Studienarbeit
Abteilung Informatik
Hochschule für Technik Rapperswil

Komplexe aber schnelle Analyse agentenbasierter Simulationsresultate

---

Autoren: Mirco Widmer, Matthias Zürcher

Betreuer: Prof. Dr. Luc Bläser

Projektpartner: Senozon AG, Zürich

21. Dezember 2012

Kapitel 1

Abstract

Die Firma `Senozon AG` ist auf die Erstellung und Auswertung von Verkehrs- und Verhaltenssimulationen spezialisiert. Bei einer Simulation werden sehr umfangreiche Datenmengen produziert, welche den Verkehrsfluss sowie Bewegungsdaten der beteiligten Agenten repräsentieren. Die Simulationsresultate werden für Verkehrsanalysen verwendet, was mehrfache sehr zeitaufwändige Aggregationen zur Folge hat.

Ziel dieser Arbeit ist es, grosse Mengen an agentenbasierten Simulationsdaten aufzubereiten und somit schnelle und interaktive Analysen zu ermöglichen. Dazu muss ein Framework zur effizienten Suche von Ereignisdaten entwickelt werden.

Es wurde ein Modell gebildet, welches Informationen über die einzelnen Ereignisse der Simulation liefert. Teil dieses Modells sind unterschiedliche Datenstrukturen, welche für den lesenden Zugriff optimiert sind. In einer Technologiestudie fand die Prüfung verschiedener Technologien auf ihren Einsatz statt. Als Datenspeicher wurde `Berkeley DB` verwendet und für die Visualisierung der Analysen wurde eine Webapplikation erstellt, die detaillierte Informationen über die Simulation liefert.

Die entwickelte Applikation `Traffic Simulation Analysis` ermöglicht den interaktiven Einsatz im Umfeld der Auswertung von Verkehrs- und Verhaltenssimulationen. Dies ermöglicht beispielsweise die Spinnenanalyse einer Hauptverkehrsachse innerhalb von 200 Millisekunden, ausgehend von 30 GB Simulationsdaten.

Kapitel 2

Management Summary

2.1 Ausgangslage

Komplexe Analysen der Verkehrs- und Verhaltenssimulation `MATSim` erfordern das Einlesen und Aggregieren verschiedener Simulationsresultate. Dies ist zeitaufwändig und ermöglicht bis jetzt keine interaktiven Analysen. In dieser Arbeit soll ein Weg gefunden werden, dies zu ermöglichen.

2.2 Vorgehen

In einer Technologiestudie prüften wir verschiedene potentielle Technologien auf ihren Einsatz. Nach der Erstellung eines Prototypen gewannen wir die Erkenntnis, dass eine Lösung nicht skaliert, welche die Datenmengen im Arbeitsspeicher hält. In der nachfolgenden Version setzten wir auf eine Lösung, welche Zwischenresultate aufbereitet. Mit dieser wurde die Grundlage für eine Webapplikation geschaffen, welche auf dem Persistenzlayer aufsetzt. In der Webapplikation wurde mit mehreren Showcases ein Teil des Systems demonstriert.

2.3 Ergebnisse

Die entwickelte Applikation `Traffic Simulation Analysis` ermöglicht das Einlesen von Resultaten der Simulationssoftware `MATSim` in eine für Abfragen optimierte Form. Mit Hilfe einer Webapplikation

können interaktive Abfragen über die Dimensionen `Zeit, Ort` sowie `Personen` getätigt werden. Die daraus generierten Ergebnisse werden visuell aufbereitet und für den Benutzer ansprechend dargestellt. Er erhält umgehend detaillierte Informationen über die von ihm verlangten visualisierten Simulationsresultate. Diese umfassen nebst grafischen Informationen über Aktivitäten auch Verkehrswege einzelner Agenten und erlauben somit interaktive Auswertungen.

