

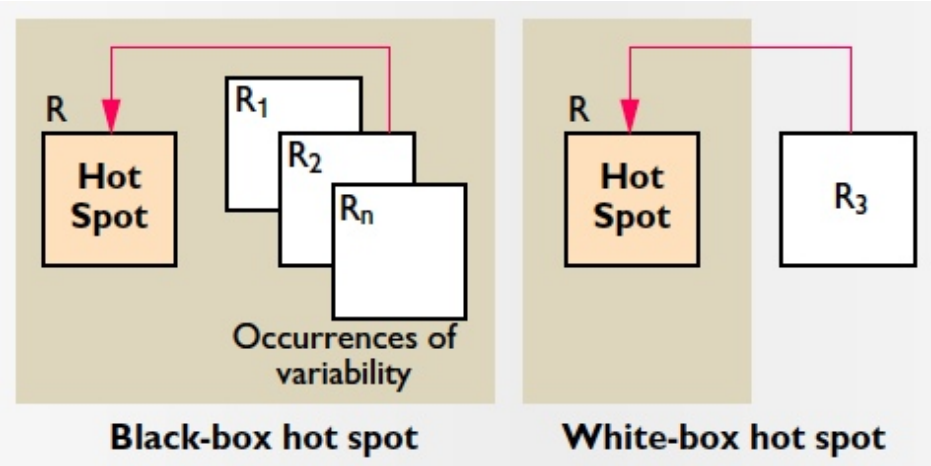

Abbildung 2.2.: Hot Spots in Blackbox- und Whitebox-Frameworks (Schmid, 1997)

Die Klassifikation nach Anwendungsbereich wie sie Fayad u. a. (1999) vornimmt, stellt 3 Klassen von Frameworks auf:

- System infrastructure
- Middleware integration
- Enterprise application

System infrastructure frameworks vereinfachen die Entwicklung von portabler und effizienter Softwareinfrastruktur, wie zum Beispiel Betriebssystemen und Frameworks für graphische Nutzeroberflächen. Als Teil der Infrastruktur haben sie üblicherweise keinen Kontakt mit Endkunden. Häufig sind diese Frameworks organisationsintern (Fayad u. a., 1999).

Middleware integration frameworks werden benutzt, um verteilte Anwendungen und Komponenten zu verbinden. Sie verbergen bestmöglichst vor dem Anwendungsentwickler, dass seine Anwendung in einer verteilten Umgebung genutzt wird (Fayad u. a., 1999).

Enterprise application frameworks dienen direkt der Entwicklung von Anwendungen in einer bestimmten fachlichen Domäne. Verglichen mit den anderen Arten von Frameworks sind sie in der Entwicklung teuer und aufwendig, liefern dafür aber direkter als andere Frameworks Einnahmen, da sie direkt der Entwicklung von Anwendungen für Endnutzer dienen (Fayad u. a., 1999).

Andere Autoren, wie Shan und Hua (2006) verwenden granularere Unterteilungen in 7 Klassen:

Conceptual Framework: übergreifendes Architekturmodell, z. B. Zachman Framework

Application Framework: Skelettstruktur für eine Anwendung, z. B. WebWork

Domain Framework: Zugeschnitten auf eine Domäne, z. B. IBM Information Framework

Platform Framework: Programmiermodell und Laufzeitumgebung, z. B. .Net oder Java EE

Component Framework: Einzelne Komponenten zur Entwicklung z. B. Hibernate, Mina

Service Framework: fachliche und technische Modelle zur Entwicklung, von serviceorientierten Anwendungen

Development Framework: Konstruktionsgrundlagen um rich-client Entwicklungsumgebungen aufzubauen, z. B. Eclipse oder Netbeans

Das in dieser Arbeit zu entwerfende Framework ist in den Bereich Application Framework/Enterprise application frameworks einzuordnen, da es sich um die Entwicklung einer Skelettstruktur handelt, in die nur noch spezifische Komponenten eingebracht werden sollen.

Eine Klassifikation nach Art der Erweiterung ist zu diesem Zeitpunkt noch nicht möglich. Die Erfahrung zeigt aber, dass die meisten neuen Frameworks eine Whitebox sind und sich mit wachsendem Verständnis für die fachlichen Probleme in Richtung Blackbox bewegen (Fayad u. a., 1999).

2.2.2. Vorteile und Probleme beim Einsatz von Frameworks

2.2.2.1. Vorteile

Der Einsatz eines Frameworks in der Anwendungsentwicklung bietet dem Einsetzenden Vorteile bei der **Modularisierung**, **Erweiterbarkeit** und der **Wiederverwendbarkeit** (Fayad u. a., 1999). Außerdem führt die Verwendung eines Frameworks zur **Beschleunigung** des

Entwicklungsprozesses für spezifische Anwendungen, die das Framework nutzen (Szyperski u. a., 2002).

Das vom Framework vorgegebene Design mit seiner Trennung zwischen variablen und nicht variablen Teilen fördert die **Modularität** der mit dem Framework entwickelten Anwendung. Änderungen werden lokalisiert und können durchgeführt werden, ohne die gesamte Applikation anpassen zu müssen (Fayad u. a., 1999). Die Modularisierung erleichtert bei vorhandenem Verständnis des Frameworks die Einarbeitung in neue Anwendungen (Flores und Aguiar, 2008).

Erweiterbarkeit der mit einem Framework entwickelten Applikationen entsteht durch das Explizitmachen von sogenannten Hook-Methoden als Anknüpfungspunkte. Diese Methoden entkoppeln auf systematische Art und Weise die stabilen Interfaces und abstrakten Klassen einer fachlichen Domäne von den notwendigen Anpassungen, die eine konkrete Anwendung vornehmen muss (Fayad u. a., 1999). Ausdruck der Wiederverwendbarkeit ist auch die gesteigerte Produktivität und die dadurch erreichbare **Beschleunigung** der Entwicklung (Szyperski u. a., 2002), (Fayad u. a., 1999).

Der Einsatz eines Frameworks ist definiert durch die **Wiederverwendung** der vom Framework in Form von abstrakten Klassen und Frameworks mitgebrachten Lösungen. Hierdurch müssen die im Framework bereits gelösten Probleme nicht erneut modelliert und implementiert werden (Fayad u. a., 1999). Wiederverwendbarkeit entsteht auch für die Fähigkeiten der Entwickler. Durch die Verwendung des Frameworks als Exemplar entsteht zwischen Entwicklern, die mit demselben Framework gearbeitet haben, ein gemeinsames Gefühl für Design (P. Brooks, 2010). Es entsteht eine Community, die weitere Entwicklungen mit einem Framework und dem Design, das es verkörpert, fördert (Sharp und Robinson, 2005). Diese machen es für ein Unternehmen leichter, offene Stelle zu besetzen, wenn ein in der Industrie verbreitetes Framework verwendet wird.

2.2.2.2. Probleme

Mögliche Probleme beginnen bei der Auswahl des korrekten Frameworks. Der Prozess, mit dem genügend Erfahrung und Sicherheit im Umgang mit einem Framework erlangt wird, um korrekt bewerten zu können, ob ein konkretes Problem mit ihm gelöst werden kann, ist in der Regel zeitintensiv und kostspielig (Froehlich u. a., 2000).

Neben dem Problem der Auswahl bringt die Entwicklung mit einem Framework häufig spezifische Probleme mit sich. In der Literatur werden unter anderem **Einarbeitungszeit**, **Integration** und **Debugging** genannt (Fayad u. a., 1999), (Flores und Aguiar, 2008).

Beim ersten Einsatz eines speziellen Frameworks zur Anwendungsentwicklung muss nicht nur das für die Anwendung notwendige fachliche Wissen strukturiert und modelliert werden, sondern auch auf die entsprechenden Komponenten des Frameworks abgebildet werden. Neben dem Lernprozess für die Fachlichkeit steht also ein weiterer Lernprozess für die Strukturen und das Design des Frameworks, der ebenfalls **Einarbeitungszeit** benötigt (Flores und Aguiar, 2008).

Ein Framework ist in der Regel nicht alleine im Einsatz, sondern wird häufig parallel zu anderen Frameworks eingesetzt. Wenn diese Frameworks nicht kompatible Designs besitzen oder nicht mit dem Gedanken an eine spätere **Integration** entwickelt wurden, ist die Entwicklung von Adaptern (Gamma u. a., 1995) notwendig, um eine Integration herbeizuführen (Fayad u. a., 1999).

Inversion of Control ist Bestandteil eines Frameworks (Fayad u. a., 1999). Die Abgabe des Kontrollflusses an das Framework erschwert das **Debugging** einer mit dem Framework entwickelten Anwendung, da der Entwickler im idealen Fall nichts von der Implementierung des Frameworks wissen will/soll. Wenn ein Fehler aber nicht eindeutig der Applikation oder dem Framework zugeordnet werden kann, muss die interne Logik des Frameworks verstanden und Fehlerbeseitigung in ihr betrieben werden (Fayad u. a., 1999).

2.2.2.3. Zusammenfassung

Der Einsatz eines Frameworks bietet neben Vorteilen auch die gezeigten Risiken. Die Entscheidung, ob und welches bestehende Framework genutzt wird oder sich eine Eigenentwicklung lohnt, ist immer situationsabhängig. Erprobte Kriterienkataloge zur Unterstützung dieser Entscheidung sind noch nicht vorhanden (Fayad u. a., 2000). Zusammenfassend ist festzuhalten, dass es sich bei der Entscheidung für ein Framework immer um eine langfristige Entscheidung handelt, die gut geplant werden sollte (Fayad u. a., 2000).

2.2.3. Frameworkdesign/-entwicklung

Die Literatur unterteilt den Lebenszyklus eines Frameworks in die drei Phasen **Initial Design**, **Exploratory Design** und **Design Generalization** (Foote, 1991).

Im **Initial Design** wird in einem „schnellen Wurf“ ein erster Prototyp entwickelt, dessen Ziel die erstmalige Lösung eines fachlichen Problems ist. In dieser Phase stehen spätere Ziele wie Wiederverwendbarkeit und robustes Design im Hintergrund. Die darauffolgende Phase des **Exploratory Designs** hat zum Ziel, durch die Entwicklung weiterer Anwendungen mit dem Framework die Hot Spots zu identifizieren. Meistens entstehen in diesem Stadium der Entwicklung breite vererbungsbasierte Klassenhierarchien, die noch nicht generell

anwendbar sind. Die Verallgemeinerung dieser Hierarchien findet in der **Design Generalization** statt. Mit dem Wissen der vorhergehenden Phasen werden die Anforderungen an das Framework erneuert und die entstandenen Klassenhierarchien neu strukturiert (Foote, 1991). Als Teil dieser Umstrukturierung sollte eine Entwicklung in Richtung Blackbox stattfinden (Foote, 1988).

In (Fayad u. a., 1999) wird dieses Vorgehen detaillierter ausgeführt und verbessert. Hier findet die Entwicklung eines Frameworks nach dem initialen Entwurf in Zyklen statt. Durch Beschränkung jeder Iteration auf entweder das Hinzufügen neuer Funktionalität oder neuer Hot Spots zum System wird versucht, die Komplexität des jeweils zu lösenden Problems zu begrenzen. Bei nicht vollständigem Verständnis der fachlichen Domäne können Hot Spots organisch wachsen, dann sind Zyklen mit einem Fokus auf der Restrukturierung der so gewachsenen Funktionalität einzuplanen. Oder wenn die fachliche Domäne wohldefiniert ist, sollten die Hot Spots systematisch eingeführt werden. Für das systematische Vorgehen bei der Generalisierung beschreibt (Fayad u. a., 1999) ein 4-Phasen-Modell:

1. Hot Spot Analyse
2. Highlevel Hot Spot Design
3. Detailliertes Hot Spot Design/Implementierung
4. Test/Evaluation

Zusammenfassend ergibt sich nach (Foote, 1991), (Fayad u. a., 1999) und (Schmid, 1997) folgende Herangehensweise an die Frameworkentwicklung:

1. Modellierung und Implementierung einer Beispielanwendung
2. Zyklische Erweiterung um Funktionalität oder Hot Spots durch das oben beschriebene systematische Vorgehen
3. Refactoring

Teil II.

Hauptteil

3. Anforderungsanalyse

Im Analysekapitel wird ein Anwendungsszenario für die Routenplanung und Simulation beschrieben. Aus dieser Beschreibung werden die Stakeholder des Prozesses herausgearbeitet, um im folgenden Spezifikationskapitel formale Anwendungsfälle und Anforderungen herausarbeiten zu können.

3.1. Stakeholder

In diesem Abschnitt sollen die verschiedenen Stakeholder an der Anwendung beschrieben werden. Ihre Motivation und ihre Fähigkeiten bilden die Grundlage für Anwendungsszenarien und Anforderungen.

3.1.1. Szenarioplaner

Der Szenarioplaner ist (vorausgesetzt es existieren schon Algorithmen) der erste an der Routenplanung Beteiligte. Seine Aufgabe ist es, Stammdaten einzupflegen. Er ist also derjenige, der die Rahmendaten der Simulation festlegt. In dieser Position muss der Szenarioplaner die Daten über Positionen von Flughäfen, Flugzeugtypen und Flottendaten in eine maschinenlesbare Form umsetzen. Er erstellt noch keine neue Daten, sondern setzt nur bestehende Daten um.

3.1.2. Algorithmen-Entwickler

Der Algorithmen-Entwickler erstellt unter Nutzung der API Algorithmen zur Planung und Umplanung des Flugnetzes, die der Routenplaner in seinen Simulationen nutzt. Ein typischer Ablauf für ihn ist das Erstellen einen Algorithmus bzw. dessen Weiterentwicklung auf Feedback des Routenplaners.

3.1.3. Routenplaner

Der Routenplaner übernimmt die Daten des Szenarioplaners, erweitert sie um Prognose- und Echtdaten zum Transportbedarf sowie Störungen. Er kann, wenn er Planungsalgorithmen für die ursprüngliche Planung und die iterative Neuplanung bei Störungseintritt ausgewählt hat, verschiedene Szenarien mithilfe der Anwendung simulieren und auf ihr betriebswirtschaftliches Ergebnis untersuchen.

3.1.4. Disponent

Aufgabe des Disponenten ist es, im operativen Betrieb die Auswirkung von Störungen zu minimieren. Auftretende Störungen (z. B. verspäteter Abflug, längere Flugdauer) sind von ihm, abhängig von erwarteter Dauer und Auswirkungen, zu bewerten und durch geeignete Umplanungen im Netzwerk zu minimieren. Für diese Umplanungen stehen dem Disponenten unter der Maßgabe vorgegebener Ziele (z. B. Umsatzmaximierung, Minimierung der Flugplanabweichung) und der eingeplanten Reserven verschiedene Optionen zur Verfügung, wie zum Beispiel der Einsatz von in Bereitschaft gehaltener Crews und Flugzeuge.

3.2. Anwendungsszenarien

Im folgenden Abschnitt werden zwei Anwendungsszenarien für eine mit dem hier zu entwerfenden Framework entwickelte Anwendung gezeigt. Zuerst wird auf die Anwendung der Simulation mit Planung eingegangen und anschließend auf den Bereich der Algorithmen-Erstellung.

3.2.1. Simulation

Für einen abgeschlossenen Wirtschaftsraum soll ein möglichst optimaler Routenplan [2] für die nächste Geschäftsperiode entwickelt werden. In einem ersten Schritt werden von einem Szenarioplaner die durch die Umwelt vorgegebenen Stammdaten angelegt. Es werden also Daten über Flugzeugtypen, Flughäfen und die verfügbare (Flugzeug-) Flotte eingepflegt. Hierbei handelt es sich um Daten, die schon bestehen und nur noch umgewandelt werden müssen.

[2] Ein optimaler Routenplan ist in der Regel aufgrund der NP-Vollständigkeit des Routeplanungsproblems nicht in realistischer Zeit erreichbar.

Im nächsten Schritt wird der Szenarioplaner die Daten erstellen, bei denen er seine eigene Diskretion und Erfahrung nutzt, um zu möglichst realistischen Vorhersagen zu kommen. Er wird also zum Beispiel prognostizierte Bedarfe anlegen.

Nachdem diese Stammdaten vom System geprüft wurden, wird als nächstes der Routenplaner aktiv. Er wählt einen Planungsalgorithmus für die initiale Planung aus und führt diese durch. Wenn dieser initiale Plan erfolgreich erstellt wurde, stellt der Routenplaner weitere Stammdaten (Störungen, reale Bedarfe) sowie den Algorithmus zur Störungsbehandlung ein. Er führt dann die Simulation durch und bewertet das von der Simulation ausgegebene wirtschaftliche Ergebnis.

Wenn er noch weiteren Planungsbedarf sieht, simuliert er mit veränderten Parametern erneut, bis er eine unter seinen Annahmen (Planungsalgorithmen) optimale Planung gefunden hat.

3.2.2. Algorithmen erstellen

Wenn der Routenplaner Probleme mit den vorhandenen Algorithmen feststellt, zum Beispiel Verbesserungspotenzial, das nicht über vorhandene Einstellungen realisiert werden kann, benötigt er andere (neue) oder angepasste Algorithmen, um seine Planung zu optimieren. Der Algorithmen-Entwickler implementiert den vom Routenplaner spezifizierten Algorithmus oder Änderungen unter Nutzung der API. Er stellt die fertige Implementierung für den Planer zur Verfügung.

3.3. Fallbeispiel

In diesem Abschnitt wird an einem konkreten Beispiel ein Anwendungslauf der Planung und der Simulation aus fachlicher Sicht gezeigt, um aus den gewonnenen Erkenntnissen später Parameter der Spezifikation und des Entwurfes abzuleiten.

In der Realität wird die Planung eines Netzwerkes bis zum Abflug üblicherweise in die 4 Phasen: Generierung eines Zeitplanes, Flottenzuweisung, Flugzeug Routing, Zuweisung der Crews unterteilt, die sequenziell im Zeitraum von 6 Monaten bis 30 Tagen vor dem Flug durchgeführt werden, siehe Kapitel 2.1.3.2.

Das Fallbeispiel konzentriert sich, um überschaubar zu bleiben, auf die Planung des Netzwerkes und die Zuweisung von Flugzeugen. Dijkstra's Algorithmus (vgl. Tanenbaum, 2003, S. 353-355) wird in modifizierter Form genutzt, um Routen mit den größten Gewinnen zu finden. Diese werden anschließend auf die verfügbaren Flugzeuge verteilt. Dijkstra's Algorithmus

wurde hier als ein bekannter und relativ simpler Vertreter aus der für dieses Problem häufig genutzten Klasse der graphentheoretischen Algorithmen gewählt (Erdmann u. a., 2001).

In der Planungsphase werden einige übliche (vgl. Erdmann u. a., 2001) Beschränkungen übernommen sowie weitere eingeführt, um das Problem leichter handhabbar zu machen.

3.3.1. Planungsalgorithmus

3.3.1.1. Einschränkungen

1. Assoziierte Heimatbasen: Jedes Flugzeug muss nachts auf einer bestimmten Basis sein.
2. Heimatländer: Heimatbasen müssen im Herkunftsland der Airline liegen (hier Deutschland).
3. Tägliche Routen: Die Streckenführung wiederhohlt sich jeden Tag.
4. Keine Treibstoffbeschränkung: Alle Flugzeuge können alle Strecken ohne Zwischenlandung fliegen.
5. Zyklische Bedarfe: Transportbedarfe zwischen 2 Flughäfen sind regelmäßig in ungefähr vergleichbarer Größe vorhanden.
6. Gewichtsklassen: Flugzeuge können sinnvoll in Gewichtsklassen eingeteilt werden.

