Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

David Seeger

Simulation des Hamburger U-Bahn-Verkehrs als Basis für Infektionsmodelle

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

David Seeger

Simulation des Hamburger U-Bahn-Verkehrs als Basis für Infektionsmodelle

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

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

Eingereicht am: 19. Dezember 2012

David Seeger

Thema der Arbeit
Simulation des Hamburger U-Bahn-Verkehrs als Basis für Infektionsmodelle

Stichworte
Öffentlicher Nahverkehr, U-Bahn Simulation, ereignisbasierte Simulation

Kurzzusammenfassung
Der öffentliche Nahverkehr ist gerade in Ballungsgebieten wie Hamburg ein potentieller Ort für die Verbreitung von Infektionen. In dieser Arbeit wird eine Simulation des Hamburger U-Bahn-Verkehrs entwickelt, um es anderen Systemen zu ermöglichen, virtuelle Agenten im simulierten Gebiet anhand von auf Anfrage berechneten Wegbeschreibungen zu den jeweiligen Zielen der Agenten zu navigieren. In diesem Zusammenhang wird auch ein System entwickelt, Fahrplandaten und Stationsinformationen aus Webseiten zu extrahieren und auch die simulierten Bahnen und das Kartenmaterial der Region graphisch darzustellen.

David Seeger

Title of the paper
Simulation of the Hamburg subway traffic as a basis for infection models

Keywords
Public short-distance traffic, subway simulation, event driven simulation

Abstract
The public short-distance traffic carries especially in metropolitan areas like Hamburg a big potential for infections to be spread. In this bachelor thesis, a simulaton of the Hamburg subway traffic is being developed to enable other systems to navigate virtual agents in the simulated area using requested route descriptions to lead them to their respective destinations. In this context also a system is developed to extract timetable data and information of stations from websites and also display the simulated trains and maps of the region.

Inhaltsverzeichnis

Listings

1 Einleitung

1.1 Motivation

Seit tausenden von Jahren treiben Menschen von Ort zu Ort oder auch länderübergreifend Handel. Anfänglich reisten die Händler zu Fuß oder auf einem Reittier. Mit sich trugen sie nicht nur ihre Handelswahren, sondern auch Krankheiten, mit denen sie sich auf ihrer Reise infizierten. Ursprünglich von Ratten auf den Menschen übertragen, verbreitete sich die Pest so über weite Teile Europas. (vgl. Tatem u. a., 2006) Heutzutage sind es nicht nur Händler, sondern auch Geschäftsreisende und Urlauber, die aus fernen Ländern Krankheiten mit nach Hause bringen oder ins Ausland tragen. Dort angekommen verbreiten sich die Krankheiten lokal. In Großstädten mit mehreren Millionen Einwohnern finden die Krankheitserreger, dank dichter Besiedelung, auch viele potenzielle Wirte. Ein Ausbreitungsmöglichkeit ist hier, durch die gewollt großflächige Struktur, das öffentliche Nahverkehrssystem.
Im Jahr 2010 transportierte der Hamburger Verkehrsverbund (HVV) über 670 Mio. Fahrgäste. Jährlich wächst diese Zahl. Auch in anderen Städten ist der öffentliche Nahverkehr ein sehr wichtiger Bestandteil der Personenbeförderung. Die Hamburger U-Bahn transportiert täglich viele Menschen auf dem über 100 km langen Streckennetz. (vgl. HVV, 2010) Zur Hauptverkehrszeit und anderen Zeiten hohen Fahrgast Aufkommens, befinden sich teils über 70 Menschen in einem Wagon. Die gegenseitige Interaktion dieser Menschen, ist hierbei unumgänglich.

1.2 Ziel dieser Arbeit

Diese Bachelorarbeit ist Teil einer Masterarbeit von Carsten Noetzel, in der untersucht werden soll, wie sich Krankheiten im Stadtgebiet, mit Hilfe des des öffentlichen Nahverkehrs ausbreiten.

1.2.1 Inhalte der Masterarbeit

In der Masterarbeit wird eine Agentensimulation entwickelt. Diese soll eine große Anzahl von Agenten im Hamburger Stadtgebiet simulieren. Agenten besitzen Wohnorte, gehen ihrer Arbeit nach und besuchen Einkaufszentren und andere Orte. Um ihre Ziele zu erreichen, nutzen die Agenten den öffentlichen Nahverkehr. Hierbei soll untersucht werden, wie sich Krankheiten unter den Agenten auf ihren Wegen ausbreiten.
Agenten sollen die Transportmöglichkeiten nicht nur nutzen, sondern auch mit ihnen interagieren. So nutzen die Agenten Mülleimer in den Wagons, drücken Knöpfe, öffnen Fenster etc.. Bei all diesen Aktionen können sie sich infizieren oder, wenn sie schon infiziert sind, den Gegenstand, den sie berühren.

1.2.2 Aufgabe dieser Bachelorarbeit

Die Aufgabe dieser Bachelorarbeit ist die Erstellung einer Simulation des Hamburger UBahn-Verkehrs. Die Simulation soll es den Agenten ermöglichen ihre Ziele zu erreichen.
Die Aufgabe lässt sich in drei Bereiche aufteilen:
Ein Teil stellt die Sammlung und Aufbereitung der Daten dar. Die Fahrpläne, Verbindungen und die Daten zu den Bahnhöfen sollen möglichst automatisch erfasst werden können. Bei der Aufbereitung der Daten sollen weitere Informationen, wie die geographischen Positionen der Bahnhöfe und Umstiegszeiten, ergänzt werden.
Der zweite Teil ist die Simulation selbst. Die Bahnen sollen anhand der ermittelten Daten dem jeweilige Fahrplan folgen, an Bahnhöfen halten und zur richtigen Zeit die Agenten informieren, damit diese reagieren können, um ein-/aus- oder umzusteigen. Hierzu ist es auch nötig, dass die Agenten informiert werden, wie sie ein Ziel von einem beliebigen Bahnhof erreichen. Weiterhin sollen die Bahnen sich gegenseitig beachten und sich nicht auf einem Gleis überholen oder gleichzeitig den selben Bahnsteig belegen.
Der letzte Teil ist die Visualisierung der Simulation. Dieser Teil soll bei der Benutzung optional sein, um die Leistung für kritische Simulationen nicht zu verringern. In der Visualisierung

soll eine grafische Aufbereitung des momentanen Zustandes der Bahnen und Bahnhöfe auf einer Karte dargestellt werden. Auch soll es Möglichkeiten geben, zusätzliche, von der Agentensimulation zur Verfügung gestellte Inhalte, darzustellen.

1.3 Begriffe

Eisenbahn: "[...] eine Eisenbahn ist ein auf zwei eisernen Scheinen und meistens eigenem Verkehrsweg laufendes, maschinengetriebenes Verkehrsmittel zur Beförderung von Personen und/oder Gütern. Für eine Eisenbahn ist die Zusammenfassung eine größeren Anzahl von Wagen und in der Regel eines Triebfahrzeuges zu einem Eisenbahnzug charakteristisch." (Pachl, 2011)

Untergrundbahn: Für Untergrundbahnen gelten die gleichen gesetzlichen Vorgaben der Straßenbahnen da sie innerstädtisch, ohne Fahrzeugübergang zwischen verschiedenen Netzen operieren. Grundsätzlich gilt hier aber auch die Definition der Eisenbahn. (vgl Pachl, 2011) Im HVV werden hauptsächlich Bahnen vom Typ DT3 und DT4 eingesetzt. DT steht hierbei für Doppeltriebwagen (vgl hochbahn), welche sich am den Enden der Bahn befinden. Häufig werden zwei dieser Bahnen zusammengekoppelt um mehr Fahrgäste transportieren zu können. In dieser gekoppelten Form, wird von den verbundenen Komponenten im Folgenden nur von einer Bahn gesprochen.

Wagon: Ein Wagon für den Personentransport besitzt Sitz- und Stand-Möglichkeiten für Fahrgäste. Wagons können an andere Wagons gekoppelt werden und bieten, je nach Bauweise, den Fahrgästen die Möglichkeit, auch während der Fahrt den Wagon zu wechseln.

Zugfolge: Die Aufeinanderfolge von Zügen auf einer Strecke.

Linie: Eine Linie ist eine Gruppierung von Fahrten verschiedener Fahrzeugen. Die einzelnen Fahrten haben einen großen Teil der Haltestellen gemein, können aber früher eine Endstation anfahren, an einem anderen Bahnhof starten oder sich verzweigen. Die Einteilung wird von den Verkehrsbetrieben vorgenommen.

2 Grundlagen

In den Grundlagen werden nun einige Technologien und Themen beschrieben, die in dieser Arbeit als Grundlage dienen. In Kapitel 2.1 wird der grundlegende Aufbau vom Gleissystemen und technische Hintergründe zu Schienenfahrzeugen erläutert. In Kapitel 2.2 geht es um die Projektion von Geoinformationen. Darauf aufbauend wird in Kapitel 2.3 geschildert, woher Bildmaterial von Karten gesammelt werden kann. Mit dem Thema, wie Geopositionen von Orten gewonnen werden können, setzt sich Kapitel 2.4 auseinander. Um Routen für Personen im Gleissystem zu bestimmen, wird in Kapitel 2.5 der Aufbau des Systems abstrahiert und die Möglichkeit zum Finden von Wegen erläutert.
In den Kapiteln 2.6 und 2.7 werden zuletzt zwei verwendete Technologien zur Auswertung und dateibasierten Beschreibung von Daten näher erklärt.

2.1 Schienenverkehr

Im Schienenverkehr wird der Fahrweg durch den Schienenverlauf bestimmt. Nur durch die Stellung von Weichen kann zwischen verschiedenen Verläufen an festgelegten Orten gewechselt werden. Somit ist das Ausweichen von Fahrzeugen schwierig und bedarf der vorherigen Planung der Fahrwege. Hinzu kommt ein deutlich höherer Bremsweg als z.B. im Straßenverkehr, was auf die Kombination von Stahlrädern auf Stahlschienen zurückzuführen ist. Daraus ergibt sich für den Schienenverkehr ein Haftreibungsbeiwert μ von nur 0,1. Dieser beschreibt das Verhältnis zwischen Reibungskraft und Anpresskraft, in diesem Fall hauptsächlich Gewichtskraft, zwischen zwei Stoffen. Zum Vergleich: Ein Reifen auf Asphalt hat einen etwa acht mal höheren Wert.

$$s = \frac{v^2}{2 \cdot \mu \cdot g} = \frac{(80 : 3{,}6)^2 \,\mathrm{m}^2 \cdot \mathrm{s}^2}{2 \cdot 0{,}1 \cdot 9{,}81 \,\mathrm{m} \cdot \mathrm{s}^2} = 251{,}896 \,\mathrm{m} \tag{2.1}$$

Somit errechnet sich für ein Fahrzeug im Schienenverkehr bei 80km/h ein Bremsweg von etwa 250 Metern, wenn von Anfang an das gesamte Zuggewicht zur Übertragung der Bremskraft ausgenutzt wird. Daher muss die Zugfolge geregelt und gesichert werden. Hierbei

gibt es mehrere Möglichkeiten. In der Praxis wird jedoch hauptsächlich das Fahren im festen Raumabstand verwendet. Hierbei ist die Strecke in Blockabschnitte unterteilt. Die Einfahrt in einen Block wird durch ein Hauptsignal erlaubt oder verboten. In einem Block darf sich nur ein einziger Zug befinden. Da Signale ortsfest sind, ist der Überwachungsbereich immer groß genug gewählt, um den Zug vor seinem Vorgänger zum Stehen zu bringen. Um dieses zu gewährleisten, überlappen sich die Überwachungsbereiche teilweise, sind aber mindestens eine Blockabschnittslänge lang.
Bei Signalstörungen oder im Rangierbetrieb kann es auch notwendig sein, auf Sicht zu fahren. Hierbei werden nur sehr kleine Geschwindigkeiten gewählt, die das rechtzeitige Stehenbleiben vor jedem gesichteten Hindernis erlauben.
Auch wäre, dank technischer Ortungsmöglichkeiten, eine Zugfolge im absoluten Bremsweg möglich. Das heißt, dass der folgende Zug, dank der Ortung des Vorrausfahrenden, immer einen Abstand einhalten kann, der seinem Bremsweg plus einem Sicherheitsabstand entspricht. Wenn auch die Geschwindigkeit der Züge bekannt ist, wäre auch eine Verkürzung des Abstandes durch Berechnung des Bremsweges des vorrausfahrenden Zuges denkbar. Jedoch wird in (Pachl, 2011) davon ausgegangen, dass auch noch die nächsten Jahrzehnte das Fahren im festen Raumabstand verwendet wird. Der Vollständigkeit halber sei auch noch das Fahren im Zeitabstand erwähnt. Hierbei wird, unter Berücksichtigung des Fahrplans und der Möglichkeit des Liegenbleibens eines Zuges, die Zugfolge gewählt. Jedoch handelt es sich bei dieser Betriebsart offensichtlich um ein veraltetes Verfahren und wird in Europa seit dem Ende des 19. Jahrhunderts nicht mehr verwendet. (vgl Pachl, 2011)

Häufig wird eine Strecke nur in eine Richtung befahren. Es gibt jedoch auch Strecken, die in beide Richtungen befahren werden oder sich überlappende Strecken, bei denen die Gleise beider Strecken so dicht beieinander liegen, dass sich zwei Züge hier berühren könnten. Da sich im simulierten Betrieb, zur Vermeidung von Deadlocks, hier NP-harte Probleme ergeben könnten (vgl. Dessouky u. a., 2002), wird dieser Fall im Weiteren nicht mehr betrachtet. Auch wäre im Regelbetrieb der Fahrplan entsprechend angepasst und es käme nur in Sonderfällen zu Komplikationen im kritischen Bereich.

2.2 Kartenprojektion

Um auf die Position von Orten auf der Oberfläche der Erde zu verweisen, können die Geo-Koordinaten verwendet werden. Das zugrunde liegende Koordinatensystem bezieht sich dabei auf ein, meist kugelförmiges Modell der Erde. Die Koordinaten setzten sich aus der geographischen Breite, welche 0° am Äquator und ±90° an den Polen ist und der geographischen Länge von -180° über den Nullmeridian bis +180°, zusammen.
Die wichtigsten Kartendienste im Internet, wie Google Maps und OpenStreetMap , nutzen die Mercator-Projektion um die Oberfläche, der annähernd kugelförmigen Erde, auf eine zweidimensionale Ebene abzubilden. Bei dieser, im 16. Jahrhundert von Gerhard Mercator entwickelten Projektion ,behalten abgebildete Kartenobjekte ihre Form, werden aber zu den Polen hin, in beiden Dimensionen, gestreckt. (vgl. Lorke, 2012)

2.3 Kartendienste

Kartendienste wie Google Maps und OpenStreetMap nutzen umfangreiche Datenbanken, um Geoinformationen zu allen Möglichen Strukturen auf der Erdoberfläche zu speichern. Mit Hilfe von Kachelservern können Auschnitte dieser Informationen in einem Bild gerendert werden. Hierbei wird das Gesamtbild der Welt, je nach geforderter Zoomstufe in unterschiedlich viele Kacheln zerlegt. In der niedrigsten Stufe kann die ganze Welt in einer Kachel gerendert werden. In der nächt höheren werden vier Kacheln verwendet.
Google Maps und OpenStreetMap bieten Möglichkeiten über HTTP-Anfragen gezielt Kacheln als Bild anzufordern. (vgl. Baldowski, 2010)

(a) Eine Kachel (b) Vier Kacheln (c) 16 Kacheln

Abbildung 2.1: Einteilung der Welt in Kacheln, je nach Zoomstufe (OpenStreetMap)

2.4 Geolokalisation

Zusätzlich zu der graphische Aufbereitung lassen sich auch gezielt Anfragen nach den Koordinaten von Orten an die Kartendienste stellen. Google Maps bietet in der API[1] Möglichkeiten nach Adressen zu suchen. Auch die Suche nach Haltestellen ist auf diese Weise im begrenzten Umfang möglich. Die Rückgabe ist eine XML oder JSON Datei mit Informationen zu den gefundenen Orten. Hierbei wird auch angegeben, um welche Art von Ort es sich handelt. Haltestellen sind auf diese Weise leicht zu identifizieren. Da OpenStreetMap alle Geodaten öffentlich zur Verfügung stellt, kann man hier direkt nach Positionen suchen. OpenStreetMap bietet den Download der planet.osm an. Diese über 100GB große Datei enthält alle gesammelten, globalen Informationen von OpenStreetMap . Auch ist es möglich, nur Teile dieser Daten zu extrahieren[2].