2.4 Ausblick

Auf dem erstellten Datenmodell können verschiedene zusätzliche Szenarien entwickelt werden. Ausserdem sind vor allem technische Features wie beispielsweise eine Batch-Analyse oder Exportmöglichkeiten in Form von CSV-Files interessant.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Erklärung

Wir, Mirco Widmer und Matthias Zürcher, erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Rapperswil, 12.12.2012

Mirco Widmer

Matthias Zürcher

Inhaltsverzeichnis

Kapitel 3

Einleitung

Die Firma `Senozon AG` ist spezialisiert auf die Erstellung und Auswertung von Verkehrs- und Verhaltenssimulationen. Für die Simulationen wird vorwiegend die Software `MATSim (Multi-Agent Transport Simulation)` [1] verwendet. Diese liefert als Resultat einer erfolgreichen Simulation verschiedene Informationen über Ereignisse (`Events`) und zusätzliche Metadaten (Netzwerk- sowie Agenteninformationen). Diese Ergebnisse werden von Kunden zum Beispiel für die Planung von Verkehrsführungen genutzt.

Für die Analyse der teilweise umfangreichen Datenmengen wurde bis jetzt ein sequentieller Ansatz gewählt. Dieser hat mehrmalige Aggregationen der Event-Files zur Folge, um sinnvolle Antworten zu liefern. Eine typische Fragestellung wäre zum Beispiel: Welche Aktivitäten wurden in einer bestimmten Zeitspanne an einem bestimmten Ort von welchen Personen durchgeführt? Weitere Fragestellungen sind in der Aufgabenstellung (Kapitel 4) näher beschrieben.

In dieser Arbeit wird eine Datenstruktur aufgebaut, welche so generisch wie möglich ist, damit gewünschte Informationen aggregiert werden können. Dennoch darf unter dieser Generalisierung die Performance nicht leiden; genau diese soll durch die voraggregierte Datenstruktur steigen.

Herausforderungen in diesem Projekt sind folgende Punkte:

- grosse Datenmenge (30 GB)
  - Die Datenmenge besteht vor allem aus einem File, welches alle Bewegungen der Simulation aufzeichnet (näher beschrieben im Kapitel 5.1.3). Diese Menge dient als Basis für die Abfragen.
- Mittelweg zwischen generischer Struktur und Performance
  - Weitere Auswertungen sollen ohne grundlegenden Änderungen an der internen Datenstruktur erstellt werden können.
- Anforderungen an moderaten Ressourcenverbrauch
  - Arbeitsspeicherverbrauch soll zwischen 4 und 8 GB bleiben.
  - Festplattenspeicherverbrauch soll max. das 10-fache der Eingabedaten (szenarioabhängig) verwenden.

Kapitel 4

Aufgabenstellung

Nachfolgend ist die Aufgabenstellung sowie zusätzliche Rahmenbedingungen aufgeführt. Weitere Details bezüglich der Aufgabenstellung können dem Kapitel 11.2 entnommen werden.

Aufgabenstellung Studienarbeit für Mirco Widmer und Matthias Zürcher:

Komplexe aber schnelle Analyse agentenbasierter Simulationsresultate

1. Auftraggeber und Betreuer

Diese Studienarbeit findet in Zusammenarbeit mit der *Senozon AG* statt.

Ansprechpartner Auftraggeber:

- Dr. Michael Balmer, CEO
- Dr. Marcel Rieser, CTO

Betreuer HSR:

- Prof. Dr. Luc Bläser, Institut für Software

2. Ausgangslage

Gemäss der Beschreibung von Senozon, Marcel Rieser:

Die Senozon AG ist ein ETH Spin-off, spezialisiert auf Anwendungen mit der agentenbasierten Verkehrs- und Verhaltenssimulation MATSim (www.matsim.org). Diese Simulation generiert Resultatedateien (sogenannte Events-Files), die für grosse Szenarien mehrere Gigabyte gross sein können, selbst in komprimierter Form. Diese Daten sind von sehr einfacher Art und Weise. Für Simulationsanalysen sind die Daten deshalb praktisch immer in der einen oder anderen Form zu aggregieren. Bislang implementierte Analysen lesen jeweils sequentiell das komplette Events-File ein und extrahieren fortlaufend die gewünschten Daten. Komplexere Analysen können zusätzliche Datenquellen voraussetzen (z.B. Informationen zum Strassennetz, zu Personen, Gebäuden, Fahrplänen), oder mehr als einen Lesevorgang der Events benötigen. Entsprechend den Datenmengen und der oftmals sequentiellen Verarbeitung der Daten sind solche Analysen zeitaufwändig und können nicht zur interaktiven Analyse verwendet werden.