3.3.1.2. Algorithmus

Die prognostizierten Frachtwerte pro Strecke werden über den Planungszeitraum gemittelt[3] um einen durchschnittlichen Frachtwert pro Tag zu erhalten. Hierbei werden anhand der Flugzeugladekapazitäten Cluster gebildet um die möglichen Kombinationen einzuschränken. Mit diesen Gewichtungen der Strecken wird dann Dijkstra's (vgl. Cormen u. a., 2009, S. 658-664) Algorithmus verwendet, um ein Routennetz mit dem maximalen Ertrag zu finden, siehe auch Abbildung 3.1.

Die Gewichtsklassen werden von schwer nach leicht abgearbeitet. Es wird also z. B. die optimale Strecke für 100 Tonnen gesucht, dann die zweitbeste für 100 Tonnen, dann die beste für 80 Tonnen, wobei die jeweils vorher verplanten Frachtsegmente als nicht mehr vorhanden betrachtet werden.

[3] arithmetisches Mittel

Im Gegensatz zum Standard Dijkstra's wird das mehrfache Besuchen eines Knotens (Flughafens) erlaubt, sofern dieser vorher verlassen wurde. Außerdem ändert sich für alle folgenden Operationen das Gewicht einer Kante durch den Transport um den Wert des Transportauftrages.

Während der Algorithmus läuft, wird die verbrauchte Zeit (Flugzeit + Turnaround) aufsummiert. Der Algorithmus bricht ab, wenn für alle möglichen Rückwege die Reisezeit größer als die vom Tag verbleibende Zeit ist. Abbildung 3.1 zeigt den rekursiv aufgerufenen Teil des Algorithmus.

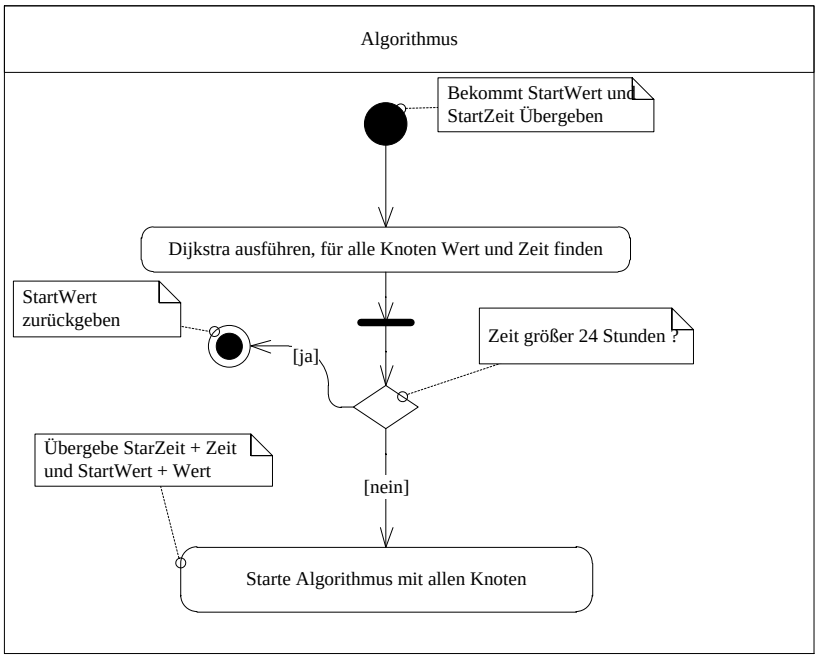

Abbildung 3.1.: Fallbeispiel Planungsalgorithmus

3.3.2. Störungsalgorithmus

Beim Auftreten von Störungen wird folgender Algorithmus verwendet:

1. Markiere alle Flugzeuge, die betroffen sind (direkt oder weil sie gestörte Routen, Flughäfen nutzen).
2. Suche für alle gestörten Flugzeuge den kürzesten Weg zur nächsten nicht gestörten Station.

3. Ist dieser Weg schneller als der ursprünglich geplante?

   a) Ja: Setze die ursprüngliche Planung von diesem Flughafen aus fort.

   b) Nein: Kehre zurück zum Startflughafen, nehme die Planung von dort aus am nächsten Tag wieder auf.

3.3.3. Startsituation

Für das Fallbeispiel sei die im Folgenden näher beschriebene Startsituation aus 4 Flughäfen, 2 Flugzeugen und den dargestellten durchschnittlichen Ladungswerten pro Tag gegeben.

3.3.3.1. Flughäfen

Es sind die in der unten stehenden Tabelle 3.1 geführten 4 Flughäfen vorhanden (HAM, LHR, LBC, LBA).

3LC	Stadt	Latitude	Longitude	Turnaround Klasse 2 (min)
HAM	Hamburg	53.630278	9.991111	90
LBC	Lübeck	53.805278	10.719167	120
LHR	London	51.4775	-0.461389	95
LBA	Leeds	53.865833	-1.660556	60

Tabelle 3.1.: Fallbeispiel Startzustand Flughäfen

3.3.3.2. Strecken

Das Streckennetzwerk zwischen Flughäfen verbindet Hamburg mit London, London mit Leeds und Leeds mit Lübeck. Details (Bezeichnungen und Entfernungen) zu den Strecken sind in der folgenden Tabelle 3.2 aufgeführt.

3.3.3.3. Flugzeuge

In der Planung sind 2 Flugzeuge enthalten, eines startet in Hamburg und hat eine Zuladung von 100 t (Klasse 2) und eines in London mit 80 t Zuladung (Klasse 2) (siehe Tabelle 3.3).

Name	Start	Ziel	Strecke (km)	Reisezeit (min)
LBA_LHR	Leeds	London	274	30
HAM_LHR	Hamburg	London	719	85
LBA_LBC	Leeds	Lübeck	801	110
HAM_LBA	Hamburg	Leeds	760	90

Tabelle 3.2.: Fallbeispiel Startzustand Routen

Tailsign	Zuladung (t)	Klasse	Kosten pro Tag
D-EINS	80	2	100
D-ZWEI	60	2	60

Tabelle 3.3.: Fallbeispiel Startzustand Flugzeuge

3.3.3.4. Prognostizierte Bedarfe

In der Tabelle 3.4 sind die durchschnittlichen Ladungswerte pro Tag.

ID	Start	Ziel	Gewicht (t)	Einnahme pro Tonne
1	HAM	LHR	40	40
2	HAM	LHR	80	10
3	HAM	LHR	200	25
4	LHR	HAM	100	12
5	LHR	HAM	50	20
6	LHR	LBA	15	30
7	LHR	LBA	200	5
8	LBA	LHR	225	2
9	LBA	LBC	30	100
10	LBC	LBA	50	40

Tabelle 3.4.: Fallbeispiel Startzustand Ladungen

3.3.3.5. Startsituation (Graphisch)

Graphisch stellt sich die Startsituation wie in Abbildung 3.2 gezeigt dar.

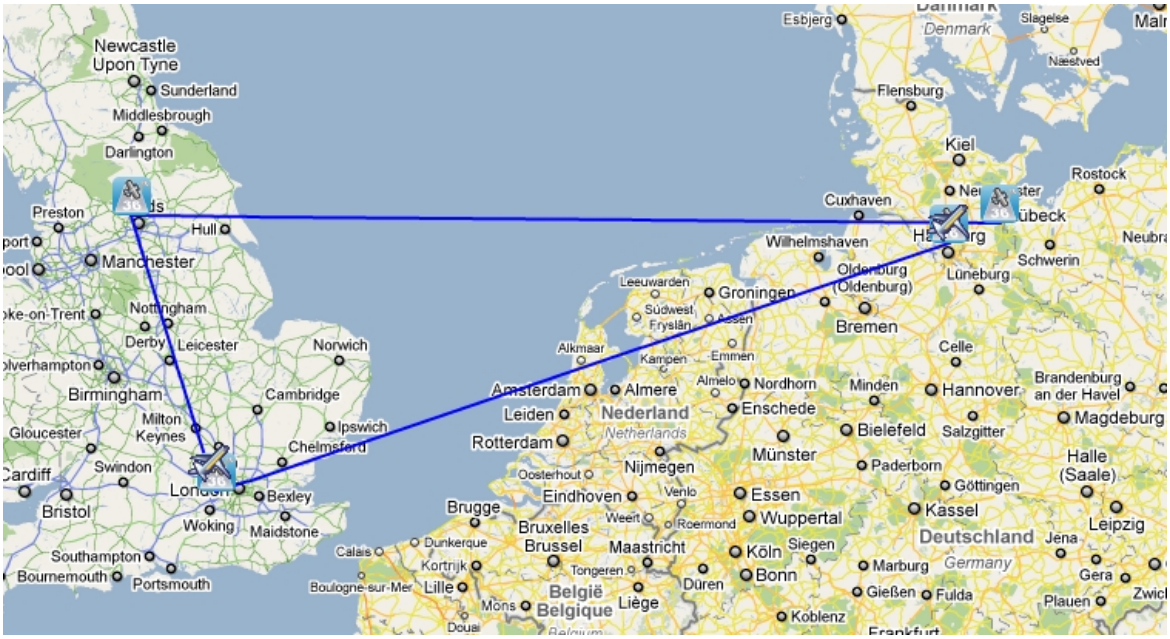

Abbildung 3.2.: Oberflächenmock Fallbeispiel Startsituation (Copyright, Google Maps 2011)

3.3.4. Planungsphase

Der Algorithmus wurde hier nur für 8 Stunden ausgeführt, um die Übersichtlichkeit der Ergebnisse zu erhalten. Es ergeben sich die in 3.5 gezeigten Strecken.

ID	Strecke	Heimatbasis	Geplantes Transportgewicht	Einnahmen
1	LBC/LBA/LBC	LBC	100	5000
3	HAM/LBA/LBC	LBC	80	3960

Tabelle 3.5.: Fallbeispiel geplante Strecken

4. Spezifikation

Im Kapitel Spezifikation werden die Ergebnisse der Anforderungsanalyse (siehe Kapitel 3) genutzt und mit den Mitteln des Software Engineering **formalisiert**.

Hierzu werden im ersten Teil des Kapitels zuerst aus den Anwendungsszenarien **Anwendungsfälle** abgeleitet. Aus der Analyse der Anwendungsfälle ergeben sich **Anforderungen, Prämissen und Ausgrenzungen** an/für das zu entwerfende Framework.

Im zweiten Teil der Spezifikation wird aus den Analyseerkenntnissen (Kapitel 3, Vorlagen aus der Literatur und den Anforderungen) ein **fachliches Datenmodell** entwickelt. Anschließend wird auf diesem fachlichen Datenmodell ein **Komponentenschnitt** durchgeführt und die sich ergebenden Komponenten werden in die **fachliche Architektur** eingeordnet. Abschließend wird ein **GUI Mock** entworfen und mit seinen wesentlichen Elementen gezeigt.

4.1. Anwendungsfälle

Aus den Anwendungsszenarien (siehe 3.2) wurden Anwendungsfälle erzeugt und strukturiert beschrieben. Beispielhaft sind vier dieser Anwendungsfälle hier aufgenommen.

4.1.1. Eingabe der Stammdaten

Der hier dargestellte Anwendungsfall beschreibt die Eingabe von Stammdaten durch den Szenarioplaner. Dieser Anwendungsfall setzt sich zusammen aus der Eingabe der Stammdaten, der darauffolgenden Prüfung durch das System und eventueller Fehlerbehebung durch den Szenarioplaner.

Titel	**Eingabe der Stammdaten**
Akteur	**Szenarioplaner**
Ziel	**Stammdaten sind eingegeben**

Auslöser:

Als Vorbedingung für eine Routenplanung will der Szenarioplaner die notwendigen Stammdaten anlegen.
Vorbedingungen:
Keine
Nachbedingungen:
Stammdaten sind vorhanden und geprüft.
Erfolgsszenario:

1. Der Planer öffnet die Stammdatendatei in einem Editor.
2. Der Planer legt die Stammdaten an.
3. Der Planer startet die Datenverifizierung der Planungskomponente.
4. →Prüfen der Stammdaten

Erweiterungen:
Keine
Fehlerfälle:

4.a siehe: →Prüfen der Stammdaten

Häufigkeit:
Vor jedem Planungsvorgang
Anforderungen:
→Prüfen von Stammdaten

4.1.2. Prüfen der Stammdaten

Im Szenario „Simulation“ wird unter anderem beschrieben, wie der Szenarioplaner Stammdaten in die Applikation eingibt und diese vom System geprüft werden. Der hier beschriebene Anwendungsfall verdeutlicht, zu welchen Zeitpunkten und mit welchen Randparametern das Prüfen der Stammdaten vom System durchgeführt werden muss. Es wird der erfolgreiche Verlauf der Prüfung gezeigt, in diesem sind die Stammdaten sowohl syntaktisch als auch semantisch korrekt. Danach werden typische Fehlerfälle wie zum Beispiel die syntaktische Invalidität der Stammdaten mit ihren assoziierten Reaktionen beschrieben.

Titel	**Prüfen der Stammdaten**
Akteur	**Anwendung**
Ziel	**Stammdaten sind syntaktisch und semantisch korrekt**

Auslöser:
Eingeben der Stammdaten, Start der Planung und Start der Simulation.
Vorbedingungen:
Stammdaten sind vorhanden.
Nachbedingungen:
Die Stammdaten sind geprüft.
Erfolgsszenario:

1. Die Planungskomponente lädt die Stammdaten.
2. Die Planungskomponente verifiziert die Stammdaten auf syntaktische Korrektheit.
3. Die Planungskomponente verifiziert die Stammdaten auf semantische Korrektheit.
4. Die Planungskomponente gibt auf ihrem `stdout` eine Erfolgsmeldung aus.

Erweiterungen:
Keine
Fehlerfälle:

1.a Das Laden der Daten schlägt fehl. Die Stammdaten müssen angelegt werden.

2.a Die Prüfung der Daten auf syntaktische Korrektheit schlägt fehl. Die Daten können nicht verarbeitet werden. Die Fehler müssen korrigiert werden.

3.a Die Prüfung der Daten auf semantische Korrektheit schlägt fehl. Die Daten müssen korrigiert werden.

Häufigkeit:
Vor jedem Planungsvorgang, jeder Simulation und bei jeder Änderung der Stammdaten.
Anforderungen:
→Prüfen von Stammdaten

4.1.3. Start der Routenplanung

Nachdem die Stammdaten eingegeben und verifiziert sind, muss die Routenplanung gestartet werden. Dieser Anwendungsfall zeigt die Schritte dieser Planung vom Laden der Stammdaten bis zum Ausgeben der Planung in einer Datei sowie die dabei notwendigen Ausführungen anderer Anwendungsfälle wie zum Beispiel „Prüfen der Stammdaten“ vor dem Start der Planung. Ein typischer Fehlerfall an dieser Stelle ist das Laden des zur Planung genutzten Algorithmus. Wenn dieser Fehler auftritt, muss er durch den Anwender behoben werden.

Titel **Start der Routenplanung**

Akteur **Szenarioplaner**

Ziel **Planung eines Streckennetzes**

Auslöser:
Es soll ein Streckennetz geplant werden.
Vorbedingungen:
Stammdaten sind eingegeben.
Nachbedingungen:
Die Routenplanung ist in der Ausgabedatei abgelegt.
Erfolgsszenario:

1. Der Szenarioplaner startet in der Kommandozeile die Planungskomponente.
2. →Prüfen der Stammdaten
3. Die Planungskomponente. lädt den in den Stammdaten angegebenen Algorithmus.
4. Die Planungskomponente führt die Planung durch.
5. Die Planungskomponente schreibt die Planung mit den Stammdaten in ihre Ausgabedatei.

Fehlerfälle:

2.a siehe: →Prüfen der Stammdaten

3.a Das Laden des Algorithmus scheitert, die Planung bricht ab. Der Algorithmus muss bereitgestellt werden oder die Planung mit einem anderen Algorithmus gestartet werden.

Häufigkeit:
In erster Näherung wird eine Häufigkeit von 20 Ausführungen pro Tag erwartet.
Anforderungen:
→Einbringung von Algorithmen

4.1.4. Ausführung der Simulation

Die Ausführung der Simulation setzt voraus, dass die vorhergehenden Anwendungsfälle alle korrekt ausgeführt wurden. Nach dem Start der Simulation und der vorhergehenden Prüfung der Stammdaten wird die Oberfläche gestartet, um dem Anwender einen Einblick in die laufende Simulation zu geben. Die Simulation läuft dann, bis die erste Störung auftritt, um dann den konfigurierten Störungsalgorithmus auszuführen und die Planung an die neue Situation anzupassen. Parallel werden von der Simulation betriebswirtschaftliche Kennzahlen wie zum Beispiel die Kosten der Flugzeuge gesammelt, um diese und ihre Summen später zur Auswertung bereitzustellen.

Titel	**Start der Simulation**
Akteur	**Routenplaner**
Ziel	**Bewertung des Netzwerkes**

Auslöser:
Eine abgeschlossene Routenplanung soll simuliert werden.
Vorbedingungen:
Routenplanung ist erfolgt (siehe: →Start der Routenplanung).
Nachbedingungen:
Die Ergebnisse der Simulation (Gewinn/Verlust, Umplanungen) sind in ihrer Ausgabedatei abgelegt.
Erfolgsszenario:

1. Der Routenplaner startet in der Kommandozeile den Simulator.
2. →Prüfen der Stammdaten
3. Der Simulator lädt den in den Stammdaten angegebenen Algorithmus zur Weiterplanung bei Störungen.
4. Die Simulation startet die Oberfläche.
5. Die Simulation läuft bis zum nächsten Störungsevent.
6. Die Routenplanung wird durch den in 3. geladenen Algorithmus auf die veränderten Bedingungen angepasst.
7. Die beiden letzten Schritte wiederholen sich, bis das Ende der Simulationsperiode erreicht ist.

8. Die Simulation akkumuliert die während ihrer Aktivität gesammelten Datenpunkte über Betriebskosten und Einnahmen und schreibt die Rohdaten sowie ihre Auswertung in die Ausgabedatei.

9. Der Routenplaner beendet die Oberfläche.

Erweiterungen:
Keine **Fehlerfälle:**

2.a siehe: →Prüfen der Stammdaten

3.a Das Laden des Algorithmus scheitert, die Planung bricht ab. Der Algorithmus muss bereitgestellt werden oder die Planung mit einem anderen Algorithmus gestartet werden.

Häufigkeit:
In erster Näherung wird eine Häufigkeit von 20 Ausführungen pro Tag erwartet.

4.2. Anforderungen, Prämissen, Abgrenzungen

In diesem Kapitel werden ausgewählte **Anforderungen**, **Prämissen** und **Abgrenzungen** für das zu entwerfende Framework dargestellt. Eine vollständige Aufzählung findet sich in Anhang A[4].

Generell dienen **Anforderungen** dazu, eine Fähigkeit einer Anwendung zu beschreiben, die diese benötigt, damit der Nutzer ein Problem lösen oder ein Ziel erreichen kann. Ergänzend hierzu stellen **Prämissen** Voraussetzungen dar, von denen bei der Entwicklung ausgegangen werden kann, während **Abgrenzungen** solche Fähigkeiten beschreiben, die das zu entwickelnde System explizit nicht abbilden soll (Sarstedt, 2009).

Für diese Arbeit wurden die Anforderungen in die vier Kategorien **API**, **GUI**, **Stammdaten**, und **Störungen** unterteilt und erfasst. Anforderungen aus der Kategorie **API** beschreiben hier also technische Anforderungen an die API, die diese erfüllen muss, damit der Nutzer sie verwenden kann. Beispielsweise beschreibt die Anforderung: „Die API ist programmiersprachenunabhängig.“ die Tatsache, dass das fertige Framework eine API bieten soll, die sprachunabhängig ist. **GUI**-Anforderungen beschreiben die graphische Nutzeroberfläche der Simulation. Also welche Daten dem Anwender während des Simulationslaufes zur Verfügung stehen, um einen Einblick in den aktuellen Systemzustand zu gewinnen. Beispielsweise soll

[4] Die im Anhang gezeigten Anforderungen stammen aus einem frühen Stadium dieser Arbeit und sind nach umfassenderem Verständnis der fachlichen Zusammenhänge teilweise nicht oder abgeändert in spätere Entwicklungszyklen eingeflossen.

die GUI während der Laufzeit der Simulation die Position aller Flugzeuge anzeigen, was durch die Anforderung „Die GUI stellt Flugzeuge sowohl im Flug als auch am Boden dar.“ festgehalten wird.