2.5 Wegsuche

Das Finden des kürzesten Weges zwischen zwei Orten ist Teil der Graphentheorie. Dieser wird auch häufig minimaler Pfad genannt. Es gibt keinen Pfad zwischen zwei Orten der kürzer ist, aber durchaus mehrere verschiedene kürzeste Pfade. In einem Graph werden die Orte als Knoten bezeichnet. Verbunden werden Knoten durch Kanten. Diese können sowohl gerichtet als auch ungerichtet sein. Gerichtet heißt, dass man von Knoten A über eine Kante zu Knoten B kommt, aber nicht über die selbe Kante von B nach A. Auch können Knoten eine Gewichtung besitzen. Diese wird bei der Pfadsuche verwendet, um zu bestimmen, wie weit zwei Knoten auseinander liegen.
Bei der Wegsuche unterscheidet man zwischen zwei Suchverfahren:
Bei der uninformierten Suche hat der Suchalgorithmus keine Informationen über den Ort des gesuchten Knotens, relativ zu dem Ausgangsknoten. Der angewandte Algorithmus muss also schlimmstenfalls den gesamten Graphen durchsuchen, bis er einen Pfad zwischen den beiden Knoten findet.
Bei der informierten Suche, ist aufgrund von Zusatzinformationen eine lokale Zuordnung der Knoten möglich. Ein Vertreter dieser Algorithmen ist der A* Algorithmus. Dieser wählt den nächsten zu durchsuchenden Knoten anhand der bisher bekannten Pfadlänge zum Ausgangsknoten und einem Schätzwert, welcher eine mögliche Entfernung zum Zielknoten angibt. Je nach Aufbau des Graphen kann der Algorithmus hier in vergleichsweise wenig Schritten den Pfad zum Ziel bestimmen. Im schlechtesten Fall muss er aber auch den ganzen Graphen

[1] Google API-Beschreibung: https://developers.google.com/maps/?hl=de
[2] Links und Informationen zu den OpenStreetMap Daten: http://wiki.openstreetmap.org/wiki/Planet.osm

durchsuchen. (vgl. Klauck und Maas, 1999)

2.6 LINQ

LINQ steht für Language Integrated Query und wurde in C# mit ab der Version 3.0 eingeführt und ist eine Sprachergänzung für das .NET Framework ab der Version 3.5. LINQ vereinheitlicht den Zugriff auf unterschiedliche Datenquellen mittels einer an SQL angelehnten Syntax.
LINQ to Objekts bildet hierbei die Grundlage. Kollektionen und Objekte lassen sich mittels LINQ manipulieren und untereinander in Beziehung setzten. LINQ to XML, LINQ to SQL und LINQ to ADO.NET sind die Standard Provider für die entsprechende Datenbasis und zeigen die Vielseitigkeit der Datenquellen von LINQ.
Ein großer Vorteil von LINQ ist, dass Abfragen zur Compilezeit auf Fehler untersucht werden können und diese nicht erst zur Laufzeit auftreten. Auch lässt sich durch die Abstraktion der Datenstruktur die Datenquelle leicht austauschen und bleibt unabhängig von der verwendeten Technologie. (vgl. Kühnel, 2010)

2.7 Extensible Markup Language

Die Extensible Markup Language oder Kurzschreibweise XML ist eine Auszeichnungssprache zur Darstellung von Daten in einer hierarchischen Struktur. Ein Element stellt die wichtigste Struktureinheit eines XML-Dokumentes dar. Elemente können weitere Elemente oder einen Wert beinhalten und beliebige Namen besitzen. Auch kann ein Element sogenannte Attribute mit beliebigen Namen besitzen. Attribute haben nur einen Wert und können keine weiteren Attribute oder Elemente beinhalten.
Der Aufbau eines Elementes ist wie folgt:

```
<Name attrubut1="..." attribut2="..." ...>
        Weitere Elemente
<\Name>
```

Wenn keine weiteren Elemente in einem Element vorhanden sind, gibt es auch eine kurz Schreibweise:

```
 <Name attrubut1="..." attribut2="..." ... \>
```

Die Notation in HTML der sogenannten HTML-Tags ist ähnlich. Jedoch gibt es im Gegensatz zu XML nur eine beschränkte Anzahl von Tag-Namen. Auch müssen XML-Dokumente wohlgeformt sein. Das heißt, dass gewisse Regeln bei dem Aufbau befolgt werden. So darf es genau ein Wurzelelement geben. Elemente müssen, wie in den obigen Beispielen, in der richtigen Reihenfolge geschlossen werden und Attribute in einem Element besitzen einen eindeutigen Namen. Die Regeln für ein HTML Dokument sind hier weniger strikt.
(vgl. Vonhoegen, 2007)

3 Stand der Technik

Das Thema der Masterarbeit ist laut dieser eine bislang wenig erforschte Thematik, daher ist auch das Thema, nämlich die Darstellung und zur Verfügungsstellung eines simulierten Nahverkehrsbetriebes für virtuelle Agenten, dieser Bachelorarbeit sehr speziell. Im folgenden werden einige Veröffentlichungen und Anwendungen erläutert, die Ähnlichkeiten mit diesem Thema dieser Bachelorarbeit aufweisen.
(Dessouky u. a., 2004) beschreibt ein Modell zur Simulation eines komplexen Schienennetzes. Hier wird das Ziel verfolgt, Züge möglichst effizient durch ein Schienennetz zu navigieren. Dem Modell liegt ein Graph zugrunde, welcher Bahnhöfe, Weichen und Verbindungen abbildet (Abb. 3.1). An jedem Knoten wird durch eine Heuristik entschieden, welchen weiteren Pfad der Zug einschlägt. Die Simulation findet anhand eines „Event calendar“ statt. Hier werden die geplanten Ereignisse, wie das Erreichen eines Bahnhofs oder einer Weiche eingetragen, um sie zu gegebener Zeit auszuwerten. Durch die Vermeidung von unsicheren Zuständen wird hier auch auf komplexen Strecken eine deadlock-freie Fahrt für die Züge ermöglicht.

In (Kanacilo und Verbraeck, 2007) wird ein System zur Planung von Strassenbahn Fahrten beschrieben. Das Hauptaugenmerk liegt hierbei auf dem Versuch, möglichst alle Aspekte wie Fahrplan Zuweisungen, die Ausbreitung von Verspätungen und Verkehrsanalyse gemeinsam zu betrachten, da Wechselwirkung bei der Optimierung einzelner Aspekte auftreten können. Das Schienensystem wird hier zur Sicherung des Fahrbetriebes in Blöcke aufgeteilt. Die Blöcke werden durch Signale voneinander getrennt. Fahrzeuge melden sich bei Signalen an um von diesen über Zustandsänderungen benachrichtigt zu werden. Verlässt ein Fahrzeug den Wirkungsbereich eines Signals, meldet es sich von diesem wieder ab. Auch nehmen die Fahrzeuge, je nach Priorität, Einfluss auf das Signalsystem und können dieses, sofern der Weg frei ist, einen Wechsel der Signalphase anfordern. Fahrzeuge mit gleicher Priorität werden in der Reihenfolge ihrer Anfrage bearbeitet.
Die Fallstudie dieses Systems betrachtet aber nur den geplanten Ausbau eines existierenden Systems.

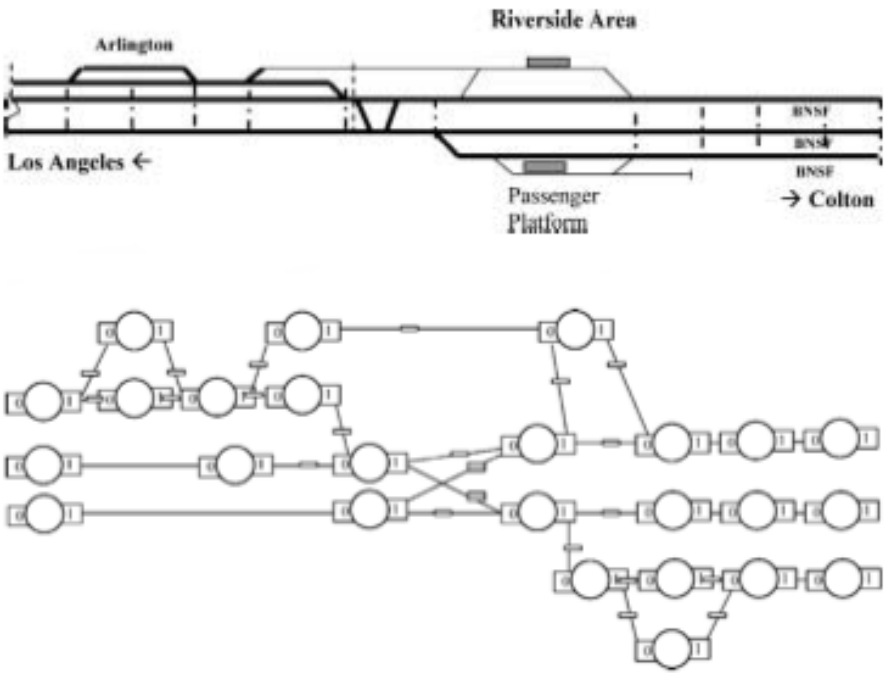

Abbildung 3.1: Abbildung einer Strecke auf den Graphen von (Dessouky u. a., 2004)

Eine andere Richtung verfolgen Programme, in denen der Benutzer selbst versucht, das System oder Teile davon zu optimieren. Bei diesen, auch als Simulationsspiele bekannten Programmen, liegt jedoch hauptsächlich der Unterhaltungswert im Fokus. Wobei z.B. einige Bahnsimulationen, in denen der Spieler die Rolle des Zugführers übernimmt, subjektiv sehr realistisch wirken. Ähnlicher im Thema sind jedoch Simulationen größerer Areale, in denen der Spieler keine direkte Kontrolle über die Fahrzeuge hat.
Hier wäre als Beispiel das Spiel „Cities in Motion“ von Paradox Interactive aus dem Jahre 2011 zu nennen. In dieser Simulation hat der Spieler die Möglichkeit, den öffentlichen Nahverkehr einer kleinen, meist an reale Städte angelehnte Stadt zu bauen und zu verwalten. Ein weiterer interessanter Aspekt dieses Spiels sind die simulierten Bürger, welche einen Wohnort besitzen und von diesem, unter Zuhilfenahme der vom Spieler gebotenen Transportmöglichkeiten ihren Arbeitsplatz oder Einkaufsmöglichkeiten etc. erreichen können. Jedoch verfolgen die Züge hier keinen Fahrplan mit Zeiteinträgen und fahren nur ihre ihnen vorgegebenen Haltestellen

schnellstmöglich ab.

Abbildung 3.2: Cities in Motion

Ein weiteres Beispiel ist Transport Tycoon von Microprose aus dem Jahre 1994 bzw. die Weiterentwicklung als Open Source Projekt unter dem Namen „Open Transport Tycoon Deluxe“ (Open TTD). Hier kann, dank eines sehr umfangreichen Signalsystems die zuvor beschriebene Blockaufteilung der Gleisanlage gut umgesetzt werden.

Es gibt auch einige Projekte, die aktuelle oder allgemeine Fahrplandaten der Verkehrsunternehmen nutzen, um die (berechnete) Position von den Fahrzeugen zu ermitteln. Das Projekt „Live train map for the London Underground“ (Quelle londonubahn), welches nach eigenen Angaben in nur wenigen Stunden entstanden ist, zeigt die aktuelle Position der Londoner Untergrundbahnen auf einer Karte an. Die Position der Bahnen berechnet sich hierbei aus den Positionen der Haltestellen und der seit der Abfahrt vergangen Zeit, seit der Abfahrt. Die Daten werden direkt aus einer, von der Londoner Untergrundbahn zur Verfügung gestellten, API abgerufen.

Ähnlich zu dieser Webseite gibt es auch in andere Seiten dieser Art. Auf (Quelle munchenbahn) ist die Bewegung der S-Bahn von München inklusive Verspätungen dargestellt und auf (Quelle swisstrain) werden, laut Seitenbeschreibung, fast alle Züge der gesamten Schweiz angezeigt.
Die beiden Spiele, besonders Cities in Motion, zeigen interessante Ansätze in der Beförderung von Agenten. Auch das Signalsystem von OpenTTD bietet viele Möglichkeiten zur realitätsnahen Simulation.
Die live Karten im Internet sind teilweise gute Abbildungen des tatsächlichen Zustandes

Abbildung 3.3: Open Transport Tycoon Deluxe. Bild von openttd.org

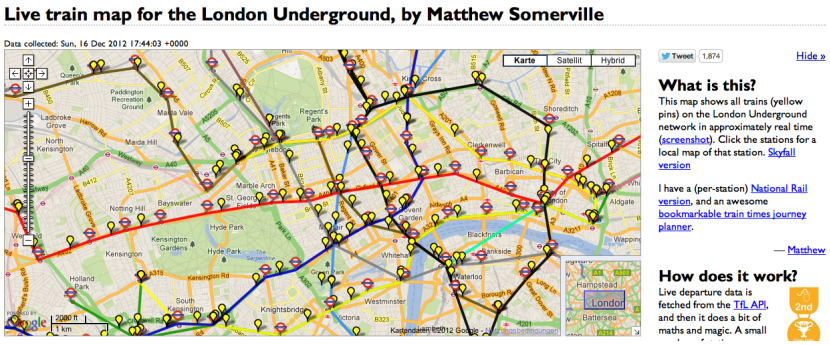

Abbildung 3.4: Live train map for the London Underground

des Betriebs. Jedoch fehlen hier die Interaktionsmöglichkeit mit Agenten gänzlich und die Interaktion zwischen den Fahrzeugen ist hier nicht gegeben, da es sich in diesem Fall entweder nur um eine Abbildung der tatsächlichen Fahrzeugpositionen oder um eine strikte Befolgung des Fahrplans handelt.

4 Analyse

4.1 Anforderungen

Aus der Masterarbeit ergaben sich folgende funktionale Anforderungen an diese Bachelorarbeit:

Simulation

FA1 Die Züge fahren möglichst anhand ihres Fahrplans und halten an den Bahnhöfen.

FA2 Züge befahren nur freie Streckenabschnitte, also Abschnitte der Strecke, die von keinem anderen Zug belegt sind.

FA3 Züge warten, bis alle Passagiere eingestiegen sind oder eine maximale Wartezeit erreicht wird.

FA4 Züge benachrichtigen Bahnhöfe über ihre Ankunft in der Station. Bei der Ankunft werden folgende Informationen vom Zug an die Station mitgeteilt: Ziel, Art des Zuges

FA5 Züge benachrichtigen Bahnhöfe über das Schließen der Türen.

FA6 Züge benachrichtigen Bahnhöfe über das Verlassen des Bahnhofs.

FA7 Züge informieren Passagiere in den Wagons über die Ankunft an einem Bahnhof. Die Benachrichtigung enthält folgende Informationen: Name des Bahnhofs.

FA8 Züge informieren Passagiere über den Halt an einem Signal.

FA9 Züge informieren Passagiere über die Weiterfahrt nach dem Halt an einem Signal.

FA10 Es können auf Anfrage Wegbeschreibungen von einer Station zu einer anderen Erzeugt werden.

Visualisierung

FA1 Züge und Haltestellen sollen auf einer Karte dargestellt werden.

FA2 Züge sollen auf der Karte anklickbar sein, um Informationen zum Zug anzuzeigen. Folgende Informationen werden dargestellt: Ziel des Zuges, Typ des Zuges, Anzahl der Agenten im Zug pro Infektionsstadium

FA3 Bahnhöfe sollen auf der Karte anwählbar sein um Informationen zum Bahnhof anzuzeigen. Beim Klick auf einen Bahnhof werden diese in der Oberfläche mit den Zügen und Agenten dargestellt. Die Visualisierung der Station wird von Carsten Noetzel bereitgestellt.

Dieses waren die Grundanforderungen. Da sich die Masterarbeit noch im Aufbau befand, kamen später weitere Anforderungen hinzu, bzw. vorhandene wurden abgeändert.

EFA1 Es können auf Anfrage Wegbeschreibungen von einer Position auf der Karte zu einer anderen erzeugt werden.

EFA2 Es sollen frei definierbare Orte und Regionen auf der Karte angezeigt werden können.

EFA3 Orte und Regionen sollen sich auf Ebenen befinden die sich ein- und ausblenden lassen.

EFA4 Orte und Regionen sollen mit der Maus anwählbar sein und entsprechende Nachrichten bei einer Anwahlaktion versenden.

Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen beziehen sich nicht auf eine spezielle Funktion, sondern meist auf das Verhalten des Gesamtsystems. (vgl. Sarstedt, 2009)
Im folgenden werden nun die nichtfunktionalen Anforderungen an das zu entwickelnde System beschrieben.

Benutzerfreundlichkeit:
Die Simulation und die beteiligten Systeme richten sich in erster Linie an die Benutzer der Agentensimulation. So wird dieses System nicht für eine breite Masse an Benutzern entwickelt und kann ein gewisses Maß an technischem Verständnis voraussetzten. Dennoch sollte die gebotene Interaktion möglichst einfach und klar strukturiert sein.

Fehlertoleranz:
Sofern Eingaben vom Benutzer erwartet werden, sollten diese überprüft werden und im Fehlerfall eine Nachricht angezeigt werden. Das System sollte bei fehlerhaften Eingaben nicht abstürzen oder diese gar weiterverwenden.
Auch bei der Kommunikation mit externen Systemen ist darauf zu achten, dass übergebene Parameter überprüft werden.

Portierbarkeit:
Diese Anforderung kann stark durch die Wahl der Programmiersprache beeinflusst werden. Hier sollte die Wahl auf eine plattformunabhängige Sprache fallen. Aber auch die Wahl von verwendeten Bibliotheken kann schon zu Einschränkung der Portierbarkeit führen. Diese sollten auch möglichst plattformunabhängig oder, wenn sie z.B. hardwarenahe Funktionen nutzen, für viele Plattformen verfügbar sein.

Performance:
Gerade bei einer Simulation die Ereignisse untersucht welche sich über einen längeren Zeitraum erstrecken ist es wichtig, dass die Berechnung schneller als in Echtzeit abläuft. Konkret wird hier gefordert, dass die Simulation des Bahnbetriebes eines ganzen Tages nicht länger als eine Minute dauert.

Wartbarkeit/Erweiterbarkeit:
Durch Modularisierung der Softwarekomponenten können diese leichter ausgetauscht, gewartet und erweitert werden. Dieses wird, gerade im späteren Verlauf, zusätzlich durch eine gute Dokumentation erleichtert.