3. Ziele und Aufgabenstellung

Gemäss der Beschreibung von Senozon, Marcel Rieser:

Ziel der Studienarbeit ist es, Datenstrukturen und Algorithmen zu entwickeln, um einige vorgegebenen Analysen innert einer Zeitspanne zu berechnen, die auch den Einsatz in interaktiven

Analysetools ermöglicht. Zur Demonstration sollte eine einfache Webapplikation entwickelt werden, die für eine oder zwei Analysen eine interaktive Abfragemöglichkeit zur Verfügung stellt und die entsprechenden Resultate darstellt. Spezifische Anforderungen für Arten der Abfragen, Datenquellen und mögliche Lösungsrichtungen sind in der detaillierten Aufgabenbeschreibung von Senozon angegeben.

Folgende spezifische Ziele werden vorgegeben:

- Analyse der Anforderungen (Abfrageformen, Datenformate, Datenmengen, Zeiten etc.)
- Architekturstudie mit Machbarkeits-Prototypen für die massive und schnelle Datenanalyse. Evaluation existierender Technologien (z.B. aus dem NoSQL-Bereich wie Lucene, Hive etc.) und/oder eigener Ansätze.
- Entwurf eines Modells und einer Sprache für die Abfragebeschreibung (Selektion, Gruppierung, Aggregation etc. auf Agenten-Events).
- Ausarbeitung der Software-Architektur für die Anwendung.
- Entwicklung des Analyse-Tools als einfache Web-Anwendung (Abfrage-Eingabe, Analyse im Backend, Resultate-Darstellung). Technologiewahl in Absprache mit dem Auftraggeber und dem Betreuer.
- Verifikation des Tools mit automatisierten Tests und/oder System-Tests.

4. Zur Durchführung

Mit dem HSR-Betreuer finden in der Regel wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen. Besprechungen mit dem Auftraggeber werden nach Bedarf durchgeführt.

Alle Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und die Ergebnisse in einem Protokoll zu dokumentieren, das dem Betreuer und dem Auftraggeber per E-Mail zugestellt wird.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein vorläufiges Feedback. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Dokumentation.

5. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen (siehe https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html?&L=0). Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Die

Dokumentation ist vollständig auf CD/DVD in 3 Exemplaren abzugeben. Auf Wunsch ist für den Auftraggeber eine gedruckte Version zu erstellen.

6. Termine

Siehe auch Terminplan auf https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html?&L=0

17.09.12 Beginn der Studienarbeit, Ausgabe der Aufgabenstellung durch die Betreuer.

17.12.12 Die Studierenden senden folgende Dokumente der Arbeit per Email zur Prüfung an ihre Betreuer:
- Kurzfassung
- A0-Poster

Vorlagen sowie eine ausführliche Anleitung betreffend Dokumentation stehen unter den allgemeinen Infos Diplom-, Bachelor- und Studienarbeiten zur Verfügung.

21.12.12 Die Studierenden senden die vom Betreuer abgenommene und freigegebene Kurzfassung als **Word-Dokument** an das Studiengangsekretariat (cfurrer(at)hsr.ch).

21.12.12 Abgabe des Berichtes an den Betreuer bis 17.00 Uhr.

7. Beurteilung

Eine erfolgreiche Studienarbeit erhält 8 ECTS-Punkten (1 ECTS Punkt entspricht einer Arbeitsleistung von ca. 25 bis 30 Stunden). Für die Modulbeschreibung der Studienarbeit siehe https://unterricht.hsr.ch/staticWeb/allModules/19456_M_SAI.html

Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/5
2. Berichte (Abstract, Mgmt Summary, techn. u. persönliche Berichte) sowie Gliederung, Darstellung, Sprache der gesamten Dokumentation	1/5
3. Inhalt *)	3/5

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit festgelegt.

Im Übrigen gelten die Bestimmungen der Abt. Informatik zur Durchführung von Studienarbeiten.