Die Kategorien **Stammdaten** und **Störungen** beschreiben fachliche Details zum Umfang der Stammdaten und Störungen. Beispielsweise „Das System prüft Stammdaten auf syntaktische und semantische Korrektheit, bevor es sie verwendet.“ und „Es sind sowohl Flugzeuge am Boden als auch in der Luft für Störungen anfällig.“

Neben Anforderungen an das Framework wurden **Prämissen und Abgrenzungen** aufgenommen, um Bereiche abzugrenzen, die für diesen Entwurf nicht relevant sind. Beispielsweise definiert die Prämisse „Stammdaten werden mit Applikationen, die nicht Bestandteil der Anwendung sind, erstellt.“, dass die Erstellung der Stammdaten durch eine externe Applikation erfolgt und hier nicht betrachtet wird. Die Abgrenzung „Die Anwendung unterscheidet bei der Kapazität von Flughafen nicht zwischen verschiedenen Flugzeugtypen.“ beschreibt den Umstand, dass in der Realität die Kapazität eines Flughafen, ausgedrückt als Anzahl der Starts/Landungen pro Stunde, aufgrund der sich in Abhängigkeit vom Flugzeuggewicht und meteorologischen Rahmenbedingungen hinter Flugzeugen bildenden Wirbelschleppen abhängig vom Typ der startenden/landenden Flugzeuge ist. Dieser Umstand wird aber in der Simulation nicht beachtet, da er vom Kernproblem ablenken würde.

4.3. Fachliches Datenmodell

In diesem Kapitel werden zuerst das Vorgehen, mit dem das fachliche Datenmodell für diese Arbeit erstellt wurde, und anschließend das Datenmodell vorgestellt.

4.3.1. Vorgehen

Im **ersten** Schritt wurden in der **Literatur** (Luxem, 1993; Dieter u. a., 2004; Schulte, 2008) verschiedene Datenmodelle für die Abbildung von Logistikbetrieben in der IT analysiert, um schließlich auf der Basis des in Luxem (1993) gezeigten ausführlichen Datenmodells weiterzuentwickeln. In dieses Modell wurden im **zweiten** Schritt die Erkenntnisse aus der Analyse (siehe Kapitel 3) eingearbeitet, d. h. das Modell wurde um diese erweitert. Da sich nach dem zweiten Schritt Überschneidungen zwischen den neu eingebrachten Entitäten und den bereits vorhandenen zeigten bzw. die bereits vorhandenen Entitäten für diese Arbeit nicht von Bedeutung sind, wurde ein weiterer Bearbeitungsschritt zur Bereinigung des Modells durchgeführt. In diesem **dritten** Schritt wurde das Datenmodell auf die Aspekte reduziert, die für diese Arbeit unbedingt notwendig sind. Beispielsweise wurden die Entitäten „Werk“ und „Produktionsauftrag“ (vgl. Luxem, 1993, S. 87) entfernt. Abschließend wurden Namen und

Eigenschaften der verbleibenden Entitäten, die noch aus dem allgemeinen Modell stammten, auf die hier verwendete Terminologie der Luftfahrtbranche geändert.

4.3.2. Datenmodell

In diesem Abschnitt werden die Entitäten des fachlichen Datenmodells mit ihren wesentlichen Eigenschaften beschrieben. Beispielhaft wird in Tabelle 4.1 eine vollständige Spezifikation der Entität „Flugzeug“ gezeigt. Eine vollständige Spezifikation für alle Entitäten ist in Anhang C enthalten.

Entität	**Flugzeug**
Attribute	• TailNumber: AlphNum • Kosten: GeldTyp • Geschwindigkeit: Float • Klasse: TransportKlasse • Position: PositionsTyp • Kapazität: KapazitätsTyp
Schlüssel	• TailNumber
Beziehungen	• Flugzeuge nutzen Routen • Flugzeuge nutzen Flughäfen • Flugzeuge erfühlen durch Transport Bedarfe • Flugzeuge werden durch Störungen betroffen

Tabelle 4.1.: Entität: Flugzeug

Abbildung 4.1 enthält als Entitäten des fachlichen Datenmodells **Flugzeug**, **Flughafen**, **Route**, **Störung**, **Bedarf** und **Prognostizierter Bedarf**.

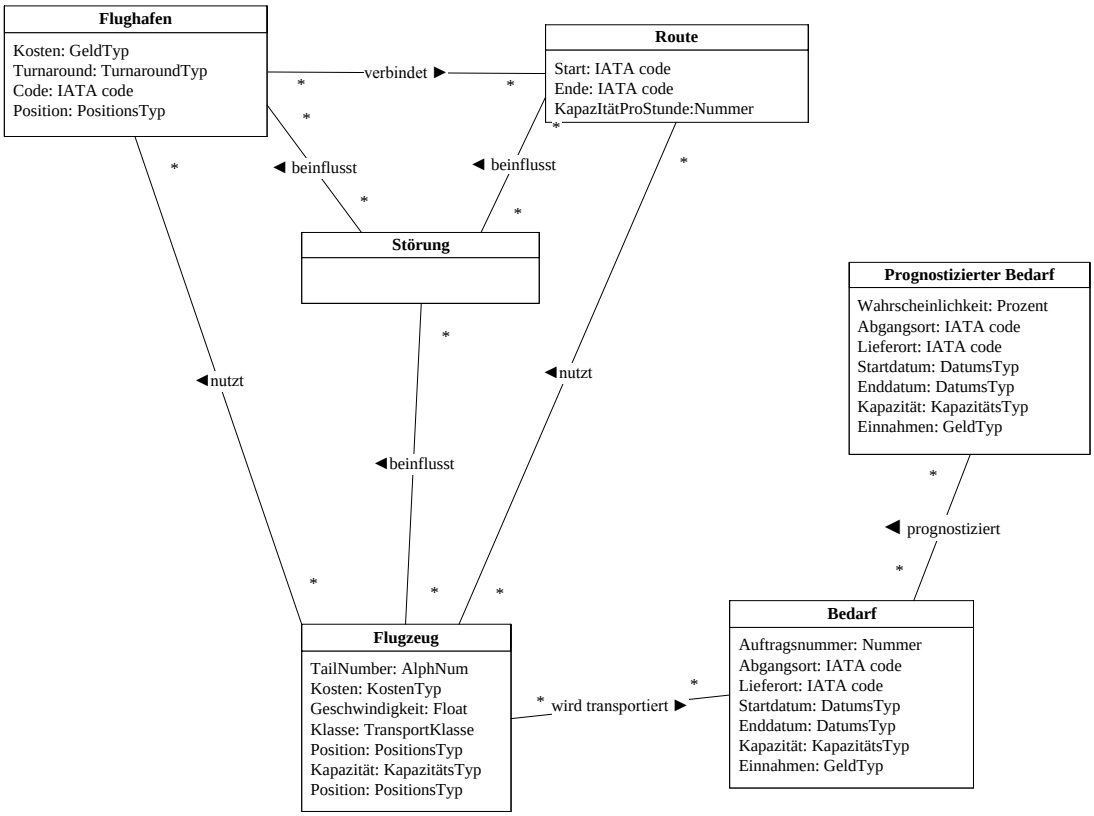

Abbildung 4.1.: Fachliches Datenmodell, Übersicht

Das Datenmodell zeigt, dass ein **Flugzeug**, das durch seine *Leitwerknummer (TailNumber)* identifiziert werden kann, Flughäfen und Routen nutzt. Hier handelt es sich um eine Dreiecksbeziehung, in der die durch ihren *IATA Code* identifizierten **Flughäfen** durch **Routen** verbunden werden, die jeweils zwischen zwei Flughäfen liegen.

Dieser Kreislauf, in dem Flugzeuge zwischen Flughäfen verkehren, kann von **Störungen** betroffen sein, die entweder auf eine bestimmte Entität (Flugzeug, Flughafen, Route) beschränkt sind oder in einem bestimmten geographischen Gebiet alle Entitäten beeinflussen, siehe hierzu auch Abbildung 4.2, in dem getrennt ein fachliches Datenmodell für Störungen gezeigt wird. Störungen wirken sich durch eine Erhöhung (prozentual oder absolut) der spezifischen Reisezeit der betroffenen Entität(en) aus. Für einen Flughafen ist die spezifische Reisezeit die Abfertigungszeit, für Flugzeuge und Routen die Flugdauer von Flugzeugen.

Bedarfe sind in diesem fachlichen Datenmodell solche Transporte, die in einem Zeitraum von einem Startflughafen zu einem Zielflughafen transportiert werden sollen. Für diesen Dienst werden Flugzeuge genutzt, die diesen Transport durchführen. Analog zum Bedarf zeigt das

Modell den **Prognostizierten Bedarf**, der in der Planungsphase der Simulation einen mit einer gewissen Wahrscheinlichkeit eintretenden Bedarf abbildet.

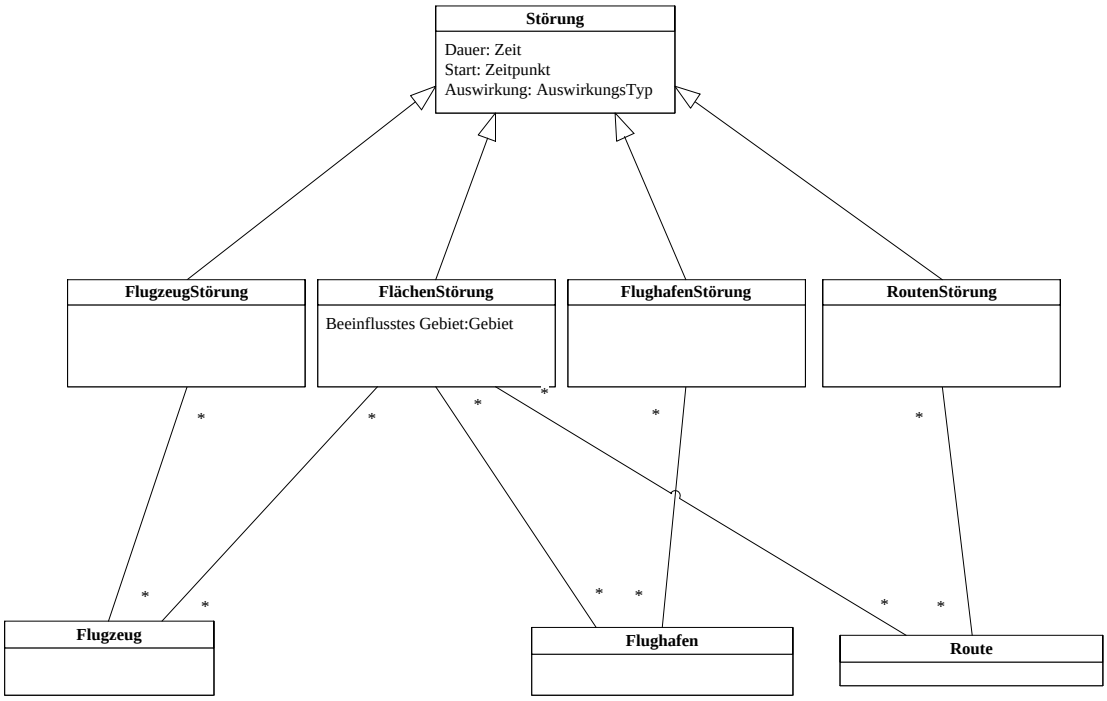

Abbildung 4.2.: Fachliches Datenmodell, Störungen

Abschließend seien noch die entscheidenden, im Modell verwendeten fachlichen Datentypen betrachtet. Eine komplette Übersicht aller Datentypen ist in Anhang C vorhanden. Die fachlich interessanten Datentypen, auf die hier gesondert eingegangen werden soll, sind **Transportklasse**, **Positionstyp**, **KapazitätsTyp**, **TurnaroundTyp** und **IATA Code**.

Die **Transportklasse** eines Flugzeuges dient der Modellierung von Beschränkungen und Eigenschaften im Zusammenspiel von Flughäfen und Flugzeugen. Auf Flughäfen können nur Flugzeuge mit bestimmten Transportklassen abgefertigt werden und von der Transportklasse hängt auch die Abfertigungszeit ab. Die Transportklasse modelliert die in der Realität in der Regel vorhandenen Wechselwirkungen:

- `größeres Flugzeug = längere Abfertigungszeit`

- `schwereres Flugzeug = längere Landebahn`

In Verbindung mit der Transportklasse stellt der **TurnaroundTyp** im Flughafen die Zeit dar, die auf einem speziellen Flughafen für die Abfertigung einer bestimmten Flugzeugklasse benötigt wird.

Der hier genutzte **Positionstyp** modelliert nur Breiten- und Längengrade, er stellt insofern eine Abstraktion der Realität dar, da im realen Flugbetrieb offensichtlich auch Höhen benötigt werden.

In der Realität ist die Beladung von Flugzeugen ein Vorgang, der von Volumen, Gewicht und Verteilung des Gewichtes im Flugzeug abhängt. Der **KapazitätsTyp** stellt nur Volumen und Gewicht dar, die Simulation der Gewichtsverteilung in einem Flugzeug mit dem Finden einer optimalen Verteilung würde aufgrund ihrer Komplexität vom Ziel dieser Arbeit ablenken.

Durch **IATA Codes** werden in der Luftfahrtbranche unter anderem Flugzeugtypen, Flughäfen und Airlines identifiziert. Die hier genutzte Modellierung dieses Datentyps identifiziert nur Flughäfen und ist ein aus drei Zeichen bestehender alphanumerischer String.

4.4. Komponenten

Auf dem in Kapitel 4.3 entwickelten Datenmodell wurde ein Komponentenschnitt vorgenommen, um die Entitäten vor der Entwicklung in fachlich zusammenhängende Komponenten aufzuteilen. Eine Analyse der Entitäten zeigte, dass sich aufgrund der engen fachlichen Verknüpfung der in Abbildung 4.3 zur Übersicht und in Tabelle 4.2 vollständig gezeigte Schnitt in **Operation**, **Störung** und **Bedarf** anbietet. Hier werden die am nicht gestörten Betrieb beteiligten Entitäten Flugzeug, Flughafen, Route in der Komponente **Operation** zusammengefasst. Alle Entitäten, die eine Störung und damit einen abnormalen Zustand modellieren (siehe fachliches Datenmodell, insbesondere Abbildung 4.2), sind in der Komponente **Störung** zusammengefasst. Die Entitäten Bedarf und Prognostizierter Bedarf sind in der Komponente **Bedarf** enthalten.

Komponente	Entitäten
Operation (Blau)	Flugzeug Flughafen Route
Störung (Braun)	Störung
Bedarf (Grün)	Bedarf Prognostizierter Bedarf

Tabelle 4.2.: Komponentenschnitt

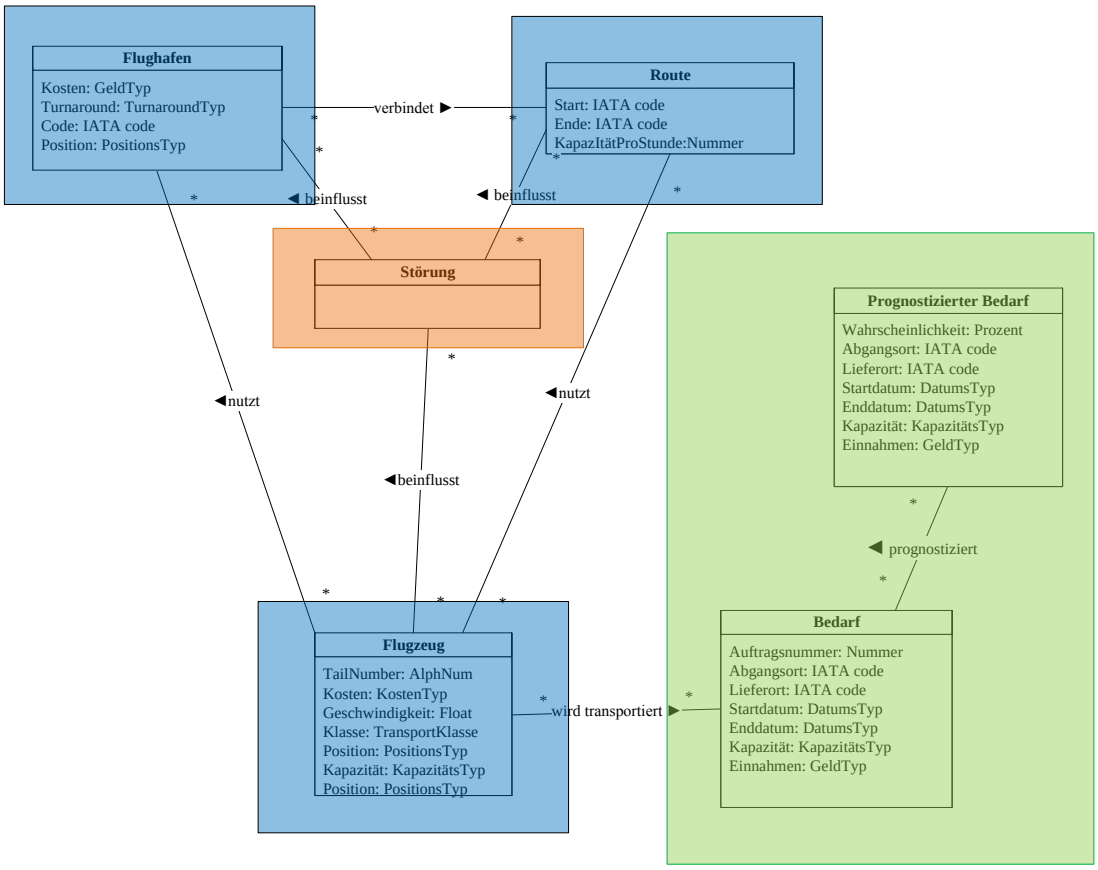

Abbildung 4.3.: Komponentenschnitt über fachlichem Datenmodell

4.5. Fachliche Architektur

Um die fachliche Komplexität eines Systems vor dem Entwurf überblicken zu können, hat es sich bewährt, diese in einer fachlichen Architektur abzubilden, um von dieser aus am Entwurf weiterzuarbeiten. Hier soll ein Überblick über die fachliche Architektur des Simulationsteils gegeben werden, um dies zu ermöglichen. Für die in Anhang E gezeigte Architektur des Planungsteils gilt überwiegend Analoges.

In dem in Abbildung 4.4 gezeigten Diagramm ist neben der Trennung der im fachlichen Datenmodell beschriebenen Komponenten Störung, Bedarf und Operation der Kommunikationsweg dieser Komponenten untereinander visualisiert, der immer über den Szenariomanager führt. Dieser Weg wurde hier gewählt, um die Kopplung der fachlichen Komponenten untereinander möglichst lose gestalten zu können.

Das Laden der fachlichen Daten wird aufgrund der zu erwartenden Ähnlichkeit im technischen Ablauf für alle drei betroffenen Komponenten über den Stammdatenadapter durchgeführt. Der Adapter bietet jeder Komponente für das Laden eine eigene Schnittstelle, um bei eventuell notwendigen Anpassungen nur jeweils eine Schnittstelle anpassen zu müssen und die Klarheit der Schnittstellen zu erhalten (vgl. Fowler, 2006).

Auf die Simulation kann von außen über die Simulationsfassade zugegriffen werden, die Schnittstellen bietet, um Algorithmen in die Simulation einzubringen sowie diese zu steuern. Die von außen eingebrachten oder durch das Framework mitgelieferten Algorithmen werden durch den Szenariomanager immer über die Algorithmenadapter geladen, um vor der Simulation zu verbergen, wie der Zugriff konkret abläuft.