5 Entwurf

Beim Entwurf werden die aus der Analyse erhaltenden Informationen ausgearbeitet. Auch werden hier die gewonnen Erkenntnisse aus der vorherigen Betrachtung, von Projekten in selben Themengebiet, für die Entscheidungen zum Entwurf Verwendung finden.
Die in der Analyse beschriebene Teilung der Aufgaben (Simulation, Visualisierung und Datenbeschaffung) sollen auch bei dem Entwurf beibehalten werden. Das Sammeln der Daten muss nur einmal stattfinden. Danach steht der Fahrplan für sämtliche Bahnen fest und muss während der Simulation nicht mehr geändert werden. Daher wird die Aufgabe der Datenbeschaffung in ein eigenes, von der Simulation unabhängiges Programm verlagert.
Der Austausch der Informationen kann über verschiedene Wege erfolgen. Hier wurde die Speicherung der Daten als XML-Datei gewählt. XML-Dateien haben den Vorteil ihre Daten Klartext abzuspeichern. Dieser Vorteil verliert sich zwar mit wachsender Datenmenge recht schnell, jedoch lassen sich die Daten des Verkehrsnetzes gut skalieren, da zur Überprüfung und Fehlerfindung Ausschnitte reichen.
Die Visualisierung als eigenes Modul zu entwickeln bringt den Vorteil, dass die Verwendung optional ist. Im späteren Verlauf der Masterarbeit ist geplant, den Verlauf der Krankheitsausbreitung zu untersuchen. Dieses ist ein Prozess, der sich über einen längeren Zeitraum entwickelt. Um schnell Ergebnisse zu erhalten, wäre hier eine Computerressourcen verbrauchende, graphische Darstellung hinderlich.
Eine große Abhängigkeit zur Simulation besteht jedoch. Die Entitäten der Simulation müssen dargestellt werden. Hierzu sind tiefere Informationen über den Aufbau des simulierten Verkehrs von Nöten.
Im Folgenden wird nun auf die Entscheidungen eingegangen, die dem Entwurf der Simulationskomponente vorangegangen sind.

5.1 Designentscheidungen

Einer auf dem Computer erstellten Simulation liegt meist ein Modell der Realität zugrunde. Dieses ist immer eine Vereinfachung und unterliegt einer beschränkten Sicht. Es müssen die wesentlichen Bestandteile herausgearbeitet werden, um das Modell realitätsnah erscheinen zu

lassen, aber nicht zu komplex, um Rechenzeit zu sparen.
Im Folgenden wird deshalb nun beschrieben, welche Entscheidungen zum Entwurf beigetragen haben.

5.1.1 Simulationsart

Die Simulation eines Verkehrsbetriebes sollte alle Ereignisse möglichst in einer zeitlichen korrekten Reihenfolge abarbeiten, da verschiedene Ereignisse voneinander abhängig sein können. Die Simulationsart beschreibt, in welcher Form diese Abarbeitung stattfindet und wie das Verhalten von Objekten in der Simulation simuliert wird. Dabei gibt es folgende Möglichkeiten der Simulation:

Kontinuierliche Simulation:
Hierbei wird in möglichst kleinen Zeitabständen die Simulation aktualisiert. Die physikalischen Vorgänge müssen durch Differentialgleichungen über die Zeit berechnet werden. Konkret heißt das, dass wenn ein simulierter Zug mit konstanter Geschwindigkeit, nach der Gleichung $s = v \cdot \Delta t$, wobei s die gefahrene Strecke, v die Geschwindigkeit und Δt der Abstand der momentanen Zeit zu der zuletzt simulierten Zeit ist, seine Position aktualisieren muss. Sollte der Zug in dieser Zeit zusätzlich eine Beschleunigung erfahren, müsste die Berechnung deutlich erweitert werden. Wenn die Zeitabstände klein genug sind, ließe sich die Beschleunigung auch zu sprunghaften aber sehr kleinen Geschwindigkeitsveränderungen vereinfachen.
Jedoch ist diese Art der Simulation recht aufwändig, da jeder Zug in kurzen Abständen neu berechnet wird, auch wenn ihm keine Zustandsänderung widerfährt und seine momentane Position für einen eventuellen Betrachter oder simulierten Fahrgast unerheblich ist.
Eine sehr ähnliche Simulationsmethode wären z.B. zelluläre Automaten. Diese werden zur Simulation von Autoverkehr, insbesondere der Staubildung genutzt. Hierbei wird die Strecke, in feste Blöcke gleicher Größe unterteilt. Ein Block kann höchstens von einem Fahrzeug besetzt sein. Ein Fahrzeug legt, je nach seiner Geschwindigkeit, eine gewisse Anzahl von Blöcken, sofern diese frei sind, pro Zeiteinheit zurück. (vgl. Nagel und Schreckenberg, 1992) Das entspräche einer Fahrzeugfolge im absoluten Bremsweg. Hier wären Abwandlungen denkbar, welche die Blöcke auf die Blockabschnitte im Schienenverkehr abbilden. Jedoch ist hier die feste Größe, da dieses unrealistisch ist, ein Problem. Sollte diese variabel gestaltet werden, sind keine festen Zeiteinheiten in der Simulation mehr möglich. Daher eignen sich zelluläre Automaten ohne starke Modifikation nur bedingt für den Schienenverkehr unter der Nutzung von Blockabschnitten

Ereignisbasierte Simulation:
Bei dieser Simulationsart werden alle zukünftig auftretenden Ereignisse, wie das Abfahren oder die Ankunft eines Fahrzeuges, in eine zeitlich sortierte Liste eingetragen. Das kleinste Element ist jenes, welches als nächstes bearbeitet werden muss, also zeitlich gesehen am dichtesten an der momentan simulierten Zeit liegt. Hierbei wird vorausgesetzt, dass der Simulator genaue Kenntnisse über den Verlauf der Simulation bis zum nächsten Ereignis hat. Im Falle eines Zuges muss also bei der Abfahrt schon klar sein, wann er ankommt bzw. was bis zum nächsten Ereignis passieren wird. In der Theorie ist dieses auch berechenbar, denn das Beschleunigungsverhalten des Zuges, der Streckenverlauf und die zulässigen Geschwindigkeiten auf der Strecke sind bekannt.
Da die Strecke zwischen zwei Signalen nur von einem Zug zur gleichen Zeit genutzt werden darf, muss zwischen den Signalen auch nicht auf behindernde Einflüsse geprüft werden. Jedoch ist eine außerplanmäßige Störung des Zuges aufwändiger, da hier das nächste Ereignis des Zuges aus der Liste der Ereignisse entfernt werden muss, die aktuelle Position des Zuges ermittelt werden muss und schließlich auf ein Ereignis gewartet werden muss, welches den Zug wieder seinen Betrieb aufnehmen lässt. Für folgende Züge stellt diese Störung auch ein Problem dar, da sie unerwartet am nächsten Signal anhalten müssen. Hier kann der Zug nicht wissen, ob er bei der Abfahrt am nächsten Signal halten muss.

Nach Abwägung der genannten Vor- und Nachteile wurde die ereignisbasierte Simulation als Simulationsart festgelegt, da sie vergleichsweise Rechenzeit sparend ist und die Abfolge der Nachrichten sequenziell und die gegenseitige Benachrichtigung der Agenten in kritischen Zeiten, wie beim Umsteigen zwischen Zügen, sichergestellt ist.

5.1.2 Abbildung des Nahverkehrssystems

Da die Aufteilung in Blockabschnitte, wie in den Grundlagen beschrieben, die größte Verbreitung hat, wird sie dieser Bachelorarbeit zugrunde gelegt. Der wirkliche Aufbau unterliegt den Verkehrsbetrieben. Die Abfahrtzeiten der Fahrzeuge sind meist allgemein zugänglich. Die Position der Signale, die Geschwindigkeitsbegrenzungen, die gemeinsam genutzten Streckenabschnitte zwischen Stationen und der genaue Verlauf der Strecken sind es hingegen nicht. In dieser Bachelorarbeit wird davon ausgegangen, dass sich die Züge im Regelbetrieb nicht gegenseitig behindern.
Aber selbst die zugänglichen Abfahrtzeiten sind nur ungenau zu bekommen. Diese sind meist im Minuten Takt angegeben. Eine kleinere Einteilung wäre auch, aufgrund von leichten Abwei-

chungen der Halte- bzw. der Fahrzeiten, nicht sinnvoll. Jedoch liegen in dichteren Stadtgebieten Haltestellen nur einige hundert Meter auseinander. Die Fahrzeit beträge also Beispielhaft von Station A zu Station B nur 30 Sekunden, von Station B zu Station C jedoch 1 Minute und 30 Sekunden. Der Fahrplan gibt jedoch jeweils eine Minute Unterschied für die Abfahrtzeiten an. Hier könnte man über die Abstände der Stationen argumentieren und die simulierten Fahrpläne dementsprechend anpassen. Jedoch fahren einige Züge gleichzeitig Stationen an und warten ggf. aufeinander, um den Fahrgästen ein möglichst zeitnahes Umsteigen zu ermöglichen. Die Haltezeiten von Zügen im HVV betragen aber nur einige Sekunden. Hier müsste bekannt sein, welcher Zug auf welchen wartet und auch ob es eine maximale Wartezeit gibt die ein Zug auf einen anderen wartet. Auch könnte planmäßig ein Zug auf einer längeren Strecke schneller fahren als auf einer kürzeren, um den Fahrplan zu erfüllen. Ein Zusammentragen all dieser Informationen würde den Umfang dieser Bachelorarbeit bei weitem sprengen. Daher wird von den angegebenen Fahrzeiten ausgegangen. Sollte späteres Wissen hinzukommen, kann dieses natürlich in den erstellten Fahrplan eingearbeitet werden.
Die erstellte Simulation nimmt eine Aufteilung der Strecken in Blockabschnitten an, wobei die gesamte Strecke zwischen zwei Stationen auch exklusiv für den Verkehr zwischen diesen Stationen genutzt wird und auch nur die Benutzung in eine Richtung zulässig ist. Zwischen zwei Stationen befinden sich demnach häufig zwei Strecken mit unterschiedlichen Richtungen. Die Blockabschnitte werden von einem einfahrenden Zug reserviert und Signale signalisieren folgenden Zügen die Erlaubnis, in den Block einzufahren. Auch wird von einer konstanten Geschwindigkeit zwischen zwei Stationen ausgegangen. Bei einem Halt an einem Signal kann somit errechnet werden, wie viel der Zug von der Gesamtstrecke zurückgelegt hat und wie lange das Erreichen des nächsten Signals bzw. der Haltestelle dauern würde.

5.1.3 Fahrplandaten

Der erste Schritt vor der eigentlichen Simulation ist das Zusammentragen der Fahrplandaten. Der HVV gibt seinen Kunden zwei verschiedene, öffentlich zugängliche Darstellungen zu den Fahrplänen:
Der Haltestellenaushang ist an jeder Haltestelle vorhanden und gibt die Abfahrtzeiten von Fahrzeugen an. Es ist nicht direkt zu erkennen, welcher spezielle Zug sich wann und wo befindet. Jedoch gibt es meist auch einen Aushang, wie lange ein Fahrzeug zu den folgenden Haltestellen unterwegs ist. Aus diesen Daten lässt sich die Fahrt jedes Zuges ermitteln.
Eine weitere Darstellung ist der Linienfahrplan. Hier wird die komplette Zugfahrt jedes Zuges aufgelistet, wobei hier jeder Haltestelle auf der Fahrt eine Abfahrtszeit zugeordnet wird. An

Haltestellen mit verlängerten Haltezeiten ist auch eine Ankunftszeit eingetragen. Der Linienfahrplan bietet daher eine detailliertere Darstellung des eigentlichen Ablaufes.
Aus diesen Daten soll hier der Fahrplan generiert werden. Als Grundlage wird der Linienfahrplan verwendet. Auf der Webseite des HVVs kann dieser für jede Linie angezeigt werden. Jedoch ist das zusammentragen dieser Daten aufwändig, da für eine Linie jeweils nur acht Fahrten angezeigt werden. Weitere Intervalle können über eine Dropdownbox gewählt werden. Auch werden viele Scripte und Session-Informationen auf der Seite verwendet. Daher ist nicht an der URL erkennbar bzw. beeinflussbar, was angezeigt wird. Eine einfachere Möglichkeit an diese Daten zukommen, bieten die Seiten von Geofox welcher auch als Dienstleister für die Fahrplanauskünfte des HVV eingesetzt wird.Über die Fahrplanauskunft (Quelle geofoxline) ist für jede Linie der komplette Linienfahrplan einsehbar und bietet somit leichter auswertbare Daten als die Seite des HVVs. Der Fahrplan wird hier in einer in HTML kodierten Tabelle angezeigt. Vertikal am linken Rand sind die Stationen in der Anfahrtsreihenfolge aufgelistet. Jeweils daneben, in der gleichen Reihe, befinden sich die Abfahrts- bzw. Ankunftszeiten. Sollte die Station nicht angefahren werden, ist das Zeitfeld leer. Wird die Station übersprungen, was im Falle einer Verzweigung möglich ist, wird dieses mit dem Pipezeichen (|) gekennzeichnet. Eine Spalte mit Zeiten enthält die gesamte Fahrt für eine Bahn.
Bei der Ausführung der Fahrplandaten sind weitere Informationen von Bedeutung, welche nicht explizit im Fahrplan angegeben sind. Bahnen in Verkehrsbetrieben können nach Abschluss ihres Fahrplans genutzt werden um die Linie in Gegenrichtung zu befahren, den Fahrplan einer andere Linie zu folgen oder aber auch an bestimmten Stellen geparkt oder gewartet zu werden um auf den nächsten Einsatz zu warten. Aber es ist aus dem Fahrplan nicht bestimmbar, woher eine Bahn kommt. Daher wird in der Simulation an der ersten Station im Fahrplan eine neue Bahn erzeugt. Hier wäre es natürlich auch möglich, Bahnen in der Nähe einer Station zu belassen und sie auf einen weiteren Einsatz an selbiger warten zu lassen. Hierbei könnte jedoch ein Ungleichgewicht von wartenden Bahnen entstehen und einige müssten verlegt werden. Da diese Simulation primär für die Unterstützung von Systemen zur untersuchung von Krankheitsausbreitungen entwickelt wird und nicht bekannt ist, wann und wo Bahnen gereinigt werden, ist hier ein Kompromiss unumgänglich und es wird bei jeder neuen Fahrt von einer sauberen Bahn ausgegangen.
In den Angaben im Fahrplan des HVVs sind nur längere Haltezeiten einer Fahrt angegeben. Im Betrieb wird ein Bahnführer darüber entscheiden, wie lange eine Bahn an einer Haltestelle hält. Dieses kann von mehreren Faktoren wie Fahrgastaufkommen und Verspätung der Bahn abhängen. Das Fahrgastaufkommen kann von der Simulation nicht berücksichtigt werden, da sie keine Informationen zu den Agenten hat. Auch bei Verspätung muss den Passagie-

ren ausreichend Zeit zum Ein- und Aussteigen bleiben und ist dann wieder abhängig vom Fahrgastaufkommen. Daher wird an Haltestellen, an denen keine Haltezeit angegeben ist, ein einstellbarer Standardwert verwendet.
Auch gibt es keine Angaben zur Länge einer Bahn und es besteht die Möglichkeit, dass sich Bahnen aufteilen oder zusammensetzen. Aber auch hierzu liegen keine Informationen vor und kann hier nicht berücksichtigt werden.
Eine Anforderung stellt die Wegsuche dar. Hierbei ist es wichtig zu wissen, wie viel Zeit eine Person beim Wechsel zu einem anderen Bahnsteig benötigt, um Gehwege zu minimieren und auch die Möglichkeit des Verpassens einer Bahn zu berücksichtigen. Diese Zeiten sind von Bahnhof zu Bahnhof unterschiedlich. Hierbei sind nur Bahnhöfe von Interesse, an denen ein Wechsel sinnvoll ist, also den Fahrgast nicht wieder zu der Station zurück bringt, von der er gekommen ist. Dieses schränkt die Anzahl der zu untersuchenden Bahnhöfe im HVV stark ein. Die Übergangszeiten sind von Person zu Person unterschiedlich und dienen nur als Richtwert. Daher ist die Genauigkeit zu vernachlässigen und es können hier Schätzwerte, die sich aus der Wegbeschreibung des HVVs bzw. des Geofox Dienstes ergeben, gewählt werden.
Aus dem Fahrplan geht lediglich der Name der Bahnhöfe hervor. Um die Bahnhöfe auf einer Karte darzustellen, müssen die Geopositionen bekannt sein. Um die Arbeit hier gering zu halten, soll hier ein externer Kartendienst die nötigen Daten liefern.

```
<Network>
 <StationList>
   <Station position="53.6506158 10.1631689"
     platformCount="2" name="Volksdorf">
     <Transit from="1" to="2" duration="0:00:01" />
   </Station>
   <Station position="53.6648322 10.1554871"
     platformCount="2" name="Buckhorn">
     <Transit from="1" to="2" duration="0:02:00" />
   </Station>
 </StationList>
 <ConnectionList>
   <Connection from="Volksdorf" to="Buckhorn" id="1" />
 </ConnectionList>
 <LineList>
   <Line name="U1" direction="1" version="1">
     <TimeTable dayFilter="1">
       <Drive>
```

```
19          <Stop time="0:18:00"
20            holdTime="0:00:10" platform="1" connection="1" />
21        </Drive>
22      </TimeTable>
23    </Line>
24  </LineList>
```

Listing 5.1: Aufbau der XML Datei

In 5.1 ist der Entwurf für die Datenstruktur der XML Datei für zwei Stationen, eine Verbindung und eine Zugfahrt abgebildet und gibt den grundlegenden Aufbau der Fahrplanbeschreibung für die Simulation wieder. Im ersten Teil werden die Stationen und Verbindungen beschrieben, die allgemeingültig für die Strecke sind. Danach folgt die Beschreibung der Linien. Eine Linie besteht aus verschiedenen Zeitplänen für unterschiedliche Tage, wie Werktage, Wochenende und Sonn- und Feiertage. In einem Zeitplan befinden sich alle Fahrten für diese Tage. Eine Fahrt besteht aus mehreren Stopps. In diesen wird angegeben, wann der Zug von der Station abfährt, wie lange er vorher dort hält, an welcher Plattform der Zug ankommt und welche Verbindung er anschließend nutzt um weiter zu fahren.