Rapperswil, den 14. September 2012

Der verantwortliche Dozent

Prof. Dr. Luc Bläser
Institut für Software
Hochschule für Technik Rapperswil

Kapitel 5

Problemanalyse

5.1 Verkehrsmodell

Für die nachfolgende Beschreibung des verwendeten Verkehrsmodells ist ein gemeinsames Vokabular notwendig. Dieses ist im Kapitel 5.1.1 beschrieben. In den XML-Files sowie im Code werden diese Begriffe jeweils auf Englisch verwendet; deshalb werden sie auch in der Beschreibung dazu genutzt. Das verwendete Verkehrsmodell stammt aus der Simulationssoftware `MATSim` [1] und liefert die Datenbasis für das in dieser Arbeit entwickelte Abfragesystem. In einer Simulation, welche die Ausgangslage für eine Auswertung bildet, existiert ein Strassennetz (Abbildung 5.1). Es verbindet einzelne Nodes (Knoten) untereinander, auf welchen der Verkehr fliesst. Eine solche Verbindung wird als Link bezeichnet. Ein Verkehrsaufkommen auf diesen Links entsteht durch virtuelle Personen, sogenannte Agenten, welche von einer Aktivität (beispielsweise `Arbeiten` oder `Einkaufen`) zur nächsten unterwegs sind.

In einer `MATSim`-Simulation werden Szenariodaten (Kapitel 5.1.2) eingelesen, simuliert und Simulationsresultate (Kapitel 5.1.3) produziert. Diese werden hauptsächlich verwendet, um Analysen zu erstellen und bilden deshalb ein zentrales Element der Problemstellung.

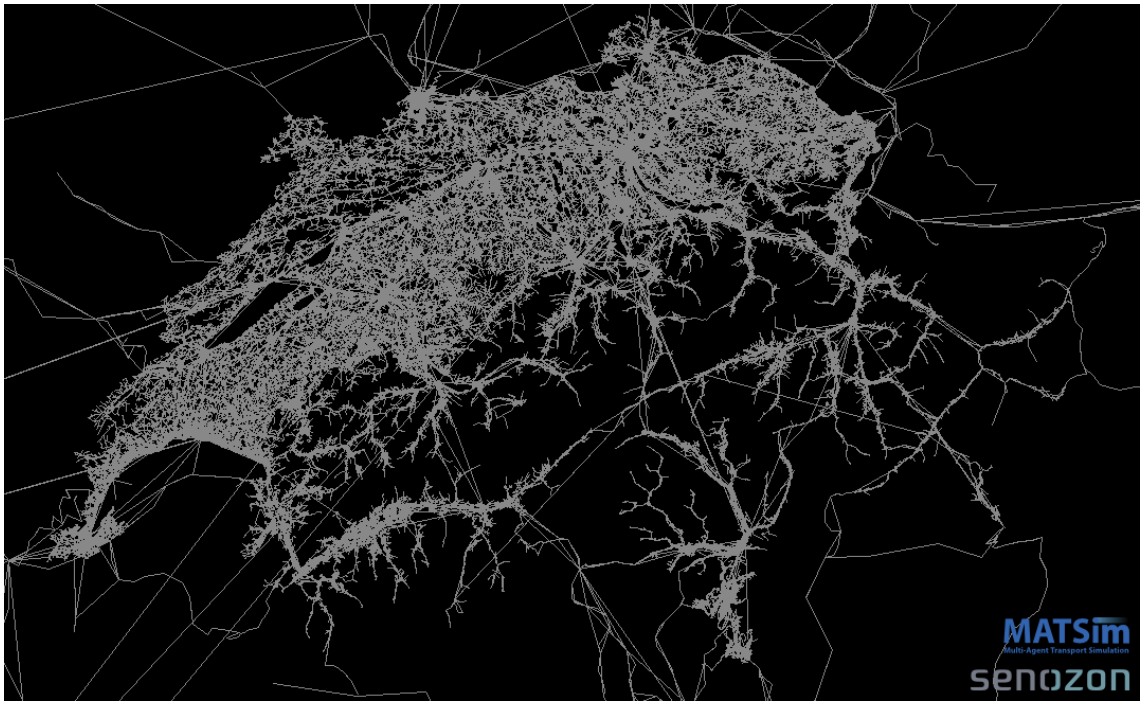


Abbildung 5.1: Beispiel für Verkehrsmodell der Schweiz

5.1.1 Begriffe

Node (Knoten) Knoten im Verkehrsnetz (Kreuzung/Strassenbeginn/Haltestelle)

Link (Kante) Gerichtete Verbindung von zwei Nodes (Strassenstück zwischen zwei Kreuzungen, richtungsgetrennt)