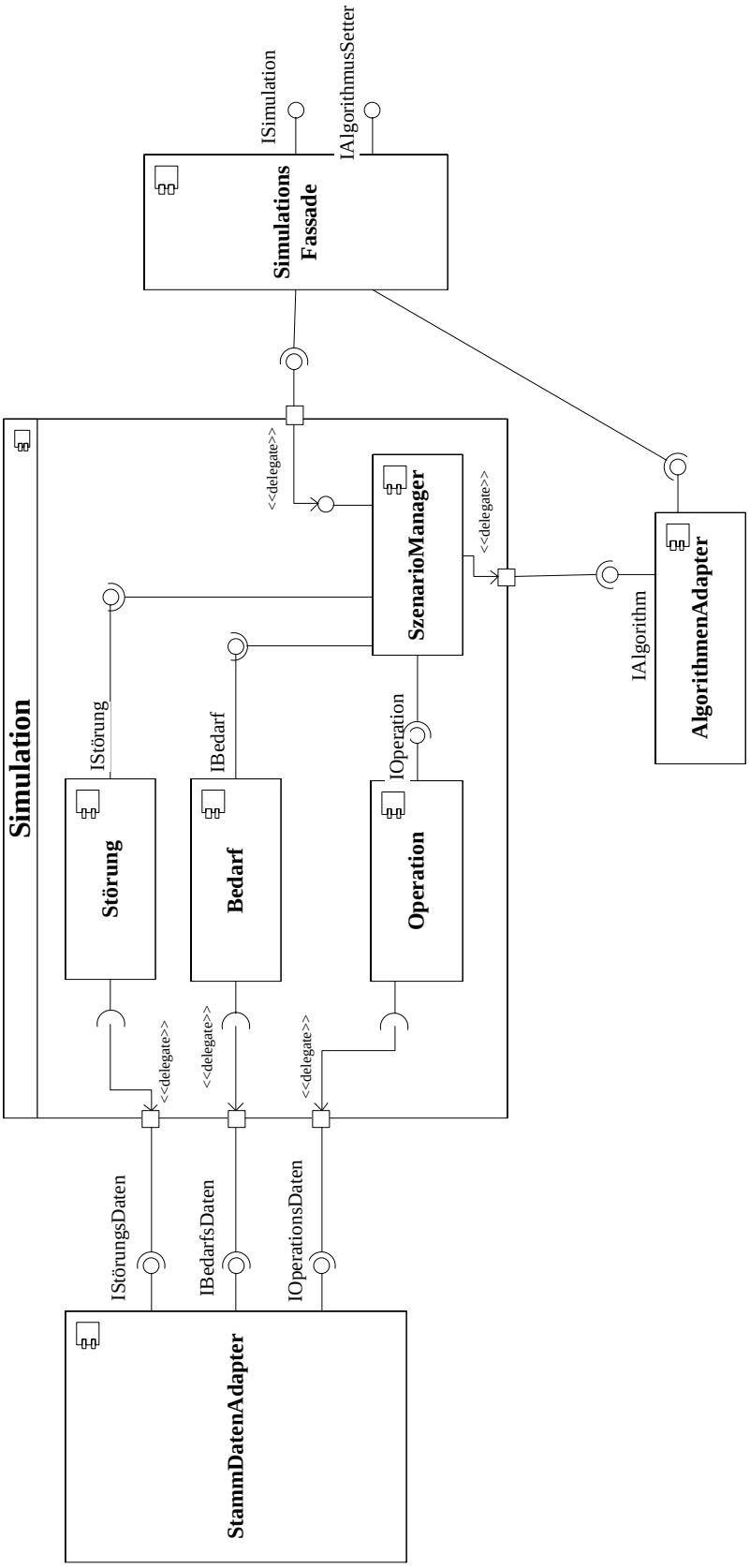

Abbildung 4.4.: Fachliche Architektur, Simulation

4.6. Graphische Benutzeroberfläche

In diesem Abschnitt wird die Graphische Benutzeroberfläche (GUI) spezifiziert. Nach einer Erläuterung der mit der GUI verfolgten Ziele werden die Informationen herausgearbeitet, die zur Erreichung der Ziele benötigt werden und ein Mock Up der GUI gezeigt.

4.6.1. Ziel

Die GUI soll dem Anwender wesentliche Parameter der simulierten Welt während der Simulation darstellen.

Der Begriff des Anwenders umfasst hier aus den in 3.1 definierten Stakeholdern Routenplaner, Disponenten und Algorithmen-Entwickler. Dem Routenplaner oder Disponenten, der die Simulation nutzt, um verschiedene Algorithmen bzw. Handlungsalternativen zu vergleichen, soll über die graphische Darstellung ein **intuitiveres Verständnis** davon vermittelt werden, wie eine Simulation abgelaufen ist, als dies nur durch Kennzahlen und statistische Ergebnisse möglich wäre. Der Algorithmen-Entwickler soll bei der **Fehlersuche** unterstützt werden, die graphische Darstellung des Systemzustandes ermöglicht es ihm, den erwarteten Zustand der Simulation mit dem tatsächlichen visuell zu vergleichen. Abhängig vom konkreten Problem kann hierdurch eine deutliche Beschleunigung der Fehlersuche erreicht werden (vgl. Wilcox u. a., 1997).

4.6.2. Informationen

Eine Analyse der in der Simulation vorhandenen Daten hat aufgezeigt, dass es grundsätzlich sehr **allgemeine Informationen** wie zum Beispiel Positionen von Flughäfen oder Flugzeugen gibt und zum anderen **detaillierte Informationen** zu einzelnen Entitäten wie zum Beispiel die komplette Routenplanung für ein Flugzeug. Die GUI unterteilt unter Ausnutzung dieser Tatsache die möglichen Informationen in solche, die immer angezeigt werden und solche, die nur auf expliziten Wunsch des Nutzers angezeigt werden.

Die **ständig sichtbaren** Informationen sind die Positionen von Flughäfen, Flugzeugen, Routen, die verbleibende Zeit in der Simulation und das aktuelle wirtschaftliche Ergebnis. Störungen werden dem Anwender durch Verfärben der betroffenen Entitäten dargestellt, die Abbildungen 4.5 und 4.6 zeigen dies beispielhaft für ein Flugzeug.

Durch Fokussierung (bspw. Mausklick) kann der Anwender **Detailinformationen** zu der im Fokus befindlichen Entität (Flugzeug, Flughafen oder Route) abrufen. Die jeweils angezeigten Informationen sind für **Flugzeuge** die derzeitige Ladung, die geplante Route und die

Abbildung 4.5.: Symbol Flugzeug, normal

Abbildung 4.6.: Symbol Flugzeug, von Störung betroffen

Auswirkungen von Störungen. An **Routen** werden die derzeitige Reisezeit und die Auslastung sowie eventuell aktive Störungen gezeigt. **Flughäfen** zeigen in ihren Detailinformationen den derzeitig an ihnen bestehenden Transportbedarf sowie die Abfertigungszeiten für alle Flugzeugklassen.

4.6.3. Mock

Abbildung 4.7 zeigt die wesentlichen Elemente der GUI im Übersichtsmodus ohne Details. Es sind die fünf in 4.6.2 genannten Elemente zu dargestellt.

1. Verbleibende Zeit in der Simulation in Minuten
2. Derzeitiges wirtschaftliches Ergebnis
3. Flughäfen
4. Flugzeuge
5. Routen

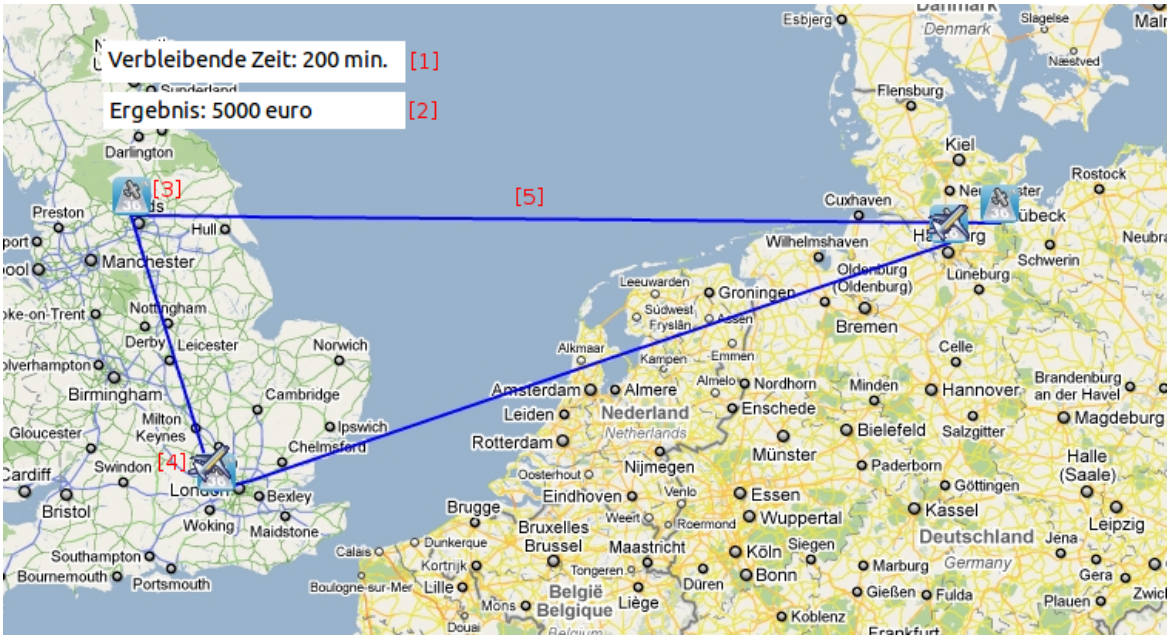

Abbildung 4.7.: GUI Mock (Copyright Maps Google 2011)

5. Entwurf und Implementierung

Im diesem Kapitel werden der Entwurf des Frameworks und die Ergebnisse der prototypischen Implementierung beschrieben. Hierzu wird zuerst das gewählte Vorgehensmodell begründet, um anschließend auf Grundlagen der Simulation einzugehen. Nach einer Erläuterung der zur Implementierung eingesetzten Software werden Aspekte des technischen Frameworkentwurfs detailliert beschrieben. Abschließend wird kurz auf die während der Implementierung des Prototypen aufgetretenen Probleme eingegangen.

5.1. Vorgehensmodell

Die Entwicklung korrekter und robuster Software ist, wie sich seit der Softwarekrise Mitte der sechziger Jahre immer deutlicher zeigt, keine triviale Aufgabe. Um die Komplexität des Softwareentwicklungsprozesses überschaubar und letztlich beherrschbar zu machen, wurden, seitdem das Problem erkannt wurde, verschiedene Vorgehens- und Prozessmodelle wickelt, um die Softwareentwicklung zu beschreiben und erfolgreiche Vorgehensweisen zu verbreiten. Hier soll ein kurzer Überblick über einige Modelle gegeben werden, um dann das in dieser Arbeit gewählte Vorgehensmodell zu beschreiben.

Beispielhaft sollen hier, geordnet nach Einführungsdatum, das **Wasserfallmodell**, das **Phasenmodell** und die **inkrementelle Entwicklung** vorgestellt werden.

Das **Wasserfallmodell** als ältestes der drei gezeigten Modelle aus den siebziger Jahren unterteilt den Entwicklungsprozess strikt in die Phasen Initialisierung, Analyse, Entwurf, Realisierung, Einführung und Nutzung. Diese Aktivitäten werden sequenziell und regelmäßig mit Rücksprung in die vorhergehende Aktivität ausgeführt. Die Vorteile des Wasserfallmodells beinhalten die klare Definition der Aktivitäten mit den sie abschließenden Dokumenten, eine klare Abschätzung des Entwicklungsstands zu jedem Zeitpunkt und gute Schätzbarkeit der Kosten (Royce, 1987). Der große Nachteil des Wasserfallmodells ist, dass durch den sequenziellen Ablauf der Aktivitäten und deren strikte Trennung vorgeschrieben verlangt wird, dass die in Analyse und Entwurf erzeugten Dokumente vollständig und stabil sind. In der Realität hat sich gezeigt, dass es für komplexe Softwaresysteme regelmäßig nicht möglich ist, die Anforderungen im Voraus komplett und korrekt zu erfassen. Ebenfalls sind Anforderungen aufgrund von Erfahrungen während der Entwicklung häufig Änderungen unterworfen.

Diese Änderungen werden bei Anwendung des Wasserfallmodells zu spät erkannt, da das Modell Eingaben von Kundenseite nur zu einigen wenigen Punkten vorsieht (Royce, 1987). Die von der zunehmenden Komplexität von Software geprägte Entwicklung von Modellen nach dem Wasserfallmodell zeigt eine Tendenz zur Aufweichung der strikten Trennung zwischen Aktivitäten und zu iterativen Methoden.

Das **Phasenmodell** unterteilt den Softwareentwicklungsprozess nicht mehr in Aktivitäten, sondern in durch Meilensteine getrennte Phasen, in denen jeweils auch mehr als eine der unter Wasserfallmodell genannten Aktivitäten durchgeführt werden kann. Hier werden die Nachteile des Wasserfallmodels vermieden, da durch die nicht mehr strikt getrennten Aktivitäten Erkenntnisse aus späteren Aktivitäten noch beachtet werden können. Nachteilig wirkt sich hier ein höherer Aufwand in der Projektverwaltung aus (Sarstedt, 2009).

In der **inkrementellen Softwareentwicklung** wird dieser Ansatz zu einem wiederholten Durchlaufen der Aktivitäten erweitert. Zuerst wird eine Kernfunktionalität entwickelt, um diese dann in jeder Iteration zu erweitern. Dieses Vorgehen hat die Vorteile, für jede Stufe der Entwicklung nur einen geringen Zeitaufwand zu haben und durch die frühe Fertigstellung von lauffähigen Teilen Feedback von Kundenseite zu ermöglichen. Nachteilig wirkt sich auch hier der relativ hohe Verwaltungsaufwand aus (Ludewig und Lichter, 2006). Es sind große Überschneidungen mit den Prinzipien agiler Softwareentwicklung zu erkennen (Beck, 2001).

Für diese Arbeit wurde ein **inkrementelles Vorgehen gewählt**. Dieses Vorgehen ist auch der in der Literatur empfohlene Ansatz zur Entwicklung eines Frameworks bzw. anderer komplexer Software (vgl. P. Brooks, 2010, S. 344), (vgl. Fayad u. a., 1999, S. 21-22). Entwurf und Entwicklung finden in kurzen Abschnitten, hier ca. 3 Tage statt, an deren Ende jeweils ein ausführbarer Prototyp stehen soll. Diese Art des Vorgehens ermöglicht, anhand der während der Entwicklung gewonnenen Erkenntnisse die Anforderungen und den Entwurf anzupassen bzw. zu erweitern. Dieses Feedback aus dem Entwicklungsprozess dient auch dazu, möglichst früh fundamentale Fehlannahmen im Prototypen zu erkennen.

Entwurf und Entwicklung finden hier also durch mehrmalige Wiederholung der folgenden vier Aktivitäten statt:

1. Iterationsumfang festlegen
2. Prototyp entwickeln
3. Testen/Evaluieren
4. Spezifikation erweitern/korrigieren

5.2. Simulation

Bei der Simulation handelt es sich um eine in der Forschung und Anwendung weit verbreitete Strategie zur Problemlösung (Banks, 1998). Das in dieser Arbeit zu entwerfende Framework soll die von Störungen beeinflussten Aktivitäten in einem Luftfrachtnetzwerk simulieren, hierbei handelt es sich um eine Simulation diskreter Ereignisse (**Discrete Event Simulation, DES**). Im Folgenden sollen **Herangehensweisen**, die sich für die Lösung von Simulationsproblemen als Standard etabliert haben, und ihre konkrete Ausprägung in dieser Arbeit gezeigt werden. Neben diesen Standards werden auch Limitierungen des gewählten Ansatzes und weitere Entwicklungsmöglichkeiten diskutiert.

5.2.1. Parallel/Sequenziell

Eine erste grundlegende Designentscheidung beim Entwurf einer Simulationsumgebung ist die, ob die Simulation parallel oder sequenziell ausgeführt werden soll. Diese Entscheidung beeinflusst wesentlich die Komplexität der Simulationsentwicklung.

Parallele Simulationen (PDES) können regelmäßig Probleme schneller bzw. Probleme ab einer gewissen Größe überhaupt erst lösen. Dies, da viele Probleme, die durch Simulation gelöst werden können, eine signifikante Menge von parallelisierbaren Aufgaben enthalten und gleichzeitig so groß sind, dass eine sequenzielle Simulation in akzeptablen Laufzeiten nicht realisierbar ist (vgl. Fujimoto, 1990, S. 31). Gleichzeitig weist das Design einer PDES Komplexitäten auf, die es deutlich schwieriger machen als eine sequenzielle Simulation. Es ist kein triviales Problem, in einer Simulation Verletzungen der Kausalitäts zu verhindern und diese gleichzeitig zu parallelisieren. Bagrodia (1996) listet unter anderem folgende Aspekte auf, die beachtet werden müssen: Verteilen der Prozesse auf Recheneinheiten, Verwaltung von Rollback checkpoints und Erkennen von Kausalitätsverletzungen.

Im Rahmen dieser Arbeit wurde eine **sequenzielle Simulation** gewählt. Diese Vorgehensweise beschränkt die maximale Simulationsgröße auf die maximale auf einer Recheneinheit in vertretbarer Zeit lauffähige Simulation. Hier wichtiger sind jedoch die Vorteile, die durch das Ausklammern von Parallelität realisiert werden können. Eine sequenzielle Simulation ist deutlich leichter zu entwerfen und zu implementieren, sowie für spätere Erweiterungen auch leichter zu verstehen. Eine spätere Parallelisierung der Simulation ist durch die sehr lockere Kopplung der fachlichen Algorithmen an die Simulation (siehe Kapitel 5.5) für den Anwender transparent und wäre so durch eine reine Anpassung des Frameworks möglich.

5.2.2. Struktur des Simulationsablaufes

Durch die Struktur des Simulationsablaufes wird definiert, wie die Simulation vorwärts läuft, d. h. wann in der Simulation neue Ereignisse generiert werden. Banks (1998) listet unter anderem drei Möglichkeiten auf, diesen Ablauf zu strukturieren: **Process-Interaction**, **Event-Scheduling** und **Activity-Scanning**.

Process-Interaction und **Event-Scheduling** haben als Grundkonzept die Orientierung an den Objekten der Simulation. Die Simulation erzeugt immer zum Beginn und Ende der Aktivität eines Objekts neue Ereignisse, wenn diese neuen Ereignisse durch die Zustandsänderung der Simulation möglich geworden sind. Process-Interaction verfolgt hierbei ein Objekt durch die Simulation, bis dieses auf ein Ereignis warten muss, während Event-Scheduling nach jedem Ereignis alle anderen Objekte auf neue Ereignisse prüft.

Activity-Scanning setzt im Gegensatz hierzu nicht direkt an den Objekten der Simulation an, sondern an der Simulationszeit. Die Zeit wird in diskreten Schritten fortgeführt und nach jedem Schritt geprüft, ob zu diesem Zeitpunkt ein Ereignis eintritt.

In dieser Arbeit wurde **Activity-Scanning** zur Steuerung des Simulationsablaufes gewählt. Da die kontinuierliche Darstellung der Simulation in der GUI erfordert, dass gezeigte Flugzeugpositionen in kleinen Schritten verändert werden, um abrupte, den Nutzer überraschende Veränderungen zu vermeiden. Gegenüber den anderen Methoden tritt hier ein höherer Simulationsaufwand auf, da Zwischenzustände simuliert werden, die für das Endergebnis nicht relevant sind. Der Nutzen, den der Anwender durch die GUI hat, wurde höher bewertet als die Geschwindigkeitssteigerung.