5.1.4 Wegsuche

Für die Agenten ist es wichtig zu wissen, wie sie ihr Ziel erreichen. Verkehrsbetriebe stellen für ihre Fahrgäste personalisierte Onlinefahrplanauskünfte zur Verfügung. Durch die Eingabe von Start, Ziel und einer Ankunfts- oder Abfahrtszeit, wird eine Wegbeschreibung generiert. Auch in der Simulation muss den Agenten diese Möglichkeit zur Verfügung stehen. Ständige Anfragen an den Verkehrsbetrieb wären aber langsam und bei sehr vielen simulierten Agenten ist damit zu rechnen, dass die Anfragen aus Sicherheitsgründen vom Server des Verkehrsbetriebes nicht mehr beantwortet werden. Aus diesem Grund fiel die Entscheidung auf die Implementierung einer eigenen Fahrplanauskunft.
Bei dem Streckennetz handelt es sich um einen Graphen. Die Stationen sind die Knoten und die Verbindungen sind gerichtete Kanten. Die Stationen haben eine geographische Position mit der eine Heuristik bestimmt werden kann, welche die Entfernung zum Ziel nutzt und eine informierte Zielsuche zulässt. Fahrzeuge einer Linie können auch unterschiedliche Endstationen haben, so müssen auch verschiedene Abfahrtzeiten an jeder Station untersucht werden. Nun ist nicht mehr die Station als Knoten zu sehen, sondern die Zeiten im Fahrplan der Station. Aus diesem Aufbau ergibt sich ein unendlich großer Graph, da der Fahrplan zyklisch ist. Das Erreichen jedes Knotens ist jedoch nur durch Warten an der Station oder durch eine Verbindung eines Knoten der zeitlich vor dem jeweiligen Knoten liegt möglich. Eine Rückkehr

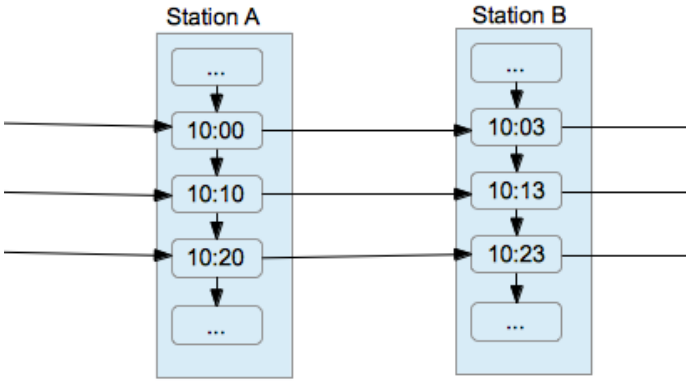

Abbildung 5.1: Graph der Verbindungen in einer Richtung

zu einem Knoten ist deshalb nicht möglich.

5.2 Fachliches Datenmodell

Im fachlichen Datenmodell wird beschrieben, wie die fachlichen Klassen und Datentypen zusammenhängen und welche Attribute von Bedeutung sind.

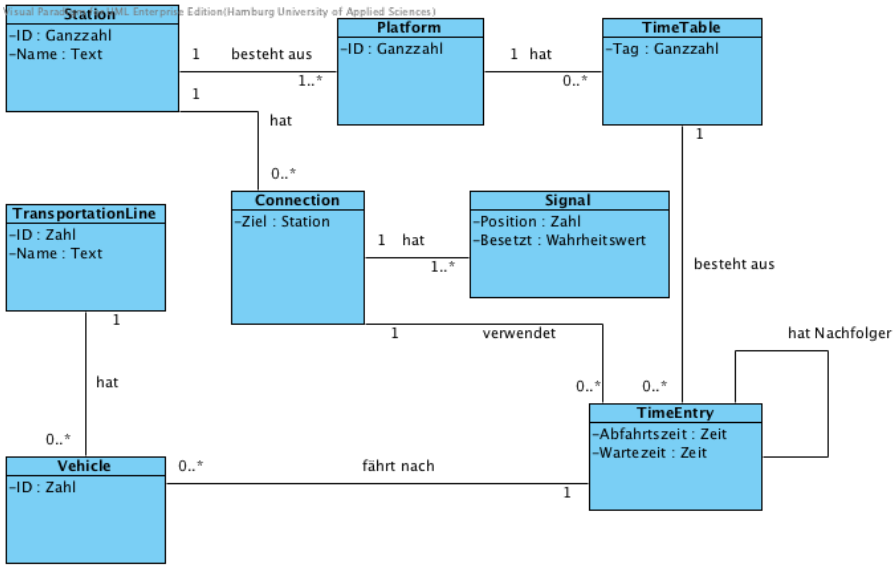

Abbildung 5.2: Fachliches Datenmodell

Ein Bahnhof (Station) kann aus mehreren Bahnsteigen (Platform) bestehen. Jedem dieser Bahnsteige sind mehrere Fahrpläne (Timetable) für unterschiedliche Tage zugeordnet, wobei sich das Attribut Tag auch auf eine Gruppe von Tagen beziehen kann. Meist werden hier Werktage und Sonn- und Feiertage gruppiert. Ein Fahrplan besteht aus mehreren Zeiteinträgen (TimeEntry). Der Aufbau entspricht so etwa dem klassischen Fahrplanaushang an einem Bahnsteig. Die Zeiteinträge besitzen jedoch noch Informationen darüber, welche Verbindung (Connection) genutzt wird und welcher der folgende Eintrag für eine Bahn (Vehicle) der Ankunftszeiteintrag ist. Eine Bahn benötigt deshalb nur einen Verweis auf einen Zeiteintrag und kann anhand diesem seine Route abfahren.

5.3 Komponentendiagramm

Nachdem das Datenmodell entworfen und die Anforderungen spezifiziert wurden, folgt der Entwurf der Anwendung. Die einzelnen Aufgaben müssen fachlich gruppiert und so in Komponenten gegliedert werden. Die einzelnen Komponenten kommunizieren untereinander über Schnittstellen und verbergen so ihre Implementation. Auf diese Weise sind sie leicht durch Komponenten, die die selben Schnittstellen implementieren, austauschbar. Zunächst wird eine Übersicht aller Komponenten in der Simulationsanwendung erläutert. Danach wird auf jede Komponente einzeln eingegangen. Der Entwurf orientiert sich hierbei an der Quasar (Qualitätssoftwarearchitektur: (Siedersleben, 2002)).

Die Komponente Simulation ist für die Verwaltung aller Ereignisse zuständig. Zusammen mit den Komponenten Gleissystem und Fahrzeug stellt sie den Kern der Anwendung dar. Ohne diese Komponenten ist das System nicht sinnvoll lauffähig. Die Komponente Wegfindung hingegen wird nur von der Fassade verwendet. Sie benötigt lediglich die Fahrplandaten aus der Gleissystemkomponente und ist daher gut austauschbar.
Das Ziel der Anwendung ist es, einen Service für ein Agentensystem zu bieten. Sie soll von diesen Systemen aus konfigurierbar sein. Ob die GUI-Komponente geladen werden soll, muss aus technischen Gründen vor dem Start der Anwendung bekannt sein. Daher wird die eigentliche Initialisation in der Fassade vorgenommen. Danach delegiert diese alle Methodenaufrufe. Im Folgenden wird nun die Innensicht der Komponenten näher betrachtet.

5.3.1 Simulations-Komponente

Die SimulationWorld ist Kern aller simulationsspezifischen Abläufe. Die Klasse EventScheduler verwaltet alle Ereignisse. Sie kann neue Ereignisse entgegennehmen und das nächste eintreffende Ereignisse bestimmen. Um die Verwaltung des EventSchedulers kümmert sich

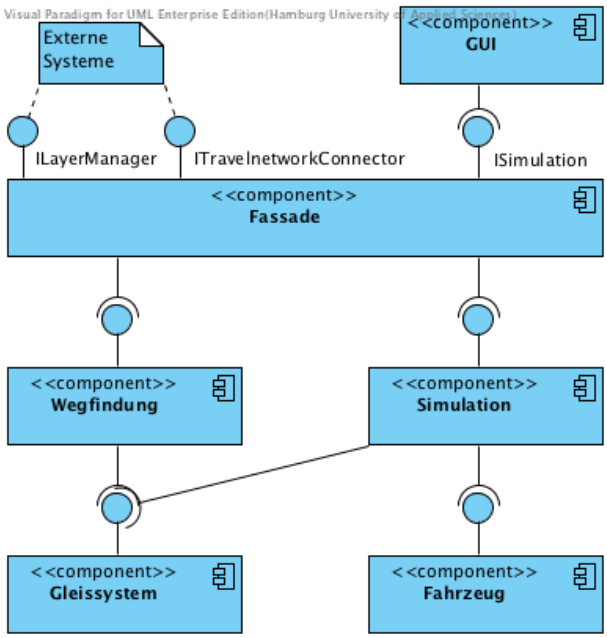

Abbildung 5.3: Komponentendiagramm

Abbildung 5.4: Bedeutung der Farbe von Klassen

die SimulationWorld. Diese Klasse wird mittels des Observerpatterns von der AWK-Fassade beobachtet, welche dann die Nachrichten über Zustandsänderungen in der Simulation an ihre Beobachter weiterleitet.

5.3.2 Fahrzeug-Komponente

In der Fahrzeug-Komponente gibt es zwei Klassen. Die Managerklasse kümmert sich um das Erzeugen und Vernichten von Fahrzeugen. Die Fahrzeugklasse stellt Methoden bereit um Statusinformationen, wie die Position, das Ziel etc. über Fahrzeuge zu erhalten. Alle Zustandsänderungen können nur von Nachrichten des EventSchedulers ausgelöst werden. Jedes Fahrzeug plant hierbei sein weiteres Verhalten selbst.

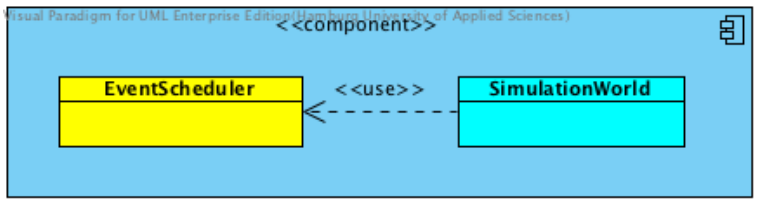

Abbildung 5.5: Simulations-Komponente

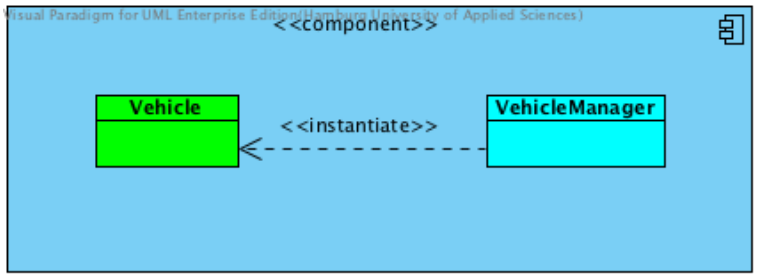

Abbildung 5.6: Fahrzeug-Komponente

5.3.3 Schienensystem-Komponente

Diese Komponente ist, an der Klassenzahl gemessen, die umfangreichste. Jedoch bieten die enthaltenen Klassen nur wenig Funktionalität. Die Klassen TimeTable, TimeEntry und Connection bieten nur die Informationen, die auch im Datenmodell beschrieben sind. Platform und Signal haben zusätzlich einen Besetztstatus. Dieser zeigt an, dass die exklusiv von einem Fahrzeug genutzt werden und somit keinem anderen zur Verfügung stehen. Fahrzeuge, die ein Signal oder einen Bahnsteig befahren wollen obwohl dieser Belegt ist, müssen sich bei diesem in eine Warteliste eintragen und werden entsprechend über das Freiwerden benachrichtigt. Die Station hat durch ihre Platformen Zugriff auf den Fahrplan. Aus diesem geht hervor, ob und wann ein Fahrzeug an der betreffenden Station seinen Fahrplan beginnt. Für dieses Ereignis registriert sich die Station um die Erstellung eines neuen Fahrzeuges einzuleiten und sich für weitere Ersteinsätze zu registrieren.

5.3.4 Wegsuchfindungs-komponente

Diese Komponente hat die Aufgabe, Routen von einem Bahnhof zu einem anderen zu berechnen und zu verwalten. Die Klasse Pathfinder sucht zu einem Start und einem Zielbahnhof und einer Startzeit eine passende Wegbeschreibung, welche durch die Klasse Path repräsentiert wird. Diese Klasse beinhaltet Informationen über die zu nutzenden Linien und Stationen.

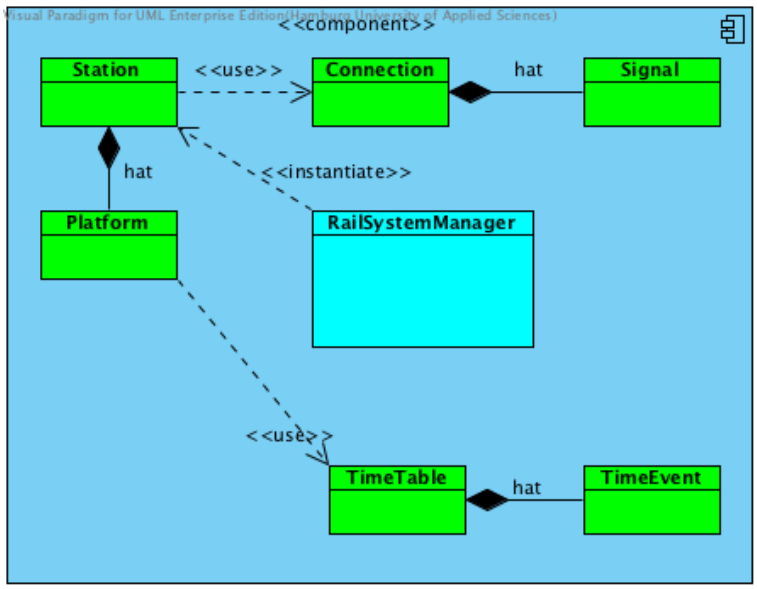

Abbildung 5.7: Schienensystem-Komponente

Nach außen wird nur die Verwaltungs-ID des Pfades gegeben. Über diese ID kann dem Pfad mitgeteilt werden, welcher Abschnitt der Route gerade genutzt wird und anhand dessen erfragt werden, ob eine Fahrt bzw. Weiterfahrt mit einer Bahn möglich ist.

5.3.5 Visualisierungs-Komponente

Diese Komponente dient zur Darstellung der Simulation. Die drei Klassen GStation, GVehicle und GConnection repräsentieren graphische Abbildungen der jeweiligen Klassen in der Simulation. Ihnen wird bei der Erstellung eine Referenz auf die entsprechende Instanz in der Simulation übergeben. Die Klasse GraphicsManager dient zur Verwaltung von Bitmaps für die graphischen Objekte. Bitmaps werden anhand ihres Namens abgefragt. Für nicht vorhandene Namen oder fehlerhafte bzw. fehlende Grafikdateien wird ein Standardbitmap vorgehalten. Die Klasse GMapGraphics dient zur Darstellung der Hintergrundkarte. Mit Hilfe der URLImageLoader Klasse werden die Graphiken von einem Kartendienst heruntergeladen. Die Klasse SimulationVisualisation ist der Kern der Visualisierung. In ihr werden alle Zeichenoperationen aufgerufen und Nutzereingaben verwaltet.
In der Komponente befindet sich die Layer Komponente. Diese beinhaltet den LayerManager, welcher für das Erstellen und Verwalten von Layern zuständig ist. In einem Layer können Elemente vom Typ ImageItem und ComplexItem erzeugt werden und dienen zur unterschiedlichen Anzeige von Informationen.

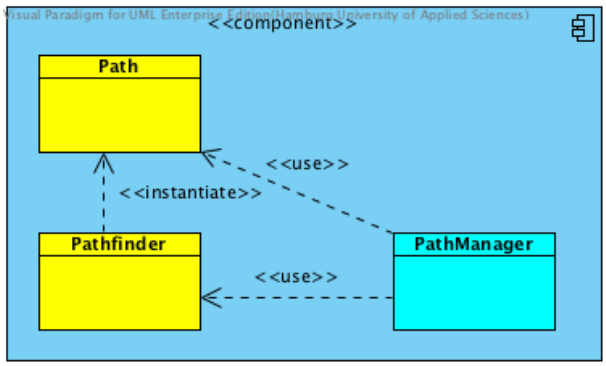

Abbildung 5.8: Wegsuchfindungs-Komponente

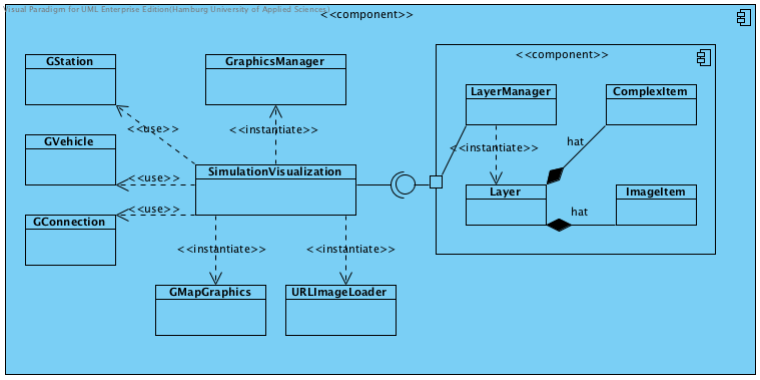

Abbildung 5.9: Visualisierungs-Komponente

5.3.6 ImportTool

Die Datestruktur dieser Software leitet sich direkt aus dem Aufbau der XML-Datei ab. Die Klasse DataReader importiert eine XML-Datei und bildet den Inhalt auf die entsprechenden Klassen ab.
Die Klasse ImportManager verwaltet alle WebImporte. Bisher existiert hier nur der Import für Daten aus dem HVV.
Die Klasse OSMData ist für das Durchsuchen der OpenStreetMap Daten zuständig um Positionen von Bahnhöfen zu ermitteln.
Von der Klasse ImportTool werden alle Funktionen gesteuert und verwaltet. Auch die GUI wird von dieser Klasse erzeugt und verwaltet.

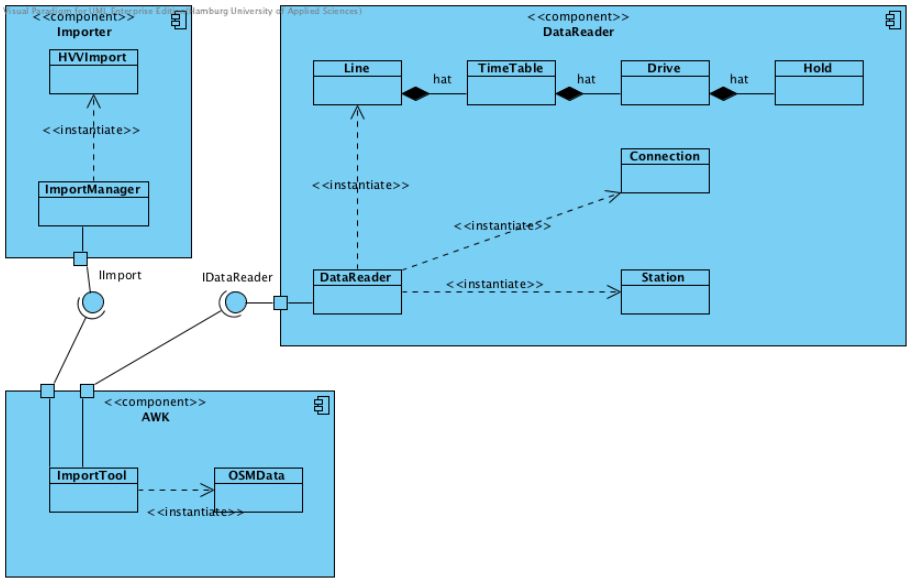

Abbildung 5.10: ImportTool

5.4 Schnittstellenentwurf

Um mit anderen Systemen, insbesondere der Masterarbeit , zusammenarbeiten zu können, muss eine Schnittstelle definiert werden, über die alle benötigten Funktionen der Simulation gesteuert werden können. Um eine möglichst lose Kopplung zu erreichen, wurde bei dem Entwurf der Schnittstelle darauf geachtet, möglichst nur primitive Datentypen zu verwenden. Die Schnittstelle stellt eine Methode zur Verfügung, die es Objekten ermöglicht, sich bei der Schnittstelle für den Erhalt von Nachrichten zu registrieren. Bei einer Nachricht handelt es sich um eine Sammlung von Informationen. Diese betreffen die Art der Nachricht und optional die beteiligten Objekte wie Bahnen, Haltestellen und Bahnsteige.
Die Masterarbeit bildet in ihrem Simulationsteil auch Bahnhöfe, Bahnsteige und Bahnen ab. Bahnen können an Bahnsteigen halten und Agenten ein- und aussteigen lassen. Damit hier die Bahnen wissen, ob und an welchem Bahnsteig sie halten, wird von der Bahnsimulation dieser Bachelorarbeit jeweils eine Nachricht für das Halten und das Abfahren der Bahnen an die Agentensimulation geschickt. Diese Nachricht enthält die betreffende Bahn ID, die ID des Bahnsteigs, an dem die Bahn hält bzw. von dem sie abfährt und die ID des Bahnhofs der diesen Bahnsteig beinhaltet. Die IDs sind Ganzzahlen. Diese ändern sich zur Laufzeit für ein Objekt nicht und sind pro Objekttyp eindeutig. Sie können aber wieder verwendet werden, wenn Objekte zerstört werden. Kurz vor der Abfahrt wird noch eine Nachricht geschickt, die das

Schließen der Türen signalisiert. Diese Nachricht enthält die gleichen Informationen wie die zwei zuvor genannten Nachrichten. Beide Systeme besitzen getrennte Repräsentationen der Bahnen. Um eine Verbindung zwischen den Bahnen beider Systeme zu bekommen, sendet die Bahnsimulation eine Nachricht über jede neu erstellte Bahn an die Agentensimulation. Diese Nachricht enthält nur die ID der Bahn, da sich die Bahn nach der Erstellung noch an keinem Bahnhof befindet. Dieses hat den Grund, dass der Bahnsteig zu diesem Zeitpunkt von einer anderen Bahn besetzt sein könnte. Daher wird sie nicht direkt auf dem Bahnsteig erstellt, sondern muss in diesen erst einfahren. Dieses löst einerseits, wie erwähnt, das Problem, dass der Bahnsteig besetzt sein könnte, andererseits ist dieses auch realistischer, da die Bahn von einem Abstellgleis kommen könnte.
Ein Sonderfall ist das Erreichen der Endstation. Hier wird bei der Ankunft der Bahn eine weitere Nachricht gesendet. Diese beinhaltet wieder sämtliche Informationen zu Bahn, Bahnsteig und Bahnhof. Sie teilt der Agentensimulation mit, dass hier alle Agenten aussteigen müssen. Nachdem die Bahn, die eine Endstation erreicht hat, wieder abfährt, wird sie aus der Bahnsimulation entfernt. Dieses wird auch zuvor per Nachricht an die Agentensimulation gemeldet, so dass diese auch ihrerseits ihre Instanz der Bahn freigeben kann.
Wenn eine Bahn das letzte Signal vor einem Bahnhof erreicht wird eine Nachricht an die Agentensimulation gesendet, die den Agenten die nächste Haltestelle mitteilen soll. Daher enthält diese Nachricht zusätzlich zur Bahn ID auch die ID des nächsten Bahnhofs. Dadurch können die Agenten sich schon vor der Ankunft zu den Ausgängen begeben und so schneller aussteigen.

6 Implementierung

Nachdem die Analyse abgeschlossen und der Entwurf erstellt wurde, kann die Implementierung der Anwendung erfolgen. Zuvor müssen aber noch die zu nutzenden Techniken gewählt werden.

6.1 Verwendete Technologien

Die Masterarbeit wurde schon in Java angefangen. Daher fiel hier auch für die Simulation des Nahverkehrs die Wahl auf Java. Java ist eine von SUN entwickelte Programmiersprache. Sie übersetzt den Code nicht für eine spezielle Plattform, sondern für eine eigens für Java entwickelte virtuelle Maschine. Auf allen Plattformen, auf der diese virtuelle Maschine lauffähig ist, sind auch die für Java entwickelten Anwendungen ausführbar.
Auch für die Implementierung der graphischen Aufbereitung wurde Java gewählt, da diese beiden Systeme sehr eng zusammenarbeiten. Zur Graphikausgabe wurde zudem Slick2D genutzt. Slick2D ist eine 2D Graphikbibliothek für Java. Sie nutzt OpenGL oder Java2D und ist damit für vielen Plattformen verfügbar und bietet mit diesen Technologien auch die Möglichkeit, Grafiken hardwarebeschleunigt ausgeben zu lassen.

Bei dem Tool zum Import der Fahrplandaten wurde nicht konsequent auf Java gesetzt. Bei ersten Tests mit sehr großen XML Dateien und der Standard Java XML Bibliothek traten Probleme auf. Deshalb wurde hier C# und Microsoft Visual Studio verwendet. Dieses bot außerdem den Vorteil, dass die GUI mit den in Visual Studio integrierten Tools sehr einfach ist und die Spracherweiterung LINQ vereinfacht das Iterieren über die verschachtelte Datenstruktur der Fahrplandaten. C# setzt ähnlich wie Java auf eine virtuelle Maschine, jedoch wird von Microsoft nur Windows offiziell unterstützt. Da die Benutzung des Tools nur zur Erstellung des Fahrplans nötig ist, überwogen hier aber die genannten Vorteile.

6.2 Umsetzung

Im ersten Schritt wurde das Simulationssystem entwickelt. Dieses hat keine Abhängigkeiten zu anderen Systemen, ist allein lauffähig und lässt sich gut testen. Auch lässt sich hierbei schnell erkennen, ob die geplante Umsetzung bei der Geschwindigkeit und Leistungsfähigkeit ausreichend ist.
Der Grundgedanke der Simulation ist eine zentrale Verwaltung aller Ereignisse. Ein Ereignis benachrichtigt bei seinem Eintreten das betreffende Objekt. Dieses wird mit einer Priority Queue gelöst. Eine Priority Queue kann auf verschiedenen Wegen implementiert werden. Die Grundlegende Funktion entspricht einer sortierten Liste. Intern werden die Daten aber meist in einer Baumstruktur aufbewahrt. Eine Priority Queue ist häufig auf bestimmte Operationen optimiert. So ist das Einfügen neuer Elemente und das Lesen und gleichzeitige Entfernen des ersten Elementes, nach dem jeweiligen Sortierkriterium, besonders schnell. Das Löschen und Finden beliebiger Objekte erfordert hingegen meist das Durchsuchen aller Einträge.
Zusätzlich werden Ereignisse, die keiner Planung bedürfen, da sie direkt auf ein Ereignis folgen, das sie auslöst, in einer verketteten Liste eingetragen und priorisiert behandelt. Dieses erspart die Verwaltung durch die Priority Queue und dient zur Steigerung der Performance.

6.2.1 Laden der Daten

Bevor die Simulation beginnen kann, müssen die Daten aus der XML-Datei mit den Fahrplandaten, den Stationen und Verbindungen geladen werden. Hier wurde der XML-Parser verwendet, der in den Standard Java-Bibliotheken vorhanden ist.

```
1  eachElement(stationList, "Station", new F1<Element>() {
2    public void f(Element station) {
3      String name = station.getAttribute("name");
4      Position position = parsePosition(
5        station.getAttribute("position")
6      );
7      int platformCount = readInt(station, "platformCount", 2);
8
9      final Station s = world.createStation(
10       name, position, platformCount
11     );
12
```

```
13      eachElement(station, "Transit", new F1<Element>() {
14        public void f(Element transit) {
15          Integer from = Integer.parseInt(
16            transit.getAttribute("from")
17          );
18          Integer to = Integer.parseInt(
19            transit.getAttribute("to")
20          );
21          int duration = parseTime(
22            transit.getAttribute("duration")
23          );
24          s.setTransitTime(from, to, duration);
25        }
26      });
27    }
28  });
```

Listing 6.1: Code zum Laden aller Stationen in C#

Durch anonyme Klasse wurde hier versucht, eine ähnlich klare Struktur, wie es im später folgenden Beispiel mit LINQ gezeigt wird, zu erreichen. Auf Fehler beim Lesen der XML Datei wird mit einem Abbruch reagiert, da nicht davon ausgegangen werden kann, dass die Simulation mit fehlerhaften Daten funktioniert.

6.2.2 Die Bahnen

Bahnhöfe erhalten eine Nachricht, wenn Bahnen bei ihnen ihren Fahrplan beginnen. Beim Eintreffen dieser Nachricht wird eine neue Bahn erstellt. Einer Bahn wird bei der Erstellung der erste Zeiteintrag aus seinem Fahrplan und die nächste zu benutzende Verbindung übergeben. Da der entsprechende Bahnsteig zu dem Zeitpunkt der Erstellung durch ein anderes Fahrzeug belegt sein könnte, muss sich jede neue Bahn ggf. auch in die Warteliste des Bahnsteiges einreihen. Nachdem die Bahn erfolgreich erstellt wurde, werden mögliche externe Systeme über die neue Bahn unterrichtet. Beim Halt an einem Bahnsteig wird überprüft, ob es sich um das Ende des Fahrplans handelt. In diesem Fall wird eine Endstationsnachricht versendet. In jedem Fall wird aber auch eine Ankunftsnachricht gesendet. Die Bahn reiht nun eine Nachricht über seine geplante Abfahrt in die Priority Queue ein. Die Wartezeit bis dorthin entnimmt sie ihrem Fahrplan. Sollte die Bahn frühzeitig den Bahnhof erreicht haben, wird die Wartezeit dementsprechend verlängert.
Beim Eintreffen der Abfahrtsnachricht bei der Bahn, wird der nächste Zeiteintrag ermittelt.

Daraus ergibt sich die zu nutzende Verbindung, das Ziel und die geplante Ankunftszeit. Angeschlossene Systeme werden nun über das Schließen der Türen und anschließend über die Abfahrt der Bahn unterrichtet.
Eine Fahrt zum nächsten Ziel wird etappenweise geplant. Es wird immer nur ein zukünftiges Ereignis des Eintreffens am nächsten Signal in die Queue eingereiht. Beim Überqueren des ersten Signals wird zudem der Bahnsteig wieder freigegeben. Dieser kann nun eventuell wartende Fahrzeuge benachrichtigen, dass sie einfahren können.

6.2.3 Wegsuche

Für die Wegsuche wurde der A* Algorithmus verwendet. Als Heuristik dient die zeitliche Entfernung zum Ziel. Diese Zeit wird mittels der geographischen Entfernung und der maximalen Geschwindigkeit bestimmt. Wie erwähnt, ist durch die strikte Minutentaktung im Fahrplan die Geschwindigkeit der Bahnen auf einige Strecken langsamer oder schneller als in der Realität. Aus diesem Grund musste hier ein unrealistisch hoher Wert gewählt werden, damit die Heuristik die Entfernung zum Ziel nicht überschätzt. Je mehr die Heuristik die Entfernung unterschätzt, desto mehr Knoten werden im allgemeinen untersucht.
Als Knoten im Suchgraphen dient ein Tripel bestehend aus Bahnhof, Bahnsteig und Ankunftszeit. Als Verbindung der Knoten dient hier die Verbindung zweier Stationen. Hierbei wird zu jeder Linie und jeder Verbindung überprüft, ob diese im Fahrplan Verwendung finden und nur dann untersucht. Auch werden nur Verbindungen untersucht, die nicht zu der Station zurückführen, von der der zu untersuchende Knoten erreicht wurde. Auf diese Weise wird sichergestellt, dass alle, von dem aktuellen Knoten erreichbare Stationen untersucht werden und auch Strecken mit verzweigten Wegen einer Linie genutzt werden.
Die ursprüngliche Anforderung, dass die Suche eines Pfades von einem Bahnhof zu einem anderen Bahnhof stattfindet, wurde im späteren Verlauf modifiziert. Die abgewandelte Anforderung verlangte, dass von einer Startposition zu einer Zielposition ein Weg gesucht wird. Dadurch ist nicht immer der nächste Bahnhof am Startpunkt der Startbahnhof, da es durchaus optimaler sein kann, einen etwas weiter entfernten Bahnhof als Start zu wählen, wenn dieser an eine andere Bahnlinie angeschlossen ist oder die Abfahrtzeiten günstiger sind. Aus diesem Grund werden alle Bahnhöfe in einem gewissen Radius als Startbahnhof in Betracht gezogen. Sollte in diesem Radius kein Bahnhof sein, wird der nächstliegende Bahnhof gewählt. Auch der Zielbahnhof ist nicht mehr eindeutig und ist von der optimalen Route abhängig. Daher wird ab einem einstellbarem Abstand zum Ziel von jedem Bahnhof ein Fußweg zum Ziel berechnet und den Zieloptionen der Wegsuche hinzugefügt. Hierbei wird nur die direkte Entfernung ge-

nommen. Straßenführungen können hierbei nicht berücksichtigt werden. Damit sichergestellt ist, dass mindestens ein Bahnhof zum Ziel führt, wird der Abstand zum nächsten Bahnhof bestimmt und als kleinste Entfernung für den Fußweg gewählt, auch wenn die Suchparameter einen noch kleineren Wert vorschreiben.