In einer späteren **Erweiterung** der Simulation kann dieses Problem durch eine Veränderung der Eventgenerierung durch den Anwender konfigurierbar gemacht werden, ohne dass dieser seine fachlichen Algorithmen verändern muss. Es würde sich nur die graphische Nutzeroberfläche verändern.

5.2.3. Komponenten der Simulation

Fujimoto (1990) zählt folgende drei Standardkomponenten von DES Systemen auf:

1. State Variables
2. Event List
3. Global Clock

In Abbildung 5.1 werden diese drei Komponenten im hier entworfenen Framework gezeigt und ihre Interaktion am Beispiel einer Flugzeugbewegung verdeutlicht.

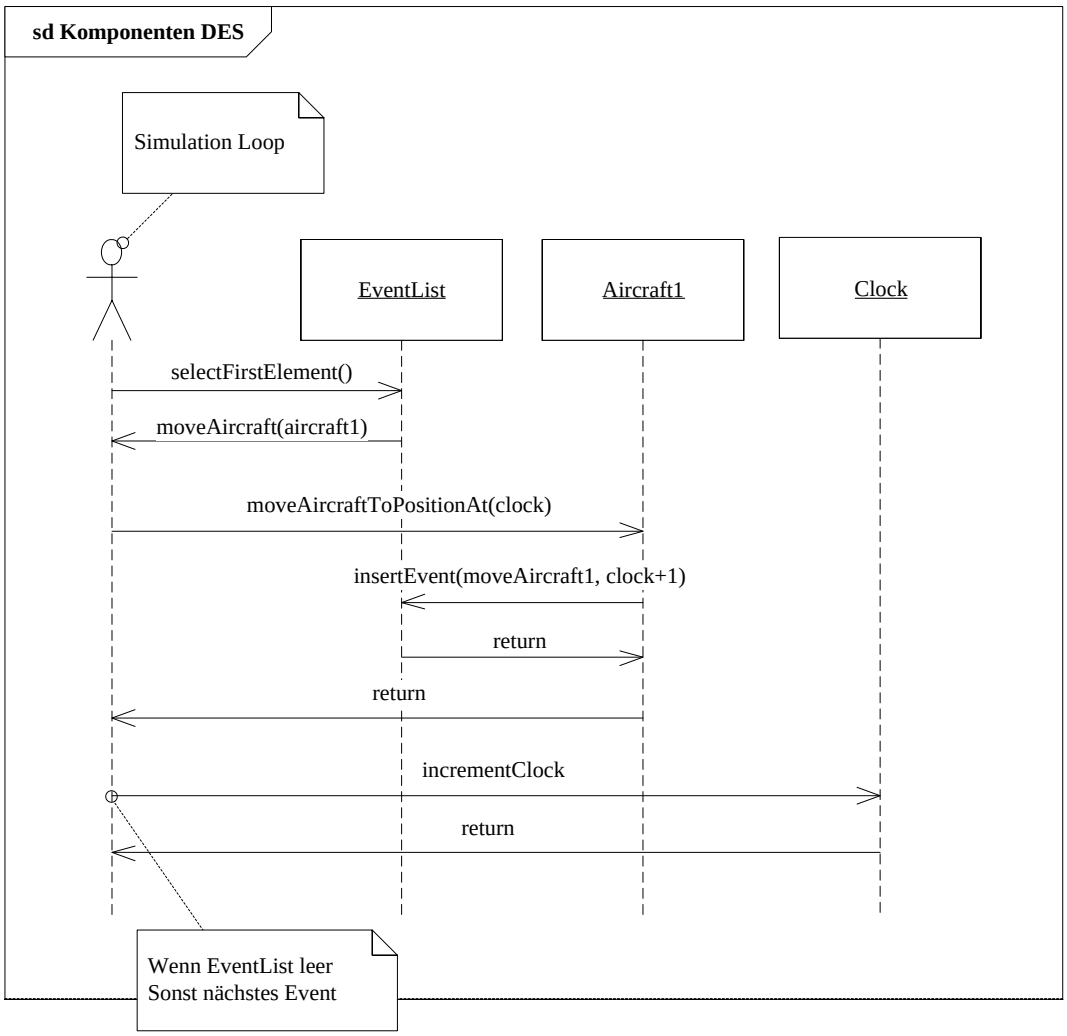

Abbildung 5.1.: Kommunikation zwischen den Komponenten des DES

5.3. Testkonzept

Aufgabe von Tests ist es, Fehlerwirkungen gezielt und systematisch aufzudecken, unter einem Test versteht die Literatur daher das Durchführen einer Operation mit einem System,

um danach den erwarteten Systemzustand mit dem tatsächlichen zu vergleichen. Tests lassen sich unter anderem in solche, die spezifizierte Funktionen testen (funktionale Tests) und solche, die nicht funktionale Anforderungen an die Qualität der Software wie zum Beispiel Geschwindigkeit und Benutzbarkeit testen (nicht funktionale Tests) teilen (Spillner und Linz, 2005).

Das hier vorliegende Testkonzept sieht nur funktionale Tests vor. Da der Prototyp nur die Machbarkeit zeigen soll, scheint es nicht zielführend, Aspekte der Softwarequalität zu diesem Zeitpunkt schon zu testen. Bei einer nicht prototypischen Implementierung sollten diese in jedem Fall um nicht funktionale Tests erweitert werden. Ein geeigneter Ansatzpunkt für eine erste Erweiterung wäre die Geschwindigkeit der API.

Ziele der Tests sind in dieser Arbeit das Aufdecken **unbekannter Abhängigkeiten** bei Änderungen und das Entdecken von **Regressionen** bei schon behobenen Fehlern. Um diese Ziele zu erreichen, wird eine Kombination von **Komponententests**, um einzelne Klassen und Methoden zu testen und **Integrationstests**, um das Zusammenspiel von Komponenten zu testen, verwendet. Die üblicherweise später in der Entwicklung stattfindenden Systemtests und Abnahmetests werden hier aufgrund des Prototypstatus nicht betrachtet (Spillner und Linz, 2005). Entsprechend dem in der Literatur vorgeschlagenen Vorgehen bei iterativer Entwicklung wurden die Integrationstests mindestens nach jeder Iteration durchgeführt. Währenddessen wurde das Vorgehen in Richtung kontinuierliche Integration verschoben. Durch das häufigere Testen, in der Regel nach jedem Einchecken in das Versionskontrollsystem, konnten Fehler deutlich schneller aufgedeckt werden.

Konkret wurden parallel zur Entwicklung Unittests als Ausprägung von **Komponententests** entwickelt, um einzelne Methoden oder Klassen zu testen. Parallel zur Entwicklung der API wurden **Integrationstests** für die einzelnen Funktionen der API entwickelt, die diese testen. Welche Methoden einzelner Klassen oder Funktionen bzw. Fehlerfälle der API durch diese systematischen Tests abgedeckt werden, wurde im Einzelfall danach entschieden, wo der Erfahrung nach Probleme zu vermuten sind (vgl. Spillner und Linz, 2005, S. 159, 160). Häufig sind dies z. B. Schnittstellen zu fremden Bibliotheken. Neben dem systematischen Testaufbau wurden Fehler, die in der Anwendung während der Entwicklung aufgetreten sind, durch Tests abgedeckt, um spätere **Regressionen** zu vermeiden.

5.4. Software

In diesem Kapitel werden die eingesetzte Software und die verwendeten Standards aufgelistet. Für die entscheidenden Bibliotheken und Standards werden die Gründe, die zum Einsatz einer speziellen Bibliothek bzw. eines Standards geführt haben, erläutert. Die gesamte Auswahl von Fremdsoftware ist geprägt von eher kleinen Bibliotheken im Gegensatz zu großen

Frameworklösungen. Dieser Ansatz wurde hier gewählt, um die in Froehlich u. a. (2000) und Flores und Aguiar (2008) beschriebenen Probleme bei der Einarbeitung in ein unbekanntes Framework zu vermeiden bzw. den möglichen Fehler auf die kleineren Bibliotheken einzugrenzen.

Softwarekomponente	**Einsatzbereich**
Apache Mina (Version 1.6)	• Abstraktionsschicht direkt auf dem Netzwerk
Apache Commons Configuration (Version 2.0.4)	• Verwaltung der für den Anwender zugänglichen Konfigurationsoptionen
Apache Commons Lang (Version 2.5)	• Realisierung von Depency Injection über Konstruktoren
Google Geocoder Java API (Version 1.5.0)	• Verwendung für Positionsdaten von Entitäten • Unterstützung bei der Entfernungsberechnung
Google Maps	• Graphische Benutzeroberfläche
jQuery (Version 1.6.1)	• Graphische Benutzeroberfläche

Google Guava (Version Release 09)	• Direkter Ersatz für Collections mit funktionalen Erweiterungen • Realisierung von zur Laufzeit typsicheren Entitätslisten • Realisierung von Prädikaten und Funktionen über Entitätslisten
google-gson (Version 1.7.1)	• Bibliothek zur Verarbeitung und Erstellung von json-Zeichenketten • Eingesetzt zur Serialisierung und Deserialisierung von Entitäten
JGraphT (Version 0.7.3)	• Support für graphentheoretische Objekte und Algorithmen • Realisiert Unterstützung für die Verarbeitung von Netzwerken als Graphen • Visualisierung von Graphen
Apache Mina (Version 2.0.4)	• Netzwerk Framework
jsonRPC2 (Version 1.19)	• Implementierung des json-RPC Standards
Log4J (Version 1.2.16)	• Logging

jUnit (Version 3.8.1)	• Tests

Tabelle 5.1.: Eingesetzte Software von Drittanbietern

5.4.1. json

Die *Serialisierung* von Entitäten ist in diesem Framework unter anderem notwendig, um sie zu persistieren bzw. über sie zum Algorithmus zu übertragen. Hier wurden verschiedene Optionen analysiert, um eine Serialisierungslösung zu finden, die leichtgewichtig und gut dokumentiert ist sowie eine breite Bibliotheksunterstützung in verschiedenen Sprachen hat. Analysiert wurden neben reinen Serialisierungslösungen auch Bibliotheken, die komplette *Middleware* Systeme implementieren. In die Betrachtung sind folgende Lösungen eingeflossen: xml, *Protocol Buffers*, *Apache Thrift*, Corba, Java Serialisierung und *json*. In Tabelle 5.2 wurden diese Alternativen nach den Kriterien Komplexität `[0]`, Dokumentation `[1]` und Bibliotheksunterstützung `[2]` bewertet (−, `0`, +).

Name	`[0]`	`[1]`	`[2]`
xml	-	+	+
Protocol Buffers	0	+	0
Apache Thrift	0	0	0
Corba	-	-	-
Java Serialisierung	+	0	-
json	+	+	+

Tabelle 5.2.: Bewertung Serialisierungslösungen

Für diese Arbeit wurde nach der Analyse **json** gewählt. Da der Fokus auf die oben genannten Kriterien gelegt wurde, werden die Nachteile von json hier nicht als entscheidend betrachtet bzw. durch die Vorteile aufgewogen. Der primäre **Nachteil** von json ist, dass nur einfache Datentypen wie z. B. `string` und `boolena` sowie simple Strukturen wie `array` und `hash` serialisiert werden können (vgl. Crockford, 2006). Diesen Nachteilen stehen die **Vorteile** wie die durch die bewusste Beschränkung auf diese simplen Datentypen/-strukturen gegebene Übersichtlichkeit und Vielzahl der verfügbaren Implementierungen gegenüber. Als weiterer Vorteil hat sich in der Praxis herausgestellt, dass json als einfache Serialisierung im Klartext für Menschen leichter zu lesen ist als komplexere Klartextformate wie z. B. xml bzw. im Gegensatz zu Binärformaten wie Protocol Buffers überhaupt direkt lesbar ist.

5.4.2. RPC

Um die Kommunikation zwischen zwei durch ein Netzwerk getrennten Komponenten zu strukturieren, ist es notwendig, hierfür in der Middlewareschicht ein Protokoll zu definieren (vgl. Tanenbaum und van Steen, 2007, S. 147,148).

Durch die hier bereits erfolgte Entscheidung, zur Serialisierung json zu nutzen (siehe 5.4.1), konnte die Suche nach einem geeigneten Protokoll auf die beiden Optionen *json-RPC* und *HTTP-REST* eingegrenzt werden. Die Entscheidung für json-RPC im Rahmen dieser Arbeit wurde letztendlich getroffen, um eine Abhängigkeit von zusätzlichen Komponenten wie einem Webserver zu vermeiden.

Beispielhaft sind in Listing 5.1 die Elemente eines json-RPCs dargestellt.

Listing 5.1: Beispielaufruf GetEntityList

```
{ "id":1,
  "method":"GetEntityList",
  "params":{
       "type":"aircrafts"
     },
  "jsonrpc":"2.0"}
```

Die einzelnen Felder **id**, **method**, **params**, **jsonrpc** sind durch den Standard definiert. Die **id** ist ein Identifikator, an dem der Client feststellen kann, welche Antwort zu welcher Anfrage gehört, der Server spiegelt immer genau die id zurück, die er vom Client mit einer Anfrage bekommen hat. Das **method** Feld gibt die Methode auf dem Server an, die der Client aufrufen will. **jsonrpc** gibt die Version des Protokolls an, die der Client verwendet und **params** die Parameter des Aufrufes. Die Parameter sind hier als json-Objekt codiert, dabei handelt es sich aber um eine Designentscheidung in dieser Arbeit. Prinzipiell könnten die einen beliebigen String enthalten (JSON RPC Project, a,b).

5.4.3. jGraph

Wie in den Grundlagen (siehe 2.1.3.2) gezeigt, werden Probleme der Routenplanung im Luftfrachtbereich regelmäßig mit den Mitteln der Graphentheorie gelöst. Um die Entwicklung neuer Algorithmen zu fördern, bindet die Algorithmen-Komponente des Frameworks die Bibliotheken *JGraphT* und *JGraph* ein. jGraph wird genutzt, um dem Anwender des Frameworks das Netzwerk zwischen Flughäfen als Graph übergeben zu können sowie ihm Algorithmen aus der Graphentheorie anzubieten. In Verbindung mit JGraphT ist eine Visualisierung der Graphen, wie in Abbildung 5.2 beispielhaft dargestellt, zur Veranschaulichung und Fehlersuche möglich.

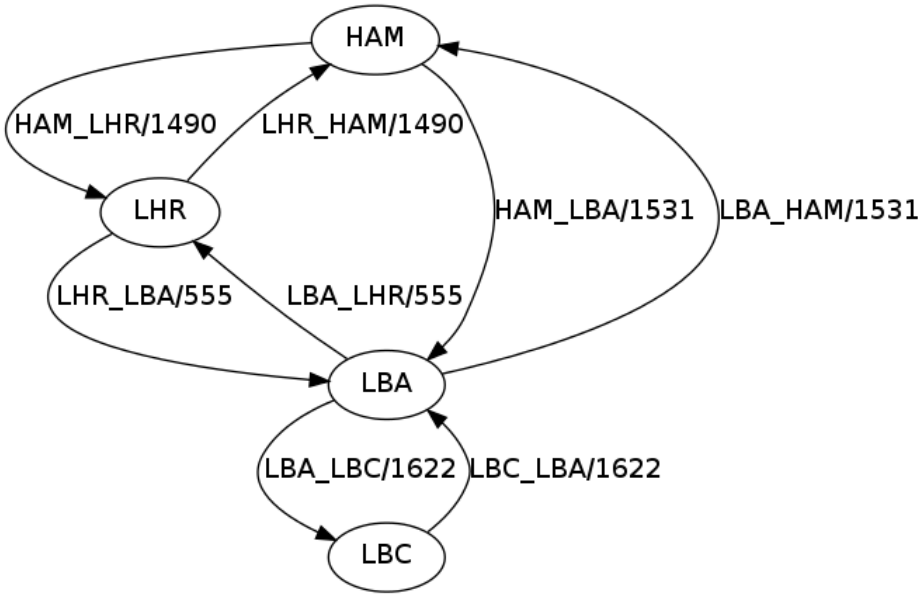

Abbildung 5.2.: Visualisierung Routennetz als Graph

5.5. Framework

Ein unbekanntes Framework zu verstehen, stellt den Anwender regelmäßig vor Probleme. Strukturierte Prozesse, um ein Framework zu evaluieren bzw. den Umgang mit ihm zu erlernen, setzen häufig an den Hot Spots (siehe 2.2) des Frameworks an (Flores und Aguiar, 2008). Für den Entwickler, der ein Framework erweitern will, ist neben diesem Verständnis der Erweiterungspunkte auch ein Überblick über die Architektur des Frameworks wichtig (Flores und Aguiar, 2008).

Hier wird, um diesen Anforderungen gerecht zu werden, zuerst ein Einblick in die technische Architektur des Frameworks gegeben. Anschließend werden die definierten Erweiterungspunkte und die API des Frameworks beschrieben.

5.5.1. Architektur

Aus der Analyse der fachlichen Anforderungen in den Kapiteln 3 und 4 wurde abgeleitet, dass der primäre Hot Spot der Applikation die Implementierung von verschiedenen Algorithmen zur Routenplanung und Störungsbehandlung ist.

In der technischen Architektur des Frameworks wird diese Vorgabe verarbeitet, indem der variable Aspekt der vom Anwender zu implementierenden Algorithmen in eine Algorithmen-Komponente ausgelagert wird, die durch eine API locker mit dem Rest der Anwendung verbunden ist.

Hier wird zuerst ein Überblick über diese Trennung gegeben, um dann auf die technische Architektur der beiden Teile des Frameworks einzugehen.

Abbildung 5.3 zeigt einen Überblick über die Aufspaltung der Anwendungsteile Planungs-/Simulationssteuerung und Algorithmen und die sie verbindende json API. Ziel dieser Trennung ist es, neben den in 5.5.2 gezeigten Möglichkeiten der Erweiterung, dem Anwender des Frameworks die weitere Option zu bieten, eine für ihn spezifische Algorithmen-Komponente zu implementieren (siehe auch 5.5.3). Es soll vermieden werden, den Anwender auf die hier getroffenen Designentscheidungen wie den Einsatz der Programmiersprache Java festzulegen.

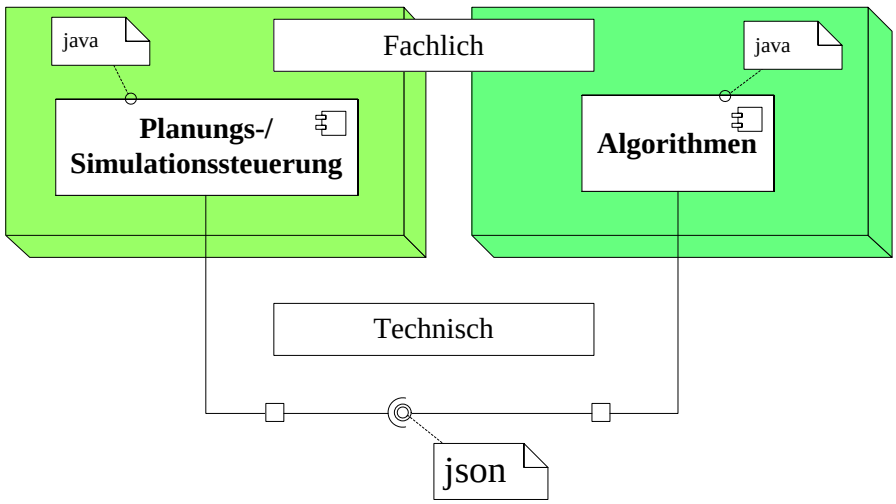

Abbildung 5.3.: Überblick Architektur

Die in Abbildung 5.4 gezeigte Planungs-/Simulationssteuerung beinhaltet neben der Steuerung und den Entitäten mit fachlicher Logik als technische Komponenten die Persistenzschicht (json) und den API Server.