6.2.4 Wegbefolgung

Damit angeschlossene Systeme nicht viel über das Verkehrsnetz wissen müssen, wird die Befolgung des in der Wegsuche berechneten Pfades auch von der Bahnsimulation übernommen. Nach dem Finden eines Pfades wird nur die ID des Pfades nach Außen weitergegeben. Reisende Agenten können unter der Angabe einer Pfad-ID erfragen, ob sie mit einem Zug weiterfahren müssen oder auf welchen Bahnsteig sie wechseln sollen, solange sie sich an einer Station befinden, die in dem Pfad vorhanden ist.
Die Pfadsuche sucht zwar nur den besten Pfad für eine bestimmt Abfahrtzeit heraus, ist aber für gewöhnlich über einen längeren Zeitraum gültig, da mehrere Fahrzeuge der gleichen Linie die gleichen Stationen abfahren. Aus diesem Grund ist der Pfad nicht an bestimmt Fahrzeuge gebunden, sondern überprüft lediglich, ob ein Fahrzeug der richtigen Linie zugewiesen ist und der nächste Bahnhof, den das Fahrzeug anfährt, im berechneten Pfad vorgesehen ist.

6.2.5 Visualisierung

Die Visualisierungskomponente wurde für das Anzeigen mehrerer Ebenen konzipiert. Auf der untersten Ebene ist eine Landkarte der Umgebung angezeigt. Die Grafiken stammen hier von OpenStreetMap und werden automatisch geladen. Hierzu wird ein Rechteck errechnet, welches alle Bahnhöfe auf der Karte umspannt. Anschließend werden die Grafiken von einem Kachelserver geladen und zusätzlich auf der Festplatte gesichert. Das spätere Laden der Grafiken geschieht hierbei transparent. Wenn die abgefragte URL für eine Kachel schon gecached wurde, wird die entsprechende Datei geladen und keine Webanfrage erstellt.
Die nächst höhere Ebene ist die Darstellung der Bahnhöfe, Bahnen und Verbindungen. Bahnhöfe und die Verbinduneg zwischen ihnen sind unveränderlich. Daher werden diese in einem QuadTree gespeichert. Bei einem QuadTree handelt es sich um eine Datenstruktur in Baumform, bei der jeder Knoten keine oder vier Kinder hat. Die Wurzel des Baums beinhaltet das gesamte geographische Gebiet der Simulation. Die möglichen vier Kindknoten der Wurzel beinhalten jeweils ein Viertel des Gebietes. Es gibt mehrere Möglichkeiten Objekte in diesen

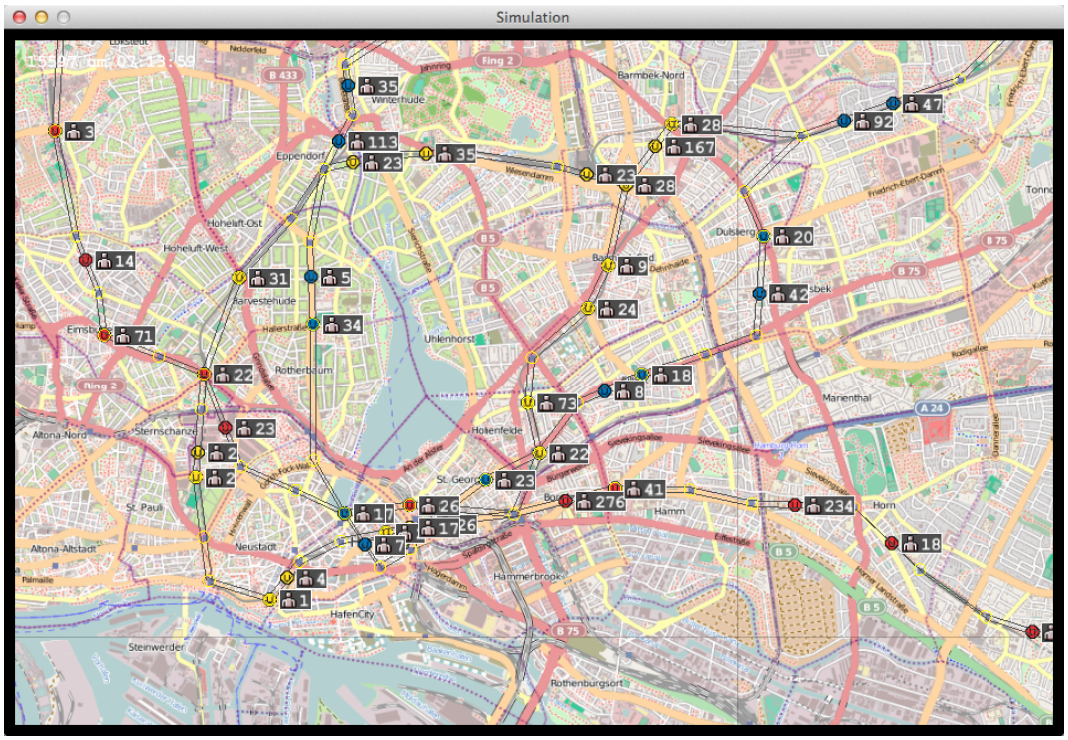

Abbildung 6.1: Visualisierung

Baum einzufügen. Die hier gewählte Methode fügt nur Objekte, die in ihrer ganzen Ausdehnung in einen Knoten passen, dem jeweilige Knoten hinzu. Bahnhöfe haben in der Simulation keine geographische Ausdehnung und werden daher immer auf der tiefsten definierten Knotenebene des Baums eingefügt werden. Verbindungen können auch auf höheren Knotenebenen eingefügt werden.

Beim Zeichnen der Karte kann schnell erkannt werden, welche Objekte überhaupt sichtbar sind. Hierzu wird ausgehend von der Wurzel überprüft, welche Kindknoten im Zeichenbereich liegen. Sollte ein Knoten nicht im Zeichenbereich liegen, können seine Kindknoten auch nicht in diesem Bereich liegen und müssen daher nicht weiter untersucht werden. Gebiete in denen keine Objekte vorhanden sind, haben auch keine tiefe Ausprägung von Knoten und können daher schnell abgearbeitet werden.

Da die Bahnen nur auf Verbindungen zwischen Bahnhöfen verkehren, werden diese nur gezeichnet, wenn die entsprechende Verbindung sichtbar ist.

Die gesamte Simulation wird durch einen externen Zeitgeber mit der aktuell simulierten Zeit versorgt. Hierbei ist nicht sichergestellt, dass diese Zeit linear abläuft. Aus der Sicht der Simulation stellt dieses kein Problem dar. Für die Visualisierung ist es aber so schwieriger eine

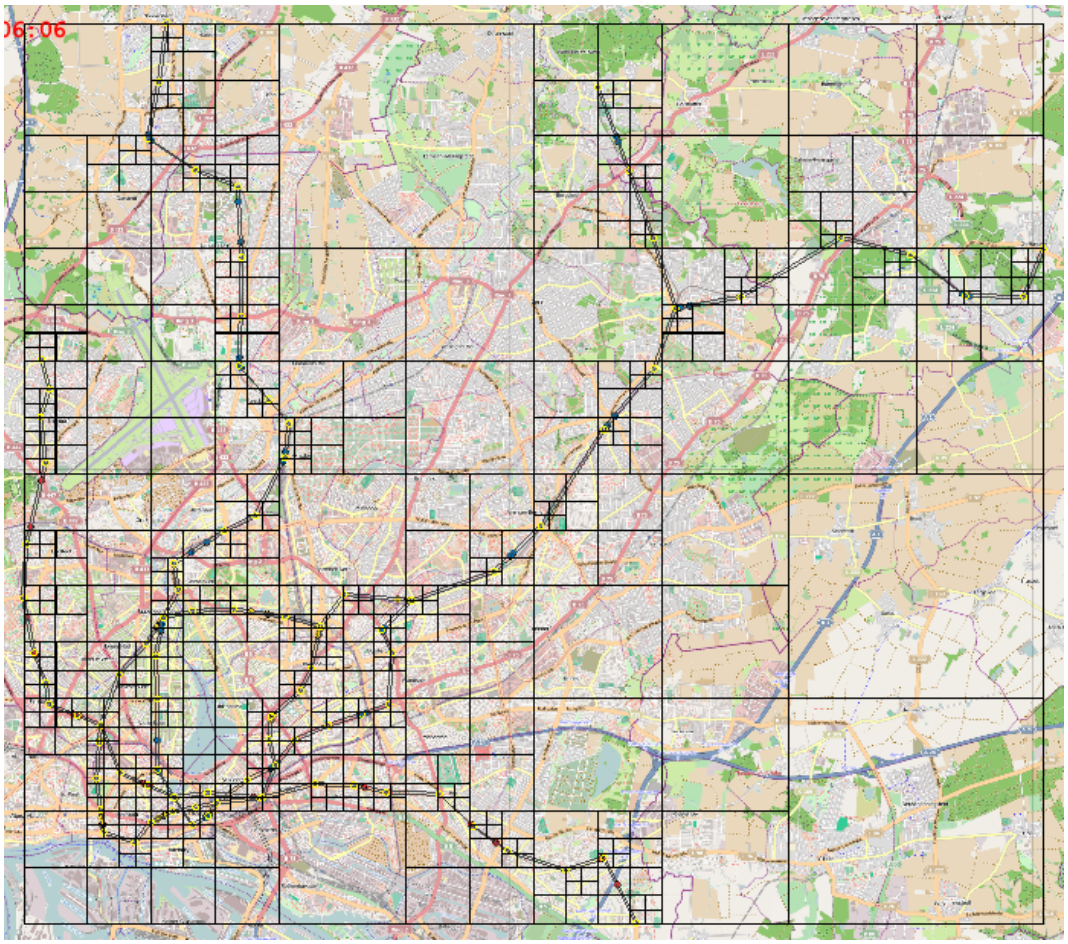

Abbildung 6.2: QuadTree am Beispiel der U-Bahnlinien in Hamburg

flüssige Bewegung der Bahnen darzustellen. Daher wird bei jeder Aktualisierung der Zeit die verstrichene Zeit, seit der letzten Aktualisierung gemessen und so eine wahrscheinliche Simulationsgeschwindigkeit bestimmt. Mittels dieses Wertes wird dann eine eigene Uhr betrieben. Diese Uhr wird dann genutzt, um mit Hilfe der zuletzt bekannten Position einer Bahn, der Zielposition und den zugehörigen Zeiten die aktuelle Position der Bahnen zu bestimmen.
Oberhalb des Gleissystems können weitere Ebenen erzeugt werden. Diese Möglichkeit wird über ein Interface auch von externen Systemen nutzbar. Hierbei gibt es zwei mögliche grafische Objekte die dargestellt werden können. Die erste Möglichkeit ist das Anzeigen eines Bildes. Hierbei wird die Position und der Dateiname einer Grafik übergeben. Es ist auch möglich, das Bild im Nachhinein zu ändern. Auf diese Weise können von der Agentensimulation Wohnorte, Geschäfte, Arbeitsplätze und sonstige Ziele der Agenten angezeigt werden. Die zweite

Möglichkeit besteht im Hinzufügen von Bereichen. Hierbei werden die Bereiche als Liste von Positionen übergeben. Der Bereich kann individuell farblich gefüllt und umrandet werden. Auf diese Weise können Stadtteile eingeblendet werden.
Sowohl die Grafiken als auch die Bilder bieten noch die Möglichkeit, zugehörige Texte einzublenden und sind mit der Maus auswählbar. Externe Systeme können sich an der Ebenenverwaltung registrieren und werden dann über die Auswahl eben jener Objekte und über die Auswahl von Bahnhöfen und Bahnen informiert.
Alle Ebenen, außer der Kartenebene, können individuell ein und ausgeblendet werden. Die Ebene des Gleissystems verbleibt immer unter den anpassbaren Ebenen. Die anderen Ebenen werden in der Reihenfolge, in der sie sichtbar gemacht wurden, angezeigt.
Die Benutzereingaben geschehen bei dieser Komponente ausschließlich mit der Maus. Mit der linken Maustaste werden Objekte ausgewählt. Durch das Gedrückthalten der rechten Maustaste und gleichzeitige Bewegen der Maus über die Karte wird der sichtbare Ausschnitt der Karte verschoben und mit dem Mausrad wird der Ausschnitt vergrößert und verkleinert. Das Skalieren der Darstellung ist in Slick2D, dank der Anbindung an OpenGL sehr einfach und performant, da der Vergrößerungsfaktor direkt angegeben werden kann und nicht bei jedem zu zeichnenden Objekt erneut mit der Position verrechnet werden muss. Das hat allerdings den Nebeneffekt, dass auch Texte, die in Slick2D auch nur aus zusammengesetzten Grafiken bestehen, mit skaliert werden und so übermäßig groß bzw. klein werden können. Daher wurde eine zusätzliche Textebene eingerichtet, die alle Textausgaben puffert und erst nachdem der Zeichenbereich zurückgesetzt wurde, also eventuelle Skalierungs- und Translations- Operationen rückgängig gemacht wurden, zeichnet. Auf diese Weise bleiben Texte immer gut lesbar.
Beim gemeinsamen Testen der Schnittstellen mit der Masterarbeit kam es zu Problemen, da beide Projekte Slick2D nutzten. Es ist nicht ohne Weiteres möglich, zwei Fenster mit Slick2D gleichzeitig zu öffnen. Daher wurde eine Bibliothek entwickelt, die es den beiden Projekten gestattet, einen gemeinsamen Zeichenbereich zu verwenden. Hierbei wird initial ein rechteckiger Zeichenbereich festgelegt. Der Aufruf zum Zeichnen wird dann entsprechend weitergeleitet. Vorher wird die Zeichenfläche transponiert, damit die Ursprungskoordinaten in der oberen linken Ecke des jeweiligen Bereiches liegen. Zusätzlich wird der Bereich, der zum Zeichnen zur Verfügung steht, auf die Größe des Rechtecks eingeschränkt. Auf diese Weise konnten die bestehenden Zeichenfunktionen ohne größere Änderungen wieder verwendet werden. Ebenso werden die Benutzereingaben durch Maus und Tastatur an die Visualisierung weitergeleitet, die zu dem Zeitpunk des Stattfindens den Fokus hat. Der Fokus wird dabei an das Objekt vergeben, über dem sich der Mauszeiger befindet. Sollte die Maus gedrückt sein und den

Bereich einer Visualisierung verlassen, behält diese den Fokus bis die Maustaste gelöst wird.

6.2.6 Importtool

Im Verlauf dieser Bachelorarbeit wurde keine Möglichkeit gewährt, direkt auf die Fahrplandaten des HVVs zuzugreifen. Aus diesem Grund wurde ein Fahrplandaten Importtool entwickelt, um die Erstellung des Fahrplans zu vereinfachen.
Im ersten Schritt wird hier im Reiter Import das entsprechende Webimportmodul ausgewählt. Dieses ist von der entsprechenden Datenquelle abhängig und wurde in dieser Bachelorarbeit für die Geofox-Seite mit den HVV Fahrplandaten erstellt. Das HVV-Webimportmodul beinhaltet eine URL zur Startseite des Linienfahrplans. Diese URL wird im eingebundenen Webbrowser aufgerufen. Hier können die Linie und weitere Einstellungen eingestellt werden. Anschließend kann der Linienfahrplan abgefragt werden.
Nach dem Laden der Linienfahrplanseite wird der HTML-Code der Seite untersucht. Im ersten Schritt werden Sonderfahrten gesucht. Über einer Fahrt befindet sich optional eine Graphik bestehend aus einer Zahl in einem schwarzen Kreis. Am unteren Ende der Seite befindet sich eine Legende zu diesen Zahlen. Hier wird die Gültigkeit für die entsprechenden Fahrten erklärt. Die Nummern der Sonderfahrten werden erkannt und können anschließend vom Benutzer, für die Erstellung des Fahrplans ausgewählt werden. Mit einem Klick auf Import werden anschließend alle erkannten Fahrten und Stationen in den Datenbestand aufgenommen. Das HVV-Webimportmodul ist dabei fest auf das HVV-Portal eingestellt. Der gesamte Code der Seite wird hierbei nach html-Tags durchsucht. Je nach gefundenem Tag wird eine andere Funktion ausgeführt oder das Tag ignoriert. Hierbei arbeitet des Importmodul in drei Modi:
Im ersten Modus werden alle Tags übersprungen, bis der Anfang der Fahrplandatentabelle erreicht ist. Nach einer bestimmten Anzahl von gefundenen Table-Row-Tags (<TR>) wird in den zweiten Modus gewechselt.
Der zweite Modus ist für die Erkennung der Sonderfahrtenmarkierung zuständig. Dieser Modus wird übersprungen, wenn keine Sonderfahrtenreihe vorhanden ist. Ob Sonderfahrplanmarkierungen vorhanden sind, wird an einer leeren Zelle am Anfang der Reihe erkannt. Andernfalls würde hier der erste Bahnhof stehen.
Im Dritten Modus werden anschließend die Bahnhöfe und die Abfahrtszeiten erkannt. Hierbei wird bei der verwendeten Linie und Richtung im Vorfeld überprüft, ob es schon Eintragungen für Bahnsteige gibt. In diesem Fall wird der bisher genutzte Bahnsteig weiter genutzt. Im anderen Fall wird die Anzahl der Bahnsteige einer Station um eins erhöht und der neue Bahnsteig für alle Halts der Bahnen genutzt.

Auf diese Weise kann nun der Gesamtfahrplan für alle gewünschten Linien erstellt werden.