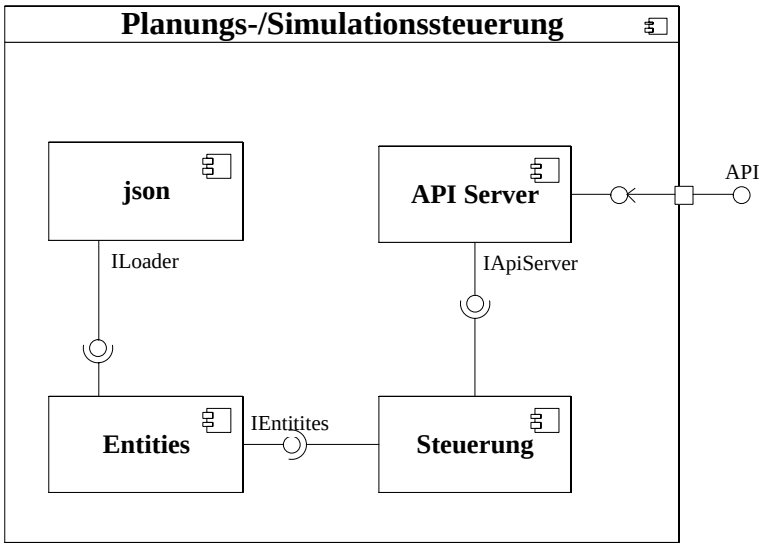

Abbildung 5.4.: Technische Architektur, Planungs-/Simulationssteuerung

Abbildung 5.5 zeigt den Algorithmen-Teil der Anwendung. Der grobe Aufbau ist dem der Planungs-/Simulationssteuerung ähnlich. Mit dem Unterschied, dass Entitäten über die API und nicht aus dem Dateisystem geladen werden. Die `IEntitiesRead` Schnittstelle stellt dem vom Anwender implementierten Algorithmus nur lesende Operationen auf Entitäten zur Verfügung, um die Kontrolle über Entitäten bei der Planungs-/Simulationssteuerung zu belassen.

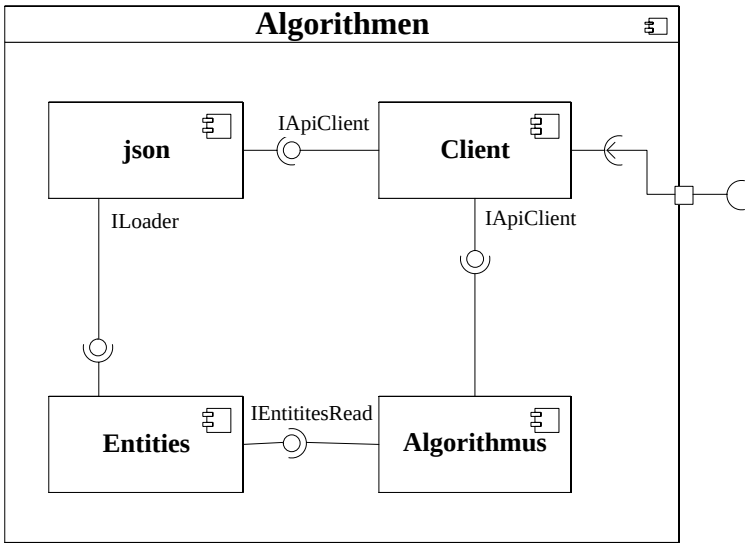

Abbildung 5.5.: Technische Architektur, Algorithmen

5.5.1.1. Message Bus

In der Simulation werden Ereignisse erzeugt, auf die unterschiedliche Komponenten reagieren müssen. Zum einen müssen fachliche Funktionen wie z. B. das Modifizieren der an einem Flughafen lagernden Gütermenge ausgeführt werden, zum anderen technische wie z. B. das Aktualisieren der graphischen Benutzeroberfläche. Zur Realisierung dieser Kommunikation stehen verschiedene Optionen zur Verfügung. Im einfachsten Fall z. B. die Verwendung eines Beobachtermusters (vgl. Gamma u. a., 1995, S. 293-303).

Hier wurde die Entscheidung getroffen, Ereignisse der Simulation über einen Message Bus zu kommunizieren. Diese Lösung bietet die Vorteile lockerer Kopplung zwischen den Objekten in der Simulation. Da das Wissen über Erzeuger und Abonnenten von Ereignissen zentral im Message Bus gekapselt und nicht wie beim Beobachter verteilt ist (vgl. Birman und Joseph, 1987, S. 130), (vgl. Hohpe und Woolf, 2003, S. 151-158). Abbildung 5.6 stellt diese Kommunikationsarchitektur schematisch dar und verdeutlicht die leichtere Erweiterbarkeit durch Hinzufügen neuer Erzeuger oder Abonnenten.

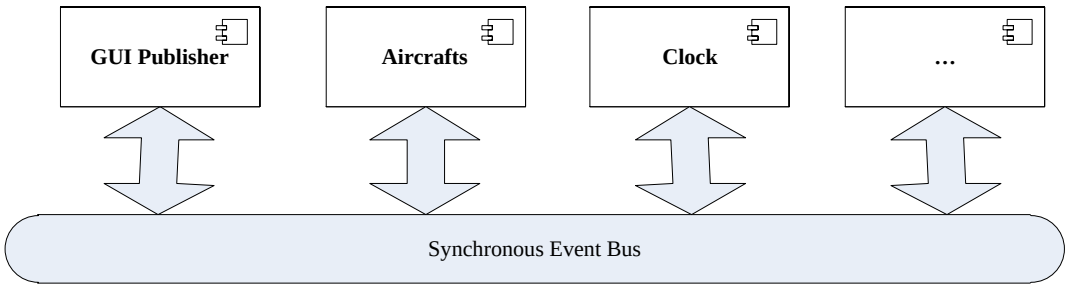

Abbildung 5.6.: Message Bus

5.5.2. Hooks

Eine Möglichkeit, das Erlernen eines neuen Frameworks zu erleichtern, ist die formale Dokumentation der im Framework vorhandenen Hot Spots, also der Ansatzpunkte für Erweiterungen im Framework (Flores und Aguiar, 2008). Froehlich u. a. (1997) beschreibt für diese Dokumentation die Technik der Hooks, in der jeder Erweiterungspunkt mit seinen Parametern in einem formalen Schema beschrieben wird. Hier sollen die zum Zeitpunkt des Entwurfs und der Implementierung erwarteten drei Erweiterungspunkte des Frameworks **dokumentiert** werden. Dies sind die Schnittstelle für neue Algorithmen, die fachlichen Störungen und die Erweiterung von Entitäten.

5.5.2.1. Algorithmen

Neue Algorithmen sollen durch den Anwender zum Framework hinzugefügt werden können, Abbildung 5.7 zeigt den Hook, der diesen Hot Spot dokumentiert. In diesem Fall wurde der Hot Spot intern als Strategiemuster (vgl. Gamma u. a., 1995, S. 315-323) modelliert. Die Modellierung mit einem bekannten Entwurfsmuster der Softwareentwicklung bietet sich bei Hot Spots an, da da es sich zum einem um eine erprobte Lösung handelt und zum anderen dem fachkundigen Anwender die Einarbeitung erleichtert wird, da er die verwendeten Strukturen schon kennt (Schmid, 1997). Hier wurde das Strategiemuster gewählt, um die Variabilität des Algorithmus zu kapseln.

Der Anwender wird bei der Realisierung eines neues Algorithmus unterstützt, indem seine Implementierung über Constructor Injection Referenzen auf die API und die Entitäten der Simulation übergeben wird. Diese automatische Übergabe von externen Abhängigkeiten

ermöglicht es ihm, sich auf die Realisierung der Fachlichkeit zu konzentrieren. Die Constructor Injection als spezielle Art der Depency Injection wurde hier gewählt, weil sie bei der kleinen Anzahl von Parametern und der voraussichtlich immer gleichen Art der Konstruktion eines Algorithmus-Objektes einfacher zu verwenden ist als Setter oder Interface Injection (Fowler, 2004).

Neben der Unterstützung des Anwenders durch diese Modellierung des Hot Spots in einer bekannten Art findet in diesem Hot Spot eine fachliche Unterstützung durch im Framework integrierte Werkzeuge der Graphentheorie statt. Der vom Anwender implementierte Algorithmus kann über die DI eine Darstellung des Routennetzwerkes zwischen Flughäfen als vom Framework erstellten gerichteten Graph erhalten, hierfür wird die jGraph Bibliothek verwendet (siehe 5.4.3). Das Framework bietet mit jGraph und der jGraphT auch Standardimplementierungen von Algorithmen der Graphentheorie, die der Anwender verwenden kann, sowie Optionen zur Visualisierung von Graphen.

Name Neuen Algorithmus hinzufügen

Requirement Ein neuer Algorithmus soll in die Applikation integriert werden

Area Werkzeuge

Type Feature hinzufügen

Participants Entities, API, Graph

Changes Vorgehen

1. Die Klasse `AbstractAlgorithm` erweitern
2. Planung implementieren
3. Den Algorithmus über das Konfigurationsfile bekannt machen

Abbildung 5.7.: Hook: Algorithmus hinzufügen

5.5.2.2. Störungen

Im fachlichen Datenmodell sind Störungen beschrieben, bei diesen wird hier davon ausgegangen, dass eine spätere Erweiterung notwendig sein wird. Dies, da zum einen die Störungen nicht den kompletten Umfang der in der Literatur beschriebenen Auswirkungen abdecken, so ist zum Beispiel im vorliegenden Entwurf nicht das komplette Schließen eines

Flughafens modellierbar. Weiterhin beinhalten Störungen regelmäßig einen Ausnahmezustand, der so nicht vorhergesehen wurde, was es unwahrscheinlich erscheinen lässt alle Möglichkeiten vorherzusehen und zu modellieren.

Um einerseits sicherzustellen, dass zum einen neue Arten von Störungen von außen in das System eingebracht werden und diese Entitäten beeinflussen können, wurde in diesem Entwurf eine Kombination der beiden Muster **Decorator** und **Builder** (vgl. Gamma u. a., 1995, S. 175-184, 97-106) gewählt. Die Definition einer Störung in der Form, wie sie der Szenarioplaner anlegt, beinhaltet unter anderem die Art der Störung. Aus der Art der Störung generiert der Builder zur Laufzeit die zu ladende Störungsklasse, was ein Einbringen neuer Störungen ermöglicht, auch wenn das Framework selber nicht im Quellcode vorliegt. Die Störungen werden durch das Decorator-Entwurfsmuster in die Lage versetzt, das Verhalten der Entitäten, die sie beeinflussen sollen, gegenüber dem Rest des Systems zu verändern.

5.5.2.3. Entität verändern

Die Erweiterungspunkte für Algorithmen und fachliche Störungen können nach den Kriterien aus dem Kapitel 2.2 als Glassbox-Erweiterungen eingeordnet werden, bei denen der interne Ablauf des Frameworks verstanden werden muss, aber der Quellcode nicht verändert wird. Das Verändern bestehender fachlicher Entitäten, um sie zum Beispiel um neue Attribute zu erweitern, erfordert im Gegensatz hierzu Veränderungen am bestehenden Quellcode des Frameworks. Da Entitätsänderungen während der Entwicklung regelmäßig zu Problemen geführt haben, wird das Vorgehen in Abbildung 5.8 beschrieben.

Name Entität anpassen

Requirement Eine Entität muss erweitert werden

Area Werkzeuge

Type Feature erweitern

Participants Entities, Serialiser

Changes Vorgehen

1. Entität erweitern
2. Serialisier anpassen
3. Deserialisier anpassen
4. Tests anpassen

Abbildung 5.8.: Hook: Entität erweitern

5.5.3. API

In 5.5.1 wurde die Trennung der Planungs-/Simulationssteuerung und der Algorithmen mit dem Ziel, eine Neuimplementierung der Algorithmen-Komponente in einer anderen Programmiersprache als Java zu ermöglichen, gezeigt. In diesem Abschnitt sollen zum einen die Gründe für diese Trennung erläutert und zum anderen die API beschrieben werden.

Der hier vorgestellte Entwurf basiert auf dem Paradigma der Objektorientierung, seine prototypische Implementierung wurde in Java durchgeführt. Während es Implementierungen anderer Sprachen und Paradigmen auch für Java Virtual Maschine (JVM), auf der Java ausgeführt wird, gibt, kann nicht ausgeschlossen werden, dass ein Wechsel der Programmiersprache bzw. Laufzeitumgebung für einen Algorithmus notwendig ist. Um ein Beispiel zu nennen, für die in den Grundlagen erwähnte (siehe 2.1.3.2) Lagrangian relaxation, bei der es sich um eine Methode zur Approximationslösung von Constraintnetzen handelt, ist Java wahrscheinlich nicht die optimale Programmiersprache. Um hier eine Lösung in einer für das Problem besser geeigneten Sprache zu ermöglichen, bietet die API eine wohldefinierte Schnittstelle, um mit der Planung und Simulation zu interagieren.

In Anhang B befindet sich eine Liste der Aufrufe, die die API bietet. Hier soll näher auf einige Designentscheidungen und ihre Auswirkungen eingegangen werden. In Kapitel 5.4.2 wurde

in Listing 5.1 ein Beispiel für einen Aufruf der API gezeigt. In diesem Aufruf sind die Parameter für die Prozedur als json-Objekt codiert. Diese Methode der **Codierung** wurde trotz ihrer gegenüber positionalen Parametern höheren Verbosität gewählt, um die Fehlersuche zu erleichtern. Sollte sich herausstellen, dass durch diese Art der Codierung die von der API erzeugten Datenmengen zu groß werden, kann dieses Problem durch Komprimierung gelöst werden (vgl. Sundaresan und Moussa, 2001) (vgl. Chen u. a., 2010, S. 3).

Die API selber kann analog zum für Störungen in 5.5.2 beschriebenen Vorgehen von außen **erweitert werden**. Bei der Implementierung des API Servers wurden spätere Erweiterungen durch eine hohe Modularisierung eingeplant, Listing 5.2 zeigt beispielhaft die serverseitige Implementierung des in Listing 5.1 gezeigten GetEntityList RPCs. Die besondere Aufmerksamkeit, die die Erweiterbarkeit der API erhält, ergibt sich daraus, dass eine API sich in der Regel inkrementel weiterentwickelt, so dass Änderungen an dieser Stelle des Frameworks wahrscheinlich sind (vgl. Tulach, 2008, S. 59-63).

Listing 5.2: Implementierung GetEntityList

```
public class ApiCommandGetEntityList extends
   AbstractApiCommand {

  String entityType;

  public ApiCommandGetEntityList(JSONObject params) {
    super(params);
    this.entityType = (String) params.get("type");
  }

  @Override
  public CommandResult execute() {
    logger.trace("execute()");
    String resultString = "";
    ILoader loader = new Loader(new RawJSONLoader());
    IEntities entities = new Entities(loader);
    List<IEntity> entityList = entities.getEntityList(this.
       entityType);
    resultString = loader.entityListAsJSON(entityList);
    return new CommandResultSuccess(resultString);
  }
}
```

Im **Testkonzept** wurden Integrationstests für die API beschrieben, diese Tests sind im Prototypen noch keine vollständige Testsuite. Die Testabdeckung von API Funktionen mit positiven

Testfällen liegt hier trotzdem schon bei 100 %, d. h. jede Funktion in der API ist mit mindestens einem Testfall versehen, in der Regel mit mehreren. Diese höhere Testabdeckung von Funktionen in der API als im Rest des Prototypen wurde gewählt, um die Nutzbarkeit der API sicherzustellen, der Test ist hier der erste Anwender. Ein weiterer Vorteil der hohen Testabdeckung in dieser Umgebung ist, dass die Tests als Spezifikation im Quellcode vorhanden sind (vgl. Tulach, 2008, S. 149-159).

5.6. Probleme während der Implementierung

Die während der Implementierung des Prototypen aufgetretenen Probleme in **Apache Mina** und der **Oberfläche** sollen hier kurz mit Lösungsvorschlägen für eine spätere Entwicklung erläutert werden.

Die für die Abstraktion des Netzwerkes gewählte **Apache Mina** Bibliothek hat, wie sich während der Implementierung herausstellte, einen Fehler, der es unmöglich macht, in der Server-Komponente einen Timeout zu konfigurieren. Im Ergebnis kann dies dazu führen, dass ein nicht korrekt arbeitender Client den Server blockiert. Da durch Inkompatibilitäten zwischen den Build-Umgebungen für Mina und diese Arbeit die Erstellung eines Patchs zu aufwendig gewesen wäre, wurde darauf verzichtet, diesen Fehler zu beheben. Sollte er später auftreten, muss der fehlende Konstruktor in der Klasse `NioSocketAcceptor` hinzugefügt werden (vgl. amt, 2011).

Ursprünglich war geplant, die webbasierte **Oberfläche** über Websockets Push-basiert an die API anzubinden. Dieses Vorhaben konnte so nicht umgesetzt werden, da es zu Problemen mit der Decodierung der HTTP-Transportschicht gekommen ist. Der Oberflächen-Prototyp ist zum Zeitpunkt des Abschlusses dieser Arbeit über einen Pull-Mechanismus an die Simulation angeschlossen. Bei einer Simulation mit vielen Objekten könnte dies zu Problemen mit der Darstellung führen (vgl. Tanenbaum und van Steen, 2007, S. 334-336). In der weiteren Entwicklung muss hier ein HTTP-Transport für Apache Mina eingebunden und der GUI Publisher, der an den Message Bus angeschlossen ist (siehe 5.5.1.1), modifiziert werden, um diesen Transport zu nutzen.

Teil III.

Schluss

6. Zusammenfassung und Ausblick

Im Folgenden werden die Ergebnisse der Arbeit zusammengefasst und ein Ausblick auf weitere Entwicklungsmöglichkeiten und Forschungsgebiete gegeben.

6.1. Zusammenfassung

Es wurde in dieser Arbeit ein **domainspezifischer Frameworkentwurf** entwickelt, der bei vollständiger Realisierung das Ziel, bei der Beurteilung der fachlichen Wertigkeit von Algorithmen zu unterstützen, erreicht. Der Anwender des Frameworks kann sich losgelöst von der Technik der Simulation und der Planungssteuerung auf die **Entwicklung von Algorithmen** konzentrieren.

Hierzu wurden im Kapitel 2 aus der Literatur die fachlichen und technischen **Grundlagen herausgearbeitet**. In Kapitel 3 wurden die aus der Anforderungsanalyse gewonnenen fachlichen Anforderungen systematisch aufgearbeitet und strukturiert. Ziel dieses Kapitels war es, als Vorarbeit zum technischen Entwurf eine **fachliche Architektur** des Frameworks zu entwickeln. Dieses Ziel wurde durch ein Bottom-up Vorgehen verwirklicht, indem aus dem fachlichen Datenmodell und dem sich daraus ergebenden Komponentenschnitt die Architektur entwickelt wurde. Im Kapitel **Entwurf und Implementierung** wurde nach dem technischen Entwurf dessen Realisierbarkeit durch die prototypische Realisierung kritischer Stellen im Framework abgesichert.