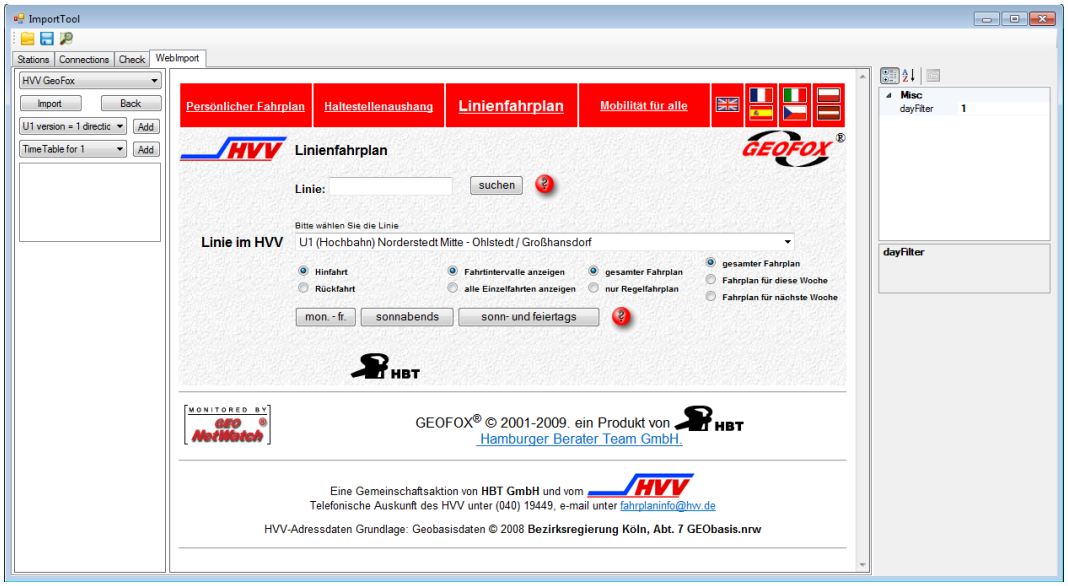

Abbildung 6.3: ImportTool: WebImport

Anschließend sind im Reiter „Stations“ alle bisher gefundenen Bahnhöfe aufgelistet. Da ein händisches Eintragen aller Positionen der Bahnhöfe sehr mühsam wäre, wurde die Aufgabe auch größtenteils automatisiert. Dazu ist im Vorfeld das Erstellen einer OpenStreetMap Regionsdatei nötig. Diese muss das gesamte Gebiet des Verkehrsnetztes umfassen. Anschließend kann diese Datei in das ImportTool geladen werden. Während des Ladens der Datei wird diese auf Positionen und Namen von Haltestellen durchsucht. Durch die Betätigung der Schaltfläche „Find Locations“ wird versucht, jeder Haltestelle im Datenbestand einen Position aus den OpenStreetMap Daten zuzuweisen. Hierbei werden die Namen der Station mit den gefunden Namen in der OpenStreetMap Datei verglichen und die Station mit der höchsten Ähnlichkeit des Namens gewählt und die zugehörige Position übernommen. Die Ähnlichkeit der Namen wird hierbei mit der Levenshtein-Distanz [1] bestimmt.
Um die Wegsuche zu verbessern können auch Umstiegszeiten angegeben werden. Dieses ist nur bei Haltestellen nötig, bei denen ein Umsteigen sinnvoll ist. Haltestellen die nur von einer Linie befahren werden, von der man durch Umsteigen nur wieder zurück zur Ausgangshaltestelle kommt, werden von der Wegsuche nicht berücksichtigt. Hierbei muss die Umstiegszeit von jedem Bahnsteig zu jedem anderen angegeben werden.

[1] Die Levenshtein-Distanz bestimmt die minimale Anzahl an Einfüge-, Lösch- und Ersetzoperationen die nötig sind, um eine Zeichenkette in eine andere zu überführen. (vgl. http://de.wikipedia.org/wiki/Levenshtein-Distanz)

Innerhalb des Tabs mit dem Namen "Check"wird eine Liste mit Fahrplankollisionen angezeigt. Es werden all jene Einträge aufgelistet, in denen zwei Bahnen zur selben Zeit den selben Bahnsteig belegen. Dieses kann auftreten, wenn Fehler beim Import geschehen sind oder aber auch, wenn zwei Bahnen sich an einem Bahnsteig aneinander koppeln und anschließend gemeinsam weiterfahren. In jedem Fall sollten diese Kollisionen aufgelöst werden, damit die Bahnen ihren Fahrplan einhalten können.
Der Tab „Connections“ enthält eine Liste aller Verbindungen zwischen den Stationen und stellt keine weiteren Funktionen bereit.
Die meisten Objekte, die in der GUI wählbar sind, können direkt im Eigenschaftsfeld an der rechten Seite bearbeitet werden. Dem Eigenschaftsfeld können die gewählten Objekte direkt übergeben werden. Anhand von Annotations [2] im Quellcode erkennt das Feld welche Eigenschaften unter welchem Namen änderbar sein sollen. Komplexere Eigenschaften wie die Position, welche aus zwei Zahlenwerten besteht, können mit Hilfe von beschreibenden Klassen ebenfalls angezeigt und bearbeitet werden. Dieses erspart eine Gruppe von jeweils angepassten Steuerelementen für jedes Datenobjekt. Änderungen in dem Eigenschaftsfeld werden direkt in dem genutzten Datenobjekt gesichert.
Wenn die Datenpflege abgeschlossen ist, können die Daten als XML-Datei gespeichert werden.

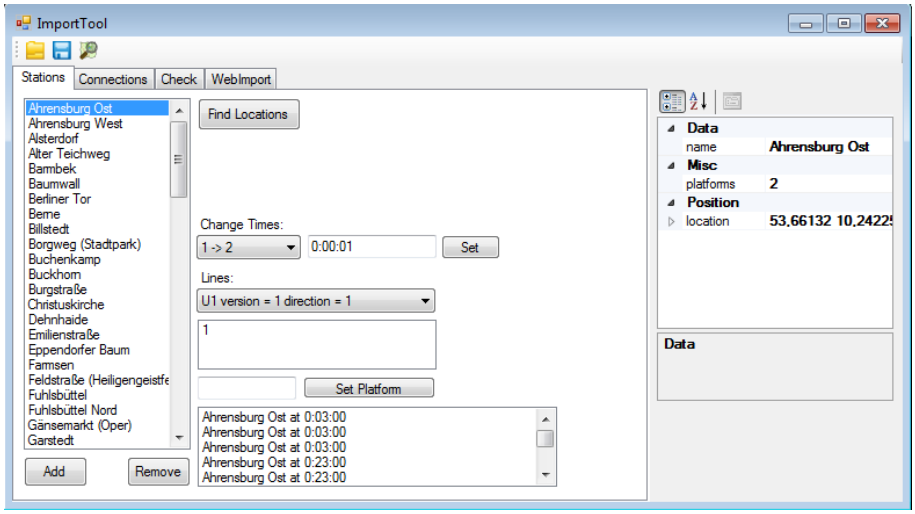

Abbildung 6.4: ImportTool: Haltestellenüberlicht

Hierzu wird erst im Speicher ein Xml-Document erstellt, welches dann in eine Datei geschrie-

[2] Annotations sind Metainformationen zu Klassen, Variablen etc. die im Quellcode beschrieben werden und zur Laufzeit ausgewertet werden können

ben wird. Zum Laden der Datei wird LINQ to XML genutzt. Hierbei wird erst die XML Datei geladen und anschließend werden mit Hilfe von LINQ die Daten extrahiert.

```
var stations =
  from station in xmlDoc.Descendants("Station")
  select new Station
  {
    name = station.Attribute("name").Value,
    location = ParseLocation(
      ReadAttribute(station, "position", null)
    ),
    platforms = Convert.ToInt32(
      ReadAttribute(station, "platformCount", "1")
    ),
    changeTimes = (
      from changeTime in station.Descendants("Transit")
      select new PlatformChange {
        fromPlatformId = Convert.ToInt32(
          ReadAttribute(changeTime, "from", "1")
        ),
        toPlatformId = Convert.ToInt32(
          ReadAttribute(changeTime, "to", "2")
        ),
        duration = Time.parse(
          ReadAttribute(changeTime, "duration", "0:00:10")
        )
  }).ToArray()
};
```

Listing 6.2: Code zum Laden aller Stationen in Java

Im Listing 6.2 ist die Implementierung für das Laden aller Bahnhöfe abgebildet. Durch die SQL ähnliche Abfragesyntax bleibt der Code kurz und übersichtlich. Durch das von C# unterstützte "Type Inference"werden alle Variablen automatisch dem passenden Typ zugeordnet, was insbesondere bei der Verwendung von LINQ zur Lesbarkeit beiträgt.

6.3 Sequenzdiagramme

Der Übersichtlichkeit halber wurden die den Diagrammen keine konkreten Werte angegeben. Daher fehlt bei „registerEvent“ eine Zeit- und eine Empfänger-Angabe, sofern der Ersteller auch der Sender des Ereignisses ist.

In Abb. 6.5 ist dargestellt, wie eine neu Bahn erzeugt wird. Jede Station wird einmalig initiali-

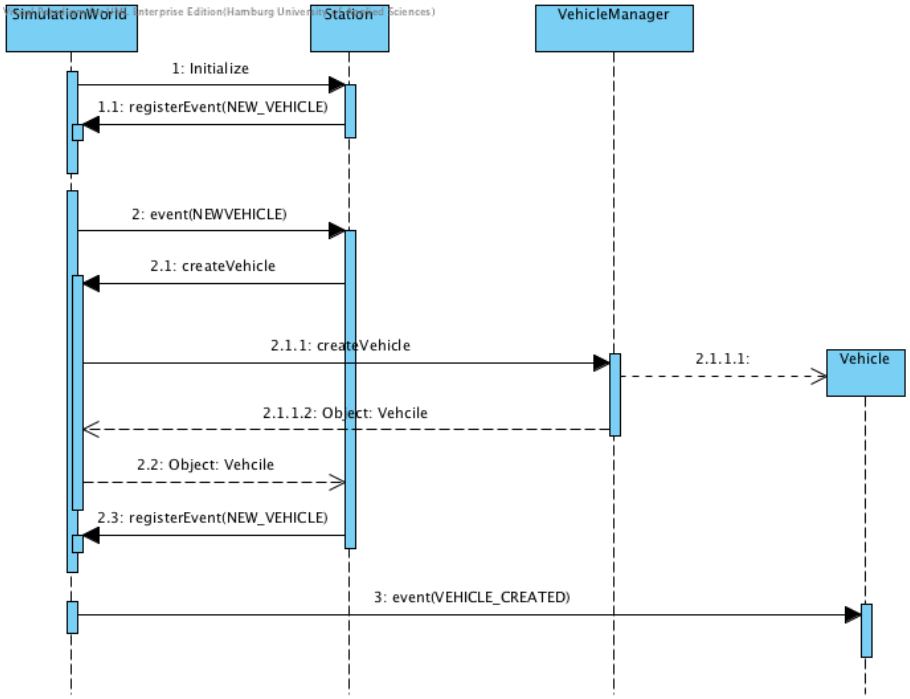

Abbildung 6.5: Erzeugung einer neuen Bahn

siert und erzeugt hierbei ein Ereignis, welches beim Eintreten eine neue Bahn erzeugen soll. Zu einem späteren Zeitpunk tritt dieses Ereignis ein und informiert den Bahnhof, welcher daraufhin ein Fahrzeug erzeugen lässt und es mit den entsprechenden Fahrplandaten ausstattet. Anschließend plant der Bahnhof das nächste zu erstellende Fahrzeug.
Die Bahn wird dann über ihre Erstellung benachrichtigt. Hier würde sich die Bahn bei einem Bahnhof in die Warteliste setzten. Dieser Punkt wird im folgenden Sequenzdiagram behandelt.

In Abb. 6.6 ist die Fahrt einer Bahn auf einer Strecke mit nur einem Signal dargestellt. Wartet die Bahn erst einmal am Abfahrtssignal. In diesem Fall gestattet es die Durchfahrt für die Bahn und erstellt ein entsprechendes Ereignis.
Darüber wird die Bahn informiert und erstellt ein Ereignis für die Ankunft am nächsten

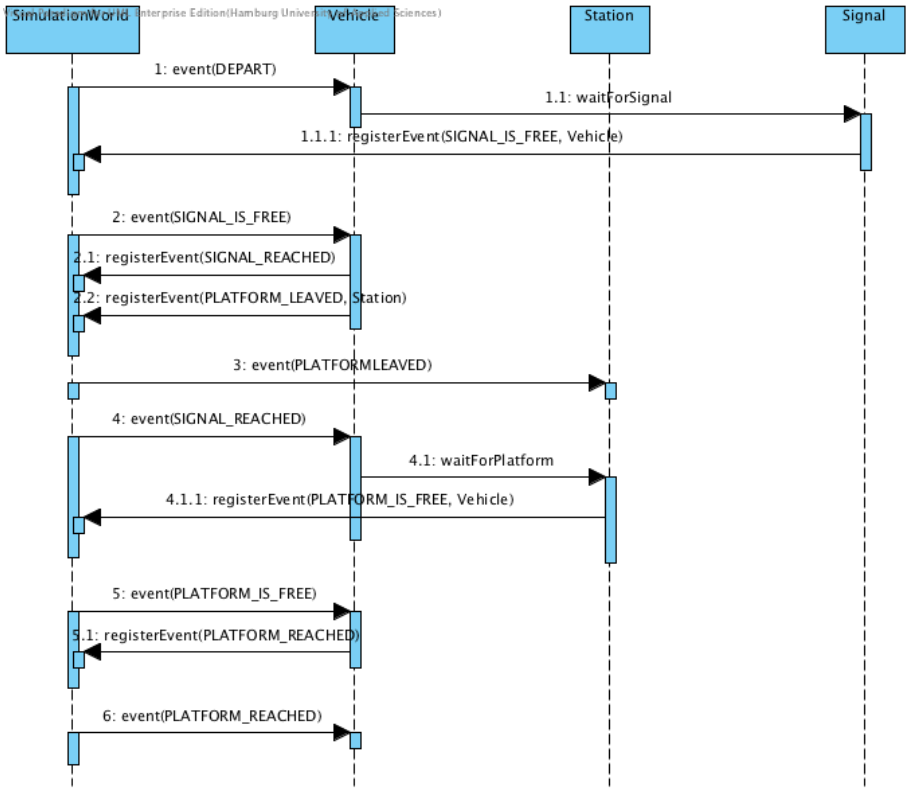

Abbildung 6.6: Fahrt einer Bahn

Signal. Auch wird ein Ereignis erstellt, welches den Bahnsteig nach einer gewissen Zeit wieder freigibt.
Dieses Ereignis wird an den Bahnhof geleitet, welcher hier weiteren Fahrzeugen Einlass gewähren könnte.
Das in Schritt 4 erreichte Signal ist das letzte Signal vor dem Bahnhof. Deshalb ist es nicht allein für die Strecke nach diesem Signal verantwortlich. Der Zug würde an diesem Signal warten, wenn der Bahnsteig belegt ist, damit das vorherige Signal keinen Zug einlässt. Die Verwaltung übernimmt jetzt aber der Bahnsteig an den die Bahn fahren wird.
Sobald die Plattform frei ist, fährt die Bahn ein. Hier könnte die Bahn ihre weitere Route planen oder ihre Fahrt beenden.

6.4 Verifikation

Um festzustellen ob eine Anwendung richtig funktioniert, sind Tests unerlässlich. Das Testen kann manuell erfolgen oder durch Testklassen die automatisch eine Reihe von Tests abarbeiten und die Ergebnisse gegenüber Vorgaben überprüfen. Letzteres hat den Vorteil dass die Tests immer gleich ablaufen.
Bei dem ImportTool lag die Hauptfehlerquelle im Import der Webseiten. Da zu den importierten Daten keine Vergleichsdaten, bis auf die Ausgangsquelle vorhanden sind, ist ein automatisches testen hier nicht möglich. Daher wurde hier weitestgehend auf automatische Tests verzichtet und eine manuelle Prüfung der Importierten Daten vorgenommen bzw. bei größeren Datenmengen Stichproben gezogen.
Da die Simulation auf Java basiert, konnte hier das JUnit-Test Framework genutzt werden. Hierbei wurden zwei Testarten verwendet. Die erste Art waren Klassentests. Hier wurden die Ergebnisse von Funktionsaufrufen auf Instanzen von Klassen getestet, die ohne große Abhängigkeiten zu anderen Klassen instantiierbar waren. Die zweite Art war das Testen gegen Interfaces. Diese Tests sind unabhängig von der Implementierung und simulieren Aufrufe von externen Systemen.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
BATrain	45,3 %	6.179	7.458	13.637
src	45,3 %	6.179	7.458	13.637
guiSystemComponent	13,5 %	506	3.251	3.757
guiLayerComponent	0,0 %	0	1.235	1.235
test	65,5 %	1.624	857	2.481
railSystemComponent	69,7 %	1.178	512	1.690
vehicleComponent	67,4 %	680	329	1.009
fassade	35,1 %	175	324	499
simulationComponent	40,6 %	191	280	471
PathfinderComponent	74,5 %	722	247	969
Main	74,9 %	652	219	871
time	64,9 %	307	166	473
Messages	79,1 %	144	38	182

Abbildung 6.7: Auswertung der Testabdeckung. Erstellt mit dem Eclipse-Plugin elcemma (http://www.eclemma.org/)

Die Testabdeckung ist gemessen nicht sehr hoch. Hierbei ist jedoch zu beachten, dass große Teile der Visualisierung nicht getestet wurden, da es sich um Zeichenoperationen handelt. Allein diese Komponenten machen über ein Drittel des Codes aus. Auch die Tests für Fassade wurden bewusst in diese Auswertung nicht mit aufgenommen, da sie die anderen Tests überdecken würden und schon als Anwendungstests eingestuft werden müssen.
Für den gesamt Test des System wurde ein einfaches Agentensystem geschaffen. Hierfür wird eine große Anzahl (1000-10000) von Agenten erzeugt und mit zufälliger Start- und Zielposition,

die sich so über das gesamte Stadtgebiet verteilen, versehen. Anschließend wird überprüft, ob jeder Agent in einer bestimmten Zeit sein Ziel erreicht.

6.4.1 Tests

Zuerst wurden Tests für die projekteigenen Datentypen entwickelt. Bei diesen Klassen wird erst überprüft, ob die im Konstruktor gesetzten Argumente korrekt beibehalten werden und ob die berechnenden Methoden korrekte Werte liefern. Hierzu werden bestimmte Parameter gewählt und das erwartete Ergebnis der Methode im Voraus berechnet. Mit diesen einfacheren Tests konnten so die Zeit- und Geopositionsklasse sowie die Klassen für die graphische Definition von Punkten abgedeckt werden.
Auch Datenstrukturen, wie der QuadTree und die TimeTable, werden auf diese Weise getestet. Da es sich hierbei nicht um unveränderliche Instanzen der Klassen handelt, wurden auch die Ergebnisse nach dem Aufruf von modifizierenden Methoden überprüft.
Die Komponententests beginnen im allgemeinen mit der Überprüfung der internen Klassen. Hier werden, wenn möglich, Methoden und Konstruktoren überprüft die keine breite Abhängigkeit zu anderen Klassen haben. Danach wird ein Szenario von interagierenden Klassen aufgebaut und dahingehend überprüft, ob der Aufbau korrekt umgesetzt wurde. Das Zusammenspiel der Klassen wird dann durch Methoden des Managers überprüft.
Da das Testen der Fahrzeugkomponente ohne einen Zeitplaner nicht ausführlich genug ist, wurde hier beim Testen eine Testklasse über das Interface der Simulations-Komponete implementiert. Diese überprüft, ob alle erzeugten Nachrichten in der richtigen Reihenfolge eintreffen und auch, ob diese überhaupt erzeugt werden. Auf diese Weise wird die komplette Erzeugung, Beendigung und Fahrt einer Bahn von einem Bahnhof zu einem anderen überprüft.
Für die Wegfindungs-Komponente wird eine eigne Strecken- und Fahrplanbeschreibung aus einer, eigens für die Tests erstellten XML-Datei geladen. Danach werden die Ergebnisse der Wegsuche überprüft. Bei diesem Test werden zwei Bahnen beschrieben, die zeitversetzt und unterschiedlich schnell fahren. Je nach Abfahrtzeit und Ziel ist ein Umsteigen nötig.
Die für den Test der Wegsuche verwendete XML-Datei wird auch verwendet um das Laden der Fahrplandaten zu überprüfen. Hierbei wird überprüft, ob alle Stationen geladen werden, ihre Namen und Positionen stimmen, der Zeitplan korrekt übernommen wurde und ob die Zuteilung der Linien korrekt übernommen wird.

7 Abschluss

7.1 Integration in die Masterarbeit

Der Simulationsteil dieser Bachelorarbeit wird als Java JAR Datei geliefert. Diese wird von der Masterarbeit als Bibliothek importiert. Mittels der zur Verfügung stehenden Methoden der Fassade wird eine Instanz der Simulation erzeugt. Anschließend registriert sich die Agentensimulation der Masterarbeit bei einem Zeitgeber und registriert danach die Bahnsimulation bei diesem Zeitgeber. Dieser Zeitgeber kann beliebig schnell eine virtuelle Zeit simulieren und sorgt dafür, dass beide Simulationen die gleiche Zeit haben. Wenn eine Bildschirmausgabe erwünscht ist, wird eine Ausgabe erzeugt, die die gewünschten Visualisationen aufnimmt und darstellt. Danach wird der Zeitgeber gestartet und die Gesamtsimulation nimmt ihre Arbeit auf.

7.2 Auswertung

In der Analyse wurde definiert welche Anforderungen bestehen. Im Entwurf wurden diese Anforderungen untersucht und ein Modell geschaffen, welches ihnen genügen soll. In der Implementation wurde dieses Modell schließlich realisiert. Während der Implementierungsphase wurden die initialen Anforderungen umgesetzt, es kamen neue hinzu und bestehende wurden abgeändert. Auch diese neuen und geänderten Anforderungen wurden im Entwurf berücksichtigt und implementiert. Am Ende konnten alle funktionellen Anforderungen umgesetzt werden.
Da sich die Masterarbeit noch in einem frühen Stadium befindet, ist eine endgültige Auswertung noch nicht möglich. Es wurde jedoch auf der Implementationsseite der Masterarbeit schon früh die Implementation diese Arbeit verwendet. Daher wurde schon sehr früh das Zusammenspiel der Agenten mit den Bahnen und die Nutzung der Wegsuche getestet. Gegen Ende dieser Bachelorarbeit wurden auch die Möglichkeiten der Visualisierung verstärkt genutzt. Die Integration verlief hierbei schnell und es traten nur wenige Probleme, die sich wiederum recht schnell lösen ließen, auf. So dauerte die erste Integration beider Projekte nicht einmal einen Tag, da die Interfaces klar definiert waren.

Die meisten Probleme ergaben sich aus dem verwendeten Framework Slick2D, da dieses nicht für mehrere Threads und auch nicht für mehrere gleichzeitige Bildschirmausgaben ausgelegt ist.
Diese Probleme ließen sich aber auch lösen und am Ende überwogen die Vorteile dieses Frameworks gegenüber den Nachteilen.

7.3 Diskussion und Ausblick

Erst im späteren Verlauf wird sich zeigen, ob die ausgearbeiteten Anforderungen an diese Arbeit den steigenden Anforderungen an die Masterarbeit genügen. Bei dieser Arbeit wurde bewusst auf die Parallelisierung der Simulation verzichtet, da Fehler im parallelen Betrieb nicht immer reproduzierbar und demnach schwerer zu finden sind. Die Option der Parallelisierung wurde aber in Hinblick auf Performanceprobleme nicht ausgeschlossen. Die Performance erwies sich jedoch für die gestellten Anforderungen als ausreichend. Sollte das zu simulierende Gebiet aber deutlich wachsen und auch andere Verkehrsmittel wie Busse in die Simulation mit aufgenommen werden, könnte dieses Thema wieder an Bedeutung gewinnen.
Weitere Verkehrsmittel welche den Schienenverkehr nutzten und auch nur von diesem bei der Fahrt beeinflusst werden, sollten sich schnell in die Simulation einfügen lassen. Bei Bussen und Straßenbahnen sind aber auch noch der Strassenverkehr, komplexe Ampelsysteme, mehrspurige Strassen etc. zu beachten. Hier müsste untersucht werden, inwiefern diese Faktoren den Fahrplan beeinflussen und ob das gewählte Simulationsmodell hier noch Anwendung finden kann.
Das Signalsystem ermöglicht jetzt schon einen kollisionsfreien Betrieb der Bahnen. Kurze Verspätungen lassen sich auf diese Weise gut abbilden. Das Gesamtsystem ändert sein Verhalten aber nicht entsprechend ab. Geplante Sperrungen von Strecken lassen sich zwar durch die Anpassung des Fahrplans simulieren, plötzlich auftretende Ereignisse, wie die Sperrung einer Station oder lokaler Stromausfall, verlangen nach Maßnahmen, die das Gesamtsystem beeinflussen. Passagiere werden in diesen Fällen alternative Routen wählen und die Bahngesellschaft kann Linien anhalten oder umleiten und andere Linien durch zusätzliche Fahrzeuge verstärken. Hier müsste untersucht werden, inwiefern sich dieses automatisieren lässt und wie viele dieser Maßnahmen noch manuell von den Betrieben getätigt werden.

8 Anhang

8.1 Interface Beschreibung

Im Folgenden werden die beiden Interfaces beschrieben, welche von externen Systemen genutzt werden können.

- Collection<Integer> getStations();
  Die Rückgabe von getStations ist eine Liste aller IDs der Bahnhöfe.
- Collection<Integer> getPlatforms(int id);
  Diese Methode liefert eine Liste aller IDs der Bahnsteige eines Bahnhofs zurück. Der Bahnhof wird mit dem Argument id festgelegt.
- Collection<Integer> getTrains();
  Mit getTrains wird eine Liste aller IDs der Bahnen zurückgegeben.
- String getStationName(int id);
  Zu der ID eines Bahnhofs liefert getStationName den Namen des Bahnhofs.
- String getTrainType(int id);
  Diese Methode liefert zu der ID einer Bahn den Typ der Bahn als String.
- int getPath(float fromLat, float fromLon, float toLat, float toLon, double walkingFactor, double startRadius, double endRadius);
  Mit getPath wird ein Pfad berechnet und die ID des Pfades zurückgegeben, sofern einer gefunden wurde. Die ersten vier Parameter beziehen sich auf die Start- bzw. Ziel-Koordinaten der Suche. Mit walkingFactor wird angegeben, wie Fußwege vermieden werden. Bei Werten größer als eins werden Fußwege zunehmend vermieden. Mit startRadius und endRadius wird bestimmt, in welchem Bereich um den Start bzw. das Ziel Bahnhöfe als Start- oder End- Bahnhof in Betracht gezogen werden. In jedem Fall

wird der nächste Bahnhof zum Ziel oder am Start mit aufgenommen, auch wenn dieser nicht im jeweiligen Radius liegen sollte.
Wird kein Pfad gefunden ist das Ergebnis -1

- void freePath(int id);
  Da sich die Wegfindung selber um die Verwaltung der Pfade kümmert, muss diese erfahren, wann ein Pfad nicht mehr benötigt wird. Mit dieser Methode wird ein Pfad anhand seiner ID aus der Verwaltung entfernt und steht nicht mehr zur Verfügung.

- int getPlatformForPath(int id, int stationID);
  Um das erste Mal einzusteigen bzw. beim Umsteigen wird mit dieser Methode die ID des Bahnsteigs geliefert, an dem die nächste Bahn zur Weiterfahrt hält. Hierbei wird mit id die ID des Pfades angegeben und mit stationID die ID des Bahnhofs.

- boolean checkTrainPathAtStation(int id, int trainID);
  Überprüft, ob die Benutzung eines Zuges, der an einem Bahnsteig steht, für den Pfad zulässig ist.

- boolean checkTrainPathForNextStation(int id, int trainID);
  Überprüft, ob ein fahrender Zug an der nächsten Haltestelle weiterhin für die Nutzung eines Pfades geeignet ist.

- IBasicWidget createMapGUI();
  Diese Methode sorgt für die Erstellung der Visualisierung und liefert diese als IBasicWidget zurück.

- public int getPathStart(int id);
  Gibt die ID des ersten Bahnhofs eines Pfades zurück.

- public int getPathEnd(int id);
  Gibt die ID des letzten Bahnhofs eines Pfades zurück.

- public ILayerManager getLayerManager(IBasicWidget gui);
  Zu einem vorher mit createMapGUI erstellten Visualisierung kann hiermit der Verwalter für die Ebenen abgerufen werden.

- public int addLayer();
  Hiermit wird eine weitere Ebene erstellt und die ID der Ebene zurückgegeben.

- public void addImageItem(int layerid, int itemid, float lat, float lon, String imagefilename);
  Mit addImageItem wird auf eine Ebene an einer Position, welche durch lat und lon festgelegt wird, ein Bild angezeigt. Geladene Bilder werden verwaltet und werden nicht erneut geladen, wenn sie sich schon im Speicher befinden. Mit itemid kann eine ID angegeben werden die das Objekt identifiziert.

- public void addComplexItem(int layerid, int itemid, float[] lat, float[] lon);
  Komplexe Objekte sind Polygone, welche eine Umrandung und eine Füllung besitzen. Komplexe Objekte können mit dieser Methode erstellt werden. Die Arrays lat und lon sollten die gleiche Anzahl von Elementen besitzen und beschreiben die Punkte des Polygons. Ein Polygon wird immer geschlossen. Daher wird automatisch eine Verbindung vom ersten zum letzten Punkt erstellt.

- public abstract void updateImageItem(int layerid, int itemid, String imagefilename);
  Um das Erscheinungsbild eines bestehenden Bildobjekts auf einer Ebene zu ändern, kann diese Methode verwendet werden.

- public void updateImageInfoText(int layerid, int itemid, String text);
  Diese Methode setzt den Infotext eines Bildobjektes. Der angegebene Text wird in der Visualisierung neben dem Objekt angezeigt. Die Angabe von „null“ als Textparameter entfernt den Text wieder.

- public void updateComplexInfoText(int layerid, int itemid, String text);
  Diese Methode arbeitet analog zu updateImageInfoText für Komplexeobjekte.

- public void updateLineColor(int layerid, int itemid, float r, float g, float b, float a);
  Um die Farbe der Umrandung eines komplexen Objektes zu ändern, muss diese Methode verwendet werden. Zusätzlich zu den Farbwerten von Rot, Grün und Blau lässt sich mit dem Parameter a die Transparenz der Umrandung definieren.

- public void updateFillColor(int layerid, int itemid, float r, float g, float b, float a);
  Diese Methode arbeitet analog zu updateLineColor. Hier wird jedoch die Fullfarbe eines komplexen Objektes beeinflusst.

- public void showLayer(int layerId);
  Hiermit wird eine Ebene angezeigt und an die oberste Zeichenebene geschoben.

- public void hideLayer(int layerId);
  Mit hideLayer wird eine Ebene ausgeblendet.

- void setDrawTrack(boolean value);
  Diese Methode bestimmt, ob die Bahnstrecken, Bahnhöfe und Bahnen auf der Karte angezeigt werden.

8.2 Abkürzungsverzeichnis

HTTP	Hypertext Transfer Protocol: Ein Netzwerkprotokoll zur Datenübertragung. Hauptsächlich im Internet eingesetzt.
JSON	JavaScript Object Notation: Im Klartext lesbares Datenformat.
HTML	Hypertext Markup Language: Auszeichnungssprache für Dokumente.
SQL	Datenbanksprache zur Modifikation von Datenbanken und zur Erstellung von Abfragen.
ADO.NET	„Bei ADO.NET handelt es sich um einen Satz von Klassen, die Datenzugriffsdienste für .NET Framework-Programmierer verfügbar machen.“ ([1])
API	Application Programming Interface: Schnittstelle für Anbindungen an ein System.
URL	Uniform Resource Locator: Lokalisation von (Netzwerk-)Ressourcen.
GUI	Graphical User Interface: Graphische Benutzeroberfläche.

Literaturverzeichnis

[geofoxline] : *Geofox Linienfahrplan.* – URL http://www.geofox.de/base/initLineSchedule.do

[hochbahn] : *Hochbahn Hamburg.* – URL http://www.hochbahn.de/wps/portal/de/home/hochbahn/unternehmen/fahrzeuge_technik/unternehmen_ubahnfahrzeuge/

[londonubahn] : *Live train map for the London Underground.* – URL http://traintimes.org.uk/map/tube/

[munchenbahn] : *S-Bahn München.* – URL http://s-bahn-muenchen.hafas.de/bin/help.exe/dn?tpl=livefahrplan

[swisstrain] : *SBB trains live.* – URL http://swisstrains.ch/

[Baldowski 2010] Baldowski, Mariusz: *Szenariobasierte Visualisierung von Geodaten*, Hochschule für angewandte Wissenschaften Hamburg, Bachelorarbeit, 2010

[Dessouky u. a. 2002] Dessouky, M.M. ; Lu, Q. ; Leachman, R.C.: Using simulation modeling to assess rail track infrastructure in densely trafficked metropolitan areas. In: *Simulation Conference, 2002. Proceedings of the Winter* 1 (2002), S. 725 – 731

[Dessouky u. a. 2004] Dessouky, M.M. ; Lu, Q. ; Leachman, R.C.: Modelling Train Movements Through Complex Rail Networks. In: *ACM Transactions on Modeling and Computer Simulation* 14, No. 1 (2004), S. 48–75

[HVV 2010] HVV: *HVV Bericht 2010.* 2010

[Kanacilo und Verbraeck 2007] Kanacilo, E.M. ; Verbraeck, A.: Assessing tram schedules using a library of simulation components. In: *Simulation Conference, 2007. Proceedings of the Winter* 1 (2007), S. 1878 – 1886

[Klauck und Maas 1999] Klauck, C. ; Maas, C.: *Graphentheorie und Operations Research für Studierende der Informatik.* Vorlesungsskript. 1999

[Kühnel 2010] Kühnel, A.: *Visual C# 2010 - Das umfassende Handbuch.* Galileo Computing, 2010

[Lorke 2012] Lorke, A.: *Die Mercator-Projektion - was genau versteht man darunter.* 2012. – URL http://www.gerhard-mercator.de/2012/02/22/die-mercator-projektion-was-genau-versteht-man-darunter/

[Nagel und Schreckenberg 1992] Nagel, K. ; Schreckenberg, M.: A cellular automaton model for freeway traffic. In: *Journal de Physique I* 2 (1992), S. 2221–2229

[Pachl 2011] Pachl, J.: *Systemtechnik des Schienenverkehrs.* Vieweg + Teubner, 2011

[Sarstedt 2009] Sarstedt, S.: *Software Engineering 1.* Vorlesungsskript. 2009

[Siedersleben 2002] Siedersleben, J.: *Moderne Softwarearchitektur.* 2. dpunkt.verlag, 2002

[Tatem u. a. 2006] Tatem, A.J. ; Rogers, D.J. ; Hay, S.I.: Global Transport Networks and Infectious Disease Spread. In: *ADVANCES IN PARASITOLOGY* 62 (2006)

[Vonhoegen 2007] Vonhoegen, H.: *Einstieg in XML.* Galileo Computing, 2007

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 19. Dezember 2012 David Seeger