Des Weiteren wurde gezeigt, wie durch Modularisierung der Architektur die Erweiterbarkeit gefördert werden kann. Hierzu wurde die Erweiterbarkeit des Frameworks in mehrere Stufen gegliedert:

1. Unterstützte Erweiterungen
2. Erwartete Erweiterungen
3. Sprachunabhängige Schnittstellen

In der **unterstützten Erweiterung** kann der Anwender die zu evaluierenden Algorithmen über das hierfür verwendete Strategiemuster (vgl. Gamma u. a., 1995, S. 315-323) in das Framework einbringen. Hierbei wird er durch die Werkzeuge, die ihm das Framework zur Implementierung seiner Algorithmen bereitstellt, wie zum Beispiel die graphentheoretischen Erweiterungen, unterstützt (siehe 5.4.3). Dies ist die für den Anwender einfachste Form der Erweiterung. Die Architektur des Frameworks ist so designt, dass an einigen *Hot Spots* **erwartete Erweiterungen** ermöglicht werden. So unterstützt zum Beispiel der Message Bus mit seiner Erzeuger-/Abonnenten-Semantik die Erweiterung der Simulation um neue Statistiken. Diese können durch das Hinzufügen eines Abonnenten leicht auf alle Ereignisse der Simulation Zugriff erhalten. Die Implementierung von Erweiterungen verlangt vom Anwender jedoch ein tieferes Verständnis für die interne Funktionsweise des Frameworks. Er wird hier, durch das Fehlen von Werkzeugen nicht so stark unterstützt wie bei der Implementierung von Algorithmen. Abschließend bietet das Framework durch die **sprachunabhängige Schnittstelle** zwischen der Planungs-/Simulationssteuerung und der Algorithmen-Komponente dem Anwender die Möglichkeit, eine komplett für ihn spezialisierte Algorithmen-Komponente zu implementieren und trotzdem die Simulation zu nutzen. Dies kann sinnvoll sein, wenn zum Beispiel ein anderes Paradigma der Programmierung für die Algorithmen genutzt werden soll als das hier verwendete objektorientierte.

6.2. Ausblick

Die Entwicklung eines Frameworks ist immer ein iterativer Prozess, der im Idealfall zu einer Blackbox führt, siehe Kapitel 2.2. Diesen Grundsatz bedenkend sollte die weitere Entwicklung des Frameworks nach einer vollständigen Implementierung des vorliegenden Konzeptes in enger Zusammenarbeit mit der Anwendungsentwicklung durchgeführt werden. Hierdurch werden zum einen Fehler in der vorliegenden Spezifikation aufgedeckt als auch die Richtung für Erweiterungen des Frameworks vorgegeben.

Vom derzeitigen Standpunkt aus bieten sich mehrere Alternativen zur weiteren Entwicklung an. Eine interessante Option stellt die Erweiterung der Oberfläche um **interaktive** Elemente dar. Im Sinne eines *Serious Games* kann dem Nutzer des Frameworks hier die direkte Manipulation der Simulation erlaubt werden. Dieser Ansatz eröffnet unter anderem die Möglichkeit, durch Abdecken von Grenzfällen die **Algorithmen-Entwicklung zu erleichtern**. Der Anwender könnte die Aufgabe eines Disponenten wahrnehmen, wenn z. B. der eingesetzte Algorithmus zum Verzögerungsmanagement in einen undefinierten Zustand gerät. Ein weiterer, durch manuelles Eingreifen in den Simulationsprozess erreichbarer Vorteil ist die Möglichkeit, die Aktionen der Disponenten aufzuzeichnen, „Improvement starts with measurement“ (Banks, 1998).

Eine zweite Möglichkeit ist, den Fokus der Forschung auf **Algorithmendesign und -evaluation** zu legen. Eine erste Suche in der Literatur zeigt z. B. keine Analysen zum Ausführen von Algorithmen zur Routenplanung bei Airlines auf modernen Graphikkarten. Speziell die Ausführung der in den Grundlagen kurz angesprochenen Lösung des Routenplanungsproblems über lineare Optimierung (Erdmann u. a., 2001) wirkt auf aktuellen Graphikkarten, z. B. nVidia *CUDA* kompatiblen untersuchenswert.

Literaturverzeichnis

[amt 2011] AMT: *NioSocketAcceptor doesnot provide an interface to input connectiontimeout parameter.* 2011. – URL https://issues.apache.org/jira/browse/DIRMINA-843. – Zugriffsdatum: 03.08.2011

[Bagrodia 1996] BAGRODIA, Rajive L.: Perils and pitfalls of parallel discrete-event simulation. In: *Proceedings of the 28th conference on Winter simulation*. Washington, DC, USA : IEEE Computer Society, 1996 (WSC '96), S. 136–143. – URL http://dx.doi.org/10.1145/256562.256592. – ISBN 0-7803-3383-7

[Ball u. a. 2006] BALL, Michael ; BARNHART, Cynthia ; NEMHAUSER, George ; ODONI, Amedeo: Air transportation: irregular operations and control. In: *Handbooks of Operations Research and Management*, 2006, S. 1–67

[Banks 1998] BANKS, Jerry (Hrsg.): *Handbook of Simulation*. Wiley-Interscience, 1998. – Principles, Methodolgy, Advances, Applications, and Practice. – ISBN 0-471-13403-1

[Beck 2001] BECK, Kent: *Manifesto for Agile Software Development*. 2001. – URL http://agilemanifesto.org/. – Zugriffsdatum: 12.07.2011

[Birman und Joseph 1987] BIRMAN, K. ; JOSEPH, T.: Exploiting virtual synchrony in distributed systems. In: *SIGOPS Oper. Syst. Rev.* 21 (1987), November, S. 123–138. – URL http://doi.acm.org/10.1145/37499.37515. – ISSN 0163-5980

[Buschmann u. a. 2004] BUSCHMANN, Frank ; MEUNIER, Regine ; ROHNERT, Hans ; SOMMERLAD, Peter ; STAL, Michael: *Wiley series in software design patterns*. Bd. 1: *Pattern-Oriented Software Architecture*. Wiley, 2004. – ISBN 0-471-95869-7

[Chen u. a. 2010] CHEN, Yanpei ; GANAPATHI, Archana ; KATZ, Randy H.: To compress or not to compress - compute vs. IO tradeoffs for mapreduce energy efficiency. In: *Proceedings of the first ACM SIGCOMM workshop on Green networking*. New York, NY, USA : ACM, 2010 (Green Networking '10), S. 23–28. – URL http://doi.acm.org/10.1145/1851290.1851296. – ISBN 978-1-4503-0196-1

[Cheng-Lung 2010] CHENG-LUNG, Wu: *Airline Operations and Delay Management*. Ashgate Publishing Company, 2010

[Cloodt 2011] CLOODT, Hubertus: Signs of recovery for air transport in Europe in 2009. In: *Statistics in focus* 2 (2011), S. 1–12

[Cormen u. a. 2009] CORMEN, Thomas H. ; LEISERON, Charles E. ; RIVEST, Ronald L. ; STEIN, Clifford: *Introduction to Algorithms*. 3. MIT Press, 2009. – ISBN 978-0-262-03384-8

[Crockford 2006] CROCKFORD, D.: *The application/json Media Type for JavaScript Object Notation (JSON)*. RFC 4627 (Informational). Juli 2006. – URL http://www.ietf.org/rfc/rfc4627.txt

[Dieter u. a. 2004] DIETER, Arnold (Hrsg.) ; KUHN, Axel (Hrsg.) ; ISERMANN, Heinz (Hrsg.) ; TEMPELMEIER, Horst (Hrsg.): *Handbuch Logistik*. 2. Springer, 2004. – ISBN 3-540-40110-5

[Erdmann u. a. 2001] ERDMANN, A. ; NOLTE, A. ; NOLTEMEIER, A. ; SCHRADER, R.: Modeling and Solving an Airline Schedule Generation Problem. In: *Annals of Operations Research* Bd. 107. Kluwer Academic Publishers, 2001, S. 117–142

[Fayad u. a. 2000] FAYAD, Mohamed E. ; HAMU, David S. ; BRUGALI, Davide: Enterprise frameworks characteristics, criteria, and challenges. In: *Commun. ACM* 43 (2000), October, S. 39–46. – URL http://doi.acm.org/10.1145/352183.352200. – ISSN 0001-0782

[Fayad u. a. 1999] FAYAD, Mohamed E. ; SCHMIDT, Douglas C. ; JOHNSON, Ralph E.: *Building Application Frameworks*. 1. Wiley Computer Publishing, 1999. – Object-Oriented Foundations of Framework Design. – ISBN 0-471-24875-4

[Federal Aviation Administration Flight Standards Service 2007] FEDERAL AVIATION ADMINISTRATION FLIGHT STANDARDS SERVICE (Hrsg.): *Aircraft Weight and Balance Handbook*. U.S. Department of Transportation, 2007

[Fielding 2000] FIELDING, R.T.: *Architectural styles and the design of network-based software architectures*, University of California, Irvine, PhD Thesis, 2000

[Flores und Aguiar 2008] FLORES, Nuno ; AGUIAR, Ademar: Patterns for understanding frameworks. In: *Proceedings of the 15th Conference on Pattern Languages of Programs*. New York, NY, USA : ACM, 2008 (PLoP '08), S. 8:1–8:11. – URL http://doi.acm.org/10.1145/1753196.1753206. – ISBN 978-1-60558-151-4

[Foote 1988] FOOTE, Brian: *Designing to facilitate change with object-oriented frameworks*, University of Illinois at Urbana-Champaign, Diplomarbeit, 1988. – Masterthesis

[Foote 1991] FOOTE, Brian: The lifecycle of object-oriented frameworks: A fractal perspective / University of Illinois at Urbana-Champaign. 1991. – Forschungsbericht. Techreport

[Fowler 2004] FOWLER, Martin: *Inversion of Control Containers and the Dependency Injection pattern*. 2004. – URL http://martinfowler.com/articles/injection.html. – Zugriffsdatum: 01.08.2011

[Fowler 2006] FOWLER, Martin: *RoleInterface*. 2006. – URL http://martinfowler.com/bliki/RoleInterface.html. – Zugriffsdatum: 17.08.2011

[Froehlich u. a. 2000] FROEHLICH, Garry ; HOOVER, H. J. ; SORENSON, Paul G.: Choosing an object-oriented domain framework. In: *ACM Comput. Surv.* 32 (2000), March. – URL http://doi.acm.org/10.1145/351936.351953. – ISSN 0360-0300

[Froehlich u. a. 1997] FROEHLICH, Gary ; HOOVER, H. J. ; LIU, Ling ; SORENSON, Paul: Hooking into object-oriented application frameworks. In: *Proceedings of the 19th international conference on Software engineering*. New York, NY, USA : ACM, 1997 (ICSE '97), S. 491–501. – URL http://doi.acm.org/10.1145/253228.253432. – ISBN 0-89791-914-9

[Fujimoto 1990] FUJIMOTO, Richard M.: Parallel discrete event simulation. In: *Commun. ACM* 33 (1990), October, S. 30–53. – URL http://doi.acm.org/10.1145/84537.84545. – ISSN 0001-0782

[Gamma u. a. 1995] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John M.: *Design Patterns: Elements of Reusable Object-Oriented Software*. 1. Addison-Wesley Professional, 1995. – ISBN 0201633612

[Grandjot u. a. 2007] GRANDJOT, Hans-Helmut ; ROESSLER, Ingo ; ROLAND, Ailine: *Air Cargo Guide*. HUSS Verlag, 2007. – An introduction to the air cargo industry. – ISBN 978-3-937711-50-8

[Hohpe und Woolf 2003] HOHPE, Gregor ; WOOLF, Bobby: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2003. – ISBN 0321200683

[IATA 2011] IATA (Hrsg.): *Dangerous goods regulations*. 52. International Air Transport Association, 2011

[JSON RPC Project a] JSON RPC PROJECT: *JSON-RPC 2.0 Specification proposal*. – URL http://groups.google.com/group/json-rpc/web/json-rpc-1-2-proposal. – Zugriffsdatum: 26.07.2011

[JSON RPC Project b] JSON RPC PROJECT: *JSON-RPC Project*. – URL http://www.jsonrpc.org. – Zugriffsdatum: 26.07.2011. – JSON RPC Projectpage

[Kecher 2009] KECHER, Christoph: *UML 2.0*. 3. Galileo Press, 2009. – ISBN 9783836214193

[Ludewig und Lichter 2006] LUDEWIG, Jochen ; LICHTER, Horst: *Software Engineering : Grundlagen, Menschen, Prozesse, Techniken*. 2. Addison-Wesley, 2006

[Luxem 1993] LUXEM, Redmer: *Untersuchungen zur Datenmodellierung am Beispiel eines Logistik-Informationssystems in der metallverarbeitenden Industrie*, Universität - Gesamthochschule Siegen, Diplomarbeit, 1993

[Narayanasamy u. a. 2006] NARAYANASAMY, Viknashvaran ; WONG, Kok W. ; FUNG, Chun C. ; RAI, Shri: Distinguishing games and simulation games from simulators. In: *Comput. Entertain.* 4 (2006), April. – URL http://doi.acm.org/10.1145/1129006.1129021. – ISSN 1544-3574

[P. Brooks 2010] P. BROOKS, Frederik: *The Design of Design*. 1. Addison-Wesley, 2010. – Essays from a Computer Scientist. – ISBN 0-201-36298-8

[Potzner 2008] POTZNER, Andreas: *Innovationskooperation entlang Supply Chains*, European Business School, Dissertation, 2008. – Eine Analye der europäischen Aviation-Industrie

[Royce 1987] ROYCE, W. W.: Managing the development of large software systems: concepts and techniques. In: *Proceedings of the 9th international conference on Software Engineering*. Los Alamitos, CA, USA : IEEE Computer Society Press, 1987 (ICSE '87), S. 328–338. – URL http://portal.acm.org/citation.cfm?id=41765.41801. – ISBN 0-89791-216-0

[Sarstedt 2009] SARSTEDT, Stefan: *Software Engineering 1*. 2009. – Unterlagen zur Vorlesung

[Schmid 1997] SCHMID, Han A.: Systematic framework design by generalization. In: *Commun. ACM* 40 (1997), October, S. 48–51. – URL http://doi.acm.org/10.1145/262793.262803. – ISSN 0001-0782

[Schulte 2008] SCHULTE, Christof: *Logistik. Wege zur Optimierung der Supply Chain*. 5. Verlag Franz Vahlen, 2008

[Shan und Hua 2006] SHAN, Tony C. ; HUA, Winnie W.: Taxonomy of Java Web Application Frameworks. In: *Proceedings of the IEEE International Conference on e-Business Engineering*. Washington, DC, USA : IEEE Computer Society, 2006 (ICEBE '06), S. 378–385. – URL http://dx.doi.org/10.1109/ICEBE.2006.98. – ISBN 0-7695-2645-4

[Sharp und Robinson 2005] SHARP, Helen ; ROBINSON, Hugh: Some social factors of software engineering: the maverick, community and technical practices. In: *Proceedings of the 2005 workshop on Human and social factors of software engineering*. New York, NY,

USA : ACM, 2005 (HSSE '05), S. 1–6. – URL http://doi.acm.org/10.1145/1082983.1083117. – ISBN 1-59593-120-1

[Spillner und Linz 2005] SPILLNER, Andreas ; LINZ, Tilo: *Basiswissen Softwaretest*. dpunkt.verlag, 2005

[Steudte 2011] STEUDTE, Armin: *Konzeption und Entwicklung eines ereignisgesteuerten Frameworks für Ambient Assisted Living Anwendungen*. 2011

[Sundaresan und Moussa 2001] SUNDARESAN, Neel ; MOUSSA, Reshad: Algorithms and programming models for efficient representation of XML for Internet applications. In: *Proceedings of the 10th international conference on World Wide Web*. New York, NY, USA : ACM, 2001 (WWW '01), S. 366–375. – URL http://doi.acm.org/10.1145/371920.372090. – ISBN 1-58113-348-0

[Szyperski u. a. 2002] SZYPERSKI, Clemens ; GRUNTZ, Dominik ; MURER, Stephan: *Component software: beyond object-oriented programming*. 2. Pearson Educationy, 2002 (Component software series). – ISBN 9780201745726

[Tanenbaum 2003] TANENBAUM, Andrew: *Computer Networks*. 4. Prentice Hall PTR, 2003. – ISBN 0-13-038488-7

[Tanenbaum und van Steen 2007] TANENBAUM, Andrew S. ; STEEN, Maarten van: *Verteilte Systeme*. 2., Aufl. Pearson Studium, 2007. – ISBN 3827372933

[Tulach 2008] TULACH, Jaroslav: *Practical API Design*. 1. Apress, 2008. – Confessions of a Java Framework Architect. – ISBN 978-1-4302-0973-7

[Wikipedia] WIKIPEDIA: *Serialisierung*. – URL http://de.wikipedia.org/wiki/Serialisierung. – Zugriffsdatum: 26.07.2011

[Wilcox u. a. 1997] WILCOX, E. M. ; ATWOOD, J. W. ; BURNETT, M. M. ; CADIZ, J. J. ; COOK, C. R.: Does continuous visual feedback aid debugging in direct-manipulation programming systems? In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM, 1997 (CHI '97), S. 258–265. – URL http://doi.acm.org/10.1145/258549.258721. – ISBN 0-89791-802-9

Teil IV.

Anhang

A. Anforderungen

A.0.1. Stammdaten

A.0.1.1. Mehrere Flughäfen

Das System muss mehrere Flughäfen unterstützen.

A.0.1.2. Kapazität

Ein Flughafen hat eine Kapazität von x Flugzeugen.

A.0.1.3. Verschieden Typen von Flugzeugen

Das System muss verschieden Typen von Flugzeugen unterstützen.

A.0.1.4. Typen von Flugzeugen

Ein Flugzeugtyp definiert sich über: Namen, Zuladung, Geschwindigkeit.

A.0.1.5. Definition von Flugzeugtypen

Der Szenarioplaner pflegt mit den Stammdaten die Flugzeugtypen ein.

A.0.1.6. Maximale Abmessungen Frachtstück

Das System muss sicherstellen, dass die maximalen Abmessungen pro Frachtstück (abhängig vom Flugzeugtyp) eingehalten werden.

A.0.1.7. Maximales Volumen Zuladung

Das System muss sicherstellen, dass das maximale Volumen der Gesamtfracht (abhängig vom Flugzeugtyp) eingehalten wird.

A.0.1.8. Maximales Gewicht Frachtstück

Das System muss sicherstellen, dass das maximale Gewicht pro Frachtstück (abhängig vom Flugzeugtyp) eingehalten wird.

A.0.1.9. Maximales Gewicht Zuladung

Das System muss sicherstellen, dass das maximale Gewicht der Gesamtfracht (abhängig vom Flugzeugtyp) eingehalten wird.

A.0.1.10. Kodierung von Stammdaten

Stammdaten werden außerhalb des AWKs json-kodiert.

A.0.1.11. Laden von Stammdaten

Die Anwendung lädt Stammdaten aus lokal abgelegten Dateien.

A.0.1.12. Prüfen von Stammdaten

Das System prüft Stammdaten auf syntaktische und semantische Korrektheit, bevor es sie verwendet.

A.0.2. Störungen

A.0.2.1. Störungen

Die Simulation unterstützt Störungen.

A.0.2.2. Arten von Störungen

Die Simulation unterstützt Störungen, die entweder räumlich oder auf ein spezielles Objekt der Simulation (Flughafen, Flugzeug) bezogen sind.

A.0.2.3. Startzeit von Störungen

Störungen haben eine Startzeit.

A.0.2.4. Dauer von Störungen

Störungen haben eine Dauer, diese kann auch permanent sein.

A.0.2.5. Auswirkungen von Störungen auf Flughäfen

Störungen können die Abfertigungszeit für Flugzeuge am Flughafen erhöhen oder den Flughafen komplett sperren.

A.0.2.6. Auswirkungen von Störungen auf Flugzeuge

Störungen können die Abfertigungszeit für Flugzeuge am Flughafen erhöhen, die Reisegeschwindigkeit senken oder zum Totalverlust des Flugzeuges führen.

A.0.2.7. Anfälligkeit für Störungen

Es sind sowohl Flugzeuge am Boden als auch in der Luft für Störungen anfällig.

A.0.2.8. Umplanungen bei Störungen

Die Simulation passt den Netzplan an Störungen an, wenn diese auftreten oder beendet werden.

A.0.2.9. Umplanungen bei laufenden Flugbewegungen

Laufende Flugbewegungen werden nur bei unbedingter Notwendigkeit umgeplant, da dadurch die Ladung verloren geht.

A.0.2.10. Erstellen von Störungen

Der Szenarioplaner erstellt Störungen als Stammdaten.

A.0.2.11. Simulation von Störungen

Die Simulation startet die Störung und beendet sie nach Ablauf ihrer Dauer.

A.0.3. API

A.0.3.1. Einbringung von Algorithmen

Die Anwendung verfügt über eine API um für die Planung von Strecken von außen Algorithmen annehmen zu können.

A.0.3.2. API Sprachunabhängigkeit

Die API ist programmiersprachenunabhängig.

A.0.3.3. API Klartext

Die API benutzt zur Kommunikation mit Algorithmen ein Klartextformat (in Abgrenzung zu einem Binärformat).

A.0.3.4. API Lesen von Daten

Die API stellt Funktionalitäten zur Verfügung, um alle Stammdaten lesen zu können.

A.0.3.5. Schreiben von Stammdaten

Über die API können Flugrouten geschrieben werden.

A.0.3.6. API Kontrolfluss

Der Kontrollfluss geht von der Anwendung aus. Sie stellt eine Anfrage über die API und erhält eine Antwort.

A.0.4. GUI

A.0.4.1. Anzeige von Flugzeugen

Die GUI stellt Flugzeuge sowohl im Flug als auch am Boden dar

A.0.4.2. Position von Flugzeugen

Alle Flugzeuge nutzen einen Indikator, um ihre Position anzuzeigen.

A.0.4.3. Position von Flughäfen

Alle Flughäfen nutzen einen Indikator, um ihre Position anzuzeigen.

A.0.4.4. Anzeige von Störungen

Die GUI stellt aktive Störungen graphisch dar.

A.0.4.5. Anzeige von räumlichen Störungen

Die GUI stellt eine räumliche Störung durch farbliches Unterlegen des gestörten Bereiches dar.

A.0.4.6. Anzeige des Flugzeugstatus

Ein Flugzeug kann im Status „Normal“ oder im Status „Gestört“sein. Im Status „Normal“ ist der Indikator grün im Status gestört rot.

A.0.4.7. Anzeige des Flugzeugstatus

Ein Flughäfen kann im Status „Normal“, „Gestört“ oder „Geschloßen“ sein. Im Status „Normal“ ist der Indikator grün im Status „gestört“ gelb und im Status „Geschloßen“ rot.

A.0.5. Prämissen

A.0.5.1. P-5: Modifizieren der Stammdaten

Stammdaten werden mit Applikationen, die nicht Bestandteil der Anwendung sind, modifiziert.

A.0.5.2. P-5: Erstellen der Stammdaten

Stammdaten werden mit Applikationen, die nicht Bestandteil der Anwendung sind, erstellt.

A.0.6. Abgrenzungen

A.0.6.1. Kapazität Flughäfen

Die Anwendung unterscheidet bei der Kapazität von Flughäfen nicht zwischen verschiedenen Flugzeugtypen.

A.0.6.2. Namen der Stammdatendatei

Die Namen der Dateien, in denen Stammdaten abgelegt werden, müssen nicht änderbar sein. Sie können als Konstante gesetzt werden.

A.0.6.3. Modifizieren der Stammdaten

Die Anwendung bietet kein Möglichkeit, um Stammdaten zu modifizieren.

A.0.6.4. Weight and Balance

Die Anwendung führt keine *Weight and Balance* Checks außer den unter →Stammdaten Gewichts- und Volumen-Checks durch.

A.0.6.5. API Schreiben

Über die API können außer den in →Schreiben von Stammdaten genannten keine Daten geschrieben werden.

B. Aufrufe in der API

Schließen der Verbindung `Bye()`

No Operation `Noop()`

Laden von Entitäten `GetEntityList()`

Echo Params `Echo()`

Laden der Zykluslänge `GetCycleLength()`

Schedule für ein Flugzeug setzen `SetSchedule()`

Schedule für ein Flugzeug löschen `DelSchedule()`

Prüft ob eine Entität von einer Störung betroffen ist `IsAffected?()`

Setzt nach der fertigen Umplanung die Simulation wieder in Gang `ContinueSimulaton()`

C. Entities

Entität	**Flugzeug**
Attribute	• TailNumber: AlphNum • Kosten: GeldTyp • Geschwindigkeit: Float • Klasse: TransportKlasse • Position: PositionsTyp • Kapazität: KapazitätsTyp
Schlüssel	• TailNumber
Beziehungen	• Flugzeuge nutzen Routen • Flugzeuge nutzen Flughäfen • Flugzeuge erfühlen durch Transport Bedarfe • Flugzeuge werden durch Störungen betroffen

Tabelle C.1.: Entität: Flugzeug

Entität	Flughafen
Attribute	• Kosten: KostenTyp • Turnaround: TurnaroundTyp • Code: IATA code • Position: PositionsTyp
Schlüssel	• Code
Beziehungen	• Flughäfen werden durch Routen verbunden • Flughäfen werden von Flugzeugen genutzt • Flughäfen werden durch Störungen betroffen

Tabelle C.2.: Entität: Flughafen

Entität	Störung
Attribute	• Dauer: Zeit • Region: RegionsTyp • Auswirkung: AuswirkungsTyp
Schlüssel	• –
Beziehungen	• Störungen betreffen Routen • Störungen betreffen Flughäfen • Störungen betreffen Flugzeuge

Tabelle C.3.: Entität: Störung

Entität	Route
Attribute	• Start: IATA code • Ende: IATA code • KapazitätProStunde: Nummer
Schlüssel	• Start • Ende
Beziehungen	• Routen verbinden Flughäfen • Routen werden von Flugzeugen genutzt • Routen sind von Störungen betroffen

Tabelle C.4.: Entität: Route

Entität	Bedarf
Attribute	• Auftragsnummer: Nummer • Abgangsort: IATA code • Lieferort: IATA code • Startdatum: DatumsTyp • Enddatum: DatumsTyp • Kapazität: KapazitätsTyp • Kosten: GeldTyp
Schlüssel	• Auftragsnummer
Beziehungen	• Bedarfe werden von Flugzeugen befriedigt

Tabelle C.5.: Entität: Bedarf

Entität	Prognostizierter Bedarf
Attribute	• Wahrscheinlichkeit: Prozent • Abgangsort: IATA code • Lieferort: IATA code • Startdatum: DatumsTyp • Enddatum: DatumsTyp • Kapazität: KapazitätsTyp • Kosten: GeldTyp
Schlüssel	• Auftragsnummer
Beziehungen	• Ein prognostizierter Bedarf antizipiert reale Bedarfe

Tabelle C.6.: Entität: Prognostizierter Bedarf

Datentyp	**Beschreibung**
AlphNum	Alphanumerische Zeichenkette
Float	Eine Fließkommazahl
Nummer	Eine Integer Nummer
DatumsTyp	Ein Datum mit Zeitinformationen
Zeit	Ein Zeitraum ohne Datum in Minuten (z. B. 60)
GeldTyp	Der GeldTyp beschreibt eine Menge von Geld. Er wird in Cent gespeichert, also z.B. 120 für 1,20 €
TransportKlasse	Die TransportKlasse fasst verschiedene Flugzeuge zu Klassen zusammen. Sie kann 1, 2 oder 3 sein. Auf einem Flughafen mit einer Transportklasse von x können alle Flugzeuge mit einer Klasse $\leq x$ landen.
PositionsTyp	Eine Koordinate aus Latitude und Longitude
KapazitätsTyp	Der KapazitätsTyp beschreibt eine Menge Fracht durch Gewicht und Volumen. Also z. B. 3 kg und 1 m^3
TurnaroundTyp	Beschreibt wie schnell Maschinen einer Transportklasse x abgefertigt werden können. (z.B. Klasse 2, 20 Minuten)
IATA code	IATA-Flughafencode (3 Buchstaben alphanumerisch)

Tabelle C.7.: Entität: Datentypen

D. Logistikkette

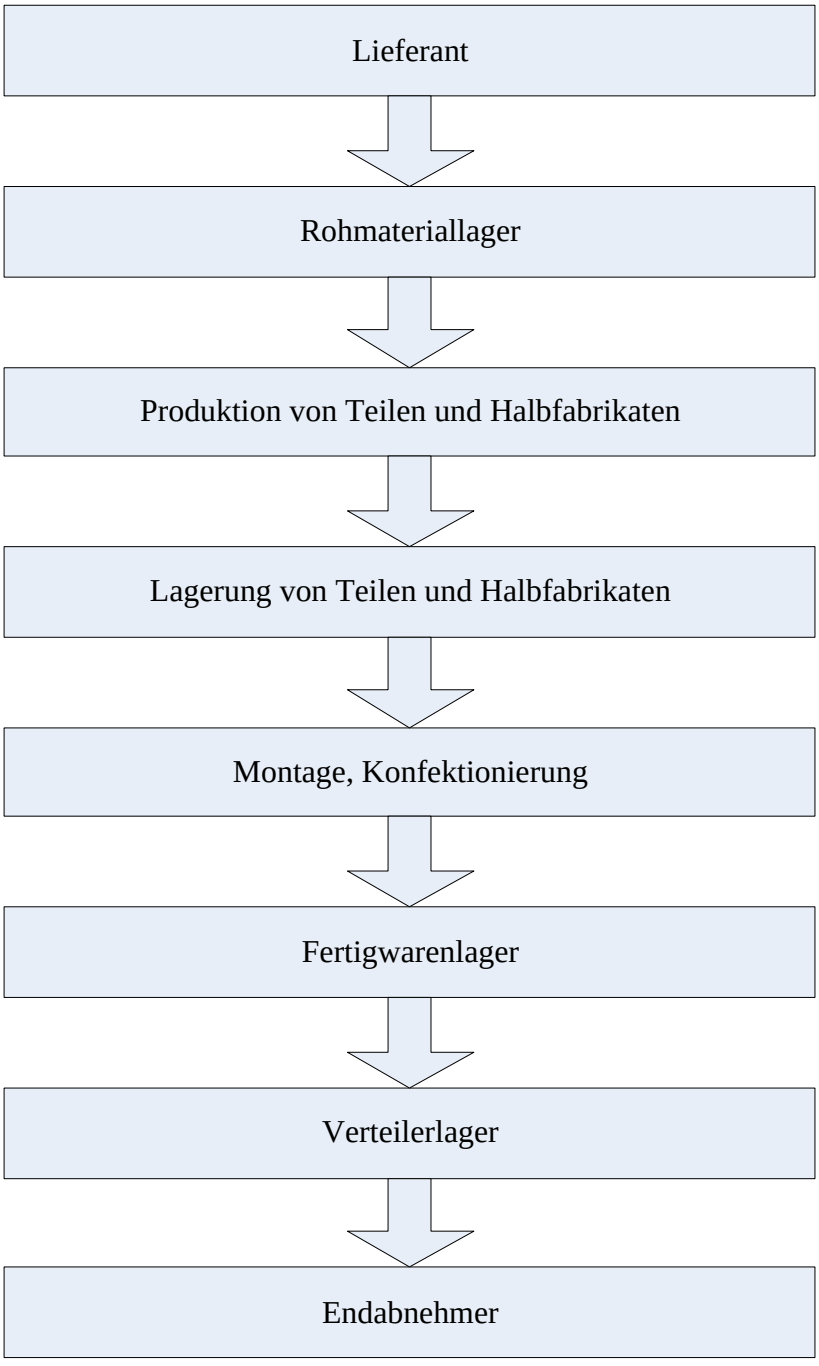

Abbildung D.1.: Logistische Kette nach Schulte (2008)

E. Fachliche Architektur: Planung

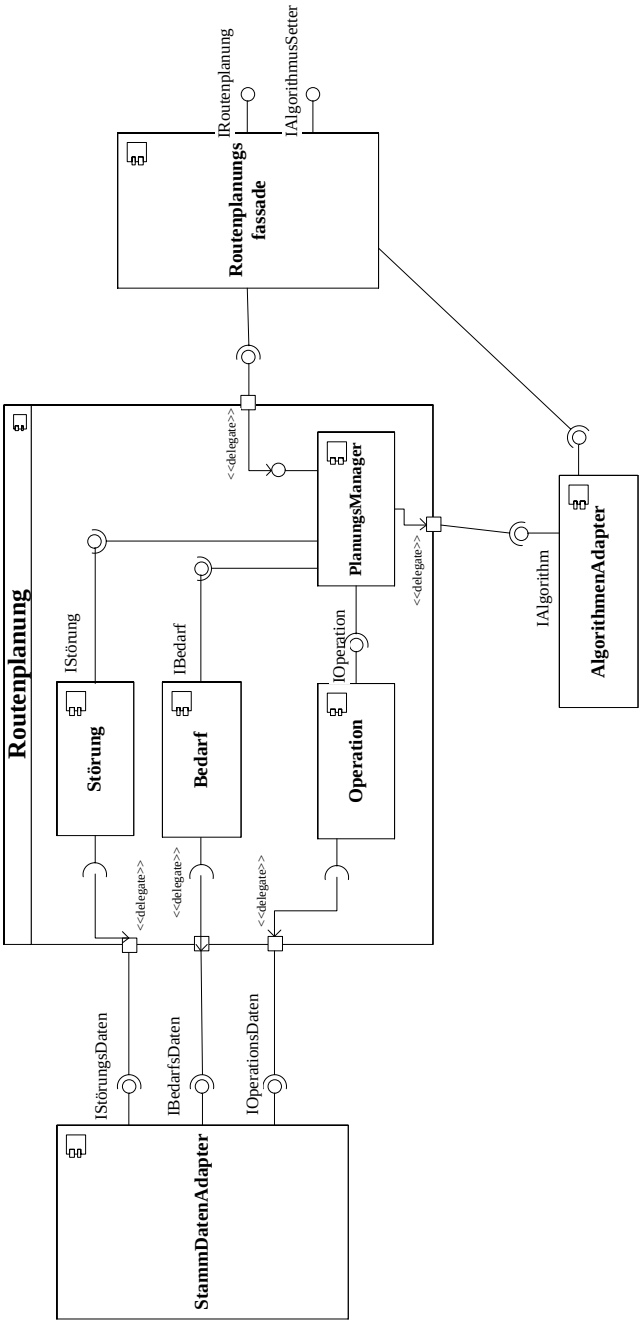

Abbildung E.1.: Fachliche Architektur Routenplanung

Glossar

Apache Thrift
: Ursprünglich von der Firma Facebook Inc. entwickelte Middleware, die inzwischen als open source Komponente im Apache Inkubator aufgenommen ist (http://thrift.apache.org/).

Compute Unified Device Architecture
: Programmiermodell/-architektur, um Graphikkarten der Firma nVidia in handelsüblichen Sprachen (üblicherweise C) ansprechen zu können. Bietet durch die Parallelisierung in modernen Graphikkarten für Algorithmen, die diese nutzen können, nennenswerte Geschwindigkeitsvorteile gegenüber der Ausführung auf einer CPU.

Einsatzgüter
: Der Input eines Produktionsprozesses, also Rohstoffe oder Zwischengüter.

Flugzeugmuster
: In der Luftfahrt gebräuchlicher Begriff für die konkrete Ausprägung eines Flugzeugmodells, z. B. B737-400 als eine konkrete Variante der B737.

Hauptlauf
: Sammelladungsverkehr zwischen einem Sammelpunkt und einem Auflösungspunkt (Schulte, 2008).

Hot Spot
: Erweiterungspunkt in einem Framework, in dem fachspezifisches Wissen eingebracht werden kann. Siehe Kapitel: Grundlagen.

IATA Code
: Eindeutiger alphanumerischer Code, der von der IATA vergeben wird und der Identifizierung von Flughäfen, Fluggesellschaften, Flugzeugtypen und in Ausnahmefällen auch von Bahnhöfen dient.

International Air Transport Association
Industrieverband der Fluggesellschaften, primäres Betätigungsfeld ist die Vereinheitlichung von Prozessen im Betrieb.

Inversion of Control
Paradigma der Programmierung, bei der der Kontrollfluss von einem Framework ausgeht, das Funktionen mit spezifischer Anwendungslogik aufruft. Wird in der Regel über Callbacks realisiert. Umgangssprachlich oft als Hollywood-Prinzip bezeichnet: „Don't call us, we'll call you".

jGraph
Eine Java Bibliothek zum Layouten und Visualisieren von Graphen (http://www.jgraph.com/).

jGraphT
Eine Java Bibliothek, die graphentheoretische Objekte und Algorithmen bereitstellt (http://www.jgrapht.org/).

JSON
Standard Javascript Object Notation (JSON), ein Serialisierungsformat. .

json RPC

> „SON-RPC is lightweight remote procedure call protocol. It is designed to be simple."

(JSON RPC Project, b) .

Lagrangian relaxation
Approximierungsmethode zur Lösung eines Contraintnetzes.

Middleware
Eine in verteilten Systemen logisch zwischen Anwendung/Nutzer und Betriebssystem/Netzwerk angeordnete Software, die vor der Anwendung die Unterschiede zwischen auf verschiedenen Rechnern genutzten Betriebssystemen und Kommunikationseinrichtungen verbergen soll (vgl. Tanenbaum und van Steen, 2007, S. 19, 20).

Nachlauf
Gütertransport vom Auflösungspunkt an einen Empfangspunkt (Schulte, 2008).

Protocol Buffers
Binäres Serialisierungsformat der Firma Google Inc., seit 2008 als open source verfügbar (http://code.google.com/p/protobuf/).

Remote Method Invocation (RMI)

> „RMI ist im Wesentlichen das gleiche wie ein RPC, außer das er auf Objekten anstatt auf Anwendungen operiert."
>
> (Tanenbaum und van Steen, 2007, S. 41).

Remote Procedure Call (RPC)
Das Aufrufen einer Methode auf einer nicht lokalen Ressource, typischerweise in einem Netzwerk.

Representational State Transfer
Architekturstil, der mögliche Aktionen einer API mit den Verben des HTTP-Protokolls (GET, PUT, POST, DELETE) ausdrückt (Fielding, 2000) .

Serialisierung

> „Serialisierung ist eine Abbildung von Objekten auf eine externe sequenzielle Darstellungsform in der Informatik. Serialisierung kann für das Erreichen von Persistenz für ein Objekt verwendet werden, aber auch in verteilten Softwaresystemen spielt Serialisierung eine bedeutende Rolle."
>
> (Wikipedia).

Serious Games

> „Serious games are a result of applying simulation technology to nonentertainment (mostly training) purposes."
>
> (Narayanasamy u. a., 2006).

Tail Number
Eindeutiger alphabetischer Code zur Identifizierung eines Luftfahrzeuges, üblicherweise auf den Seitenleitwerken aufgebracht z. B. „D-ABAD".

Umläufe
Dt. für Rotation. Die Strecke, die ein konkretes Flugzeug in einem Einsatz fliegt. Also z. B. FRA-HAM-FRA-HAM.

Vorlauf
Gütertransport vom Liefer- zum Sammelpunkt (Schulte, 2008).

Weight and Balance

Begriff aus der Luftfahrt. Bezeichnet den Themenkomplex um die zulässige Zuladung eines Flugzeuges und die Verteilung dieser Last in Flugzeugen (Federal Aviation Administration Flight Standards Service, 2007).

Abkürzungsverzeichnis

PDES Parallel Discrete Event Simulation

DES Discrete Event Simulation

PPS Produktionsplanung und -steuerung

IATA International Air Transport Association

GUI Graphische Benutzeroberfläche

CUDA Compute Unified Device Architecture

3LC 3 letter code, Abkürzung für einen flughafenidentifizierenden IATA Code

REST Representational State Transfer

API Application programming interface

DI Dependency Injection

xml Extensible Markup Language

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 21. August 2011	
Ort, Datum	Unterschrift