Ort Link, Kreuzung, Koordinatenpaar

Trip Weg zwischen zwei Aktivitäten, Zusammenfassung einzelner Events (beispielsweise der Weg von Zuhause zur Arbeit)

Leg (Abschnitt) Abschnitt eines Trips (beispielsweise Weg von Zuhause bis zur Bushaltestelle oder Fahrt mit dem Auto von A nach B)

Strassenbelastung Summe aller Trips pro Link

Person Agent mit verschiedenen Tagesaktivitäten

Aktivität Zeitraum, in welchem eine Person an einem Ort beschäftigt ist

Event Ereignis aus der `MATSim`-Simulation

Spinnenanalyse Trips aller Agenten, die in einem bestimmten Zeitraum an einem Ort vorbeikommen. Die Spinnenanalyse zeigt die Anfahrts- und Abfahrtswege im Bezug auf ein angegebenes Strassenstück. Dadurch kann beispielsweise eruiert werden, wo Fahrzeuge ausweichen würden, falls dieses Strassenstück gesperrt ist.

5.1.2 Szenariodaten

Die für die Analyse nicht notwendigen Attribute wurden aus den XML-Auszügen aus Gründen der Übersichtlichkeit weggelassen. Das Format wurde durch `MATSim` vorgegeben und wird zur Verständnisbildung zusammengefasst.

network.xml In der Datei `network.xml` sind die einzelnen Nodes beschrieben. Ein Node ist neben einer Id durch eine X-, sowie Y-Koordinate definiert:

```
<node id="1" x="1050" y="1050" />
<node id="2" x="2050" y="2950" />
```

Pro Link gibt es immer einen Start- und Endnode. Die Richtung ist ebenfalls berücksichtigt. Es gibt also beispielsweise zwei Einträge für die beidseitige Verbindung zwischen zwei Nodes:

```
<link id="12" from="1" to="2" length="2000.00" capacity="2000" modes="train" />
<link id="21" from="2" to="1" length="2000.00" capacity="2000" modes="train" />
```

population.xml In der Datei `population.xml` sind alle Aktivitäten aufgeführt, die von den Personen den Tag durch ausgeführt werden. Die Aktivitäten finden an unterschiedlichen Orten statt. Eine Startzeit für die jeweilige Aktivität ist nicht angegeben: Diese ergibt sich durch die Ankunft am Zielort. Falls auf einer bestimmten Strecke beispielsweise ein starkes Verkehrsaufkommen vorhanden ist, erreicht die Person den Zielort später und kann somit auch erst später mit der Aktivität beginnen. Als Ort, an welchem die Aktivität stattfindet, ist einerseits die Koordinate angegeben, andererseits ein Link. Bei diesem Link handelt es sich um den am nächsten zur genauen Koordinate gelegenen Link.

```
<person id="1" employed="no">
  <plan selected="yes">
    <act type="h" link="1112" x="890.0" y="685.0" end_time="07:18:16" />
    <leg mode="car" />
    <act type="w" link="2223" x="1802.0" y="2782.0" end_time="17:43:36" />
    <leg mode="car" />
    <act type="h" link="1112" x="890.0" y="685.0" />
  </plan>
</person>
```

transitSchedule.xml In der Datei `transitSchedule.xml` ist der Fahrplan abgebildet. Unter den `transitStops` sind alle öffentlichen Haltestellen mit der genauen Position aufgeführt sowie dem nächstliegenden Link (analog zu den Aktivitäten). Unter `routeProfile` ist die Abfolge von Haltestellen aufgelistet, die eine bestimmte `transitRoute` befährt. Im Abschnitt `route` sind alle Links aufgeführt, welche eine `transitRoute` effektiv befährt, um die Haltestellen zu bedienen. Unter `departures` ist zu sehen, um welche geplante Zeit die `transitRoute` an den jeweiligen Start-Haltestellen abfährt.

```
<transitSchedule>
  <transitStops>
    <stopFacility id="1" x="1050" y="1050" linkRefId= "11" />
    <stopFacility id="2a" x="2050" y="2940" linkRefId= "12" />
    <stopFacility id="2b" x="2050" y="2960" linkRefId= "32" />
    <stopFacility id="3" x="3950" y="1050" linkRefId= "33" />
  </transitStops>
  <transitLine id="Blue Line">
    <transitRoute id="1to3">
      <transportMode>train</transportMode>
      <routeProfile>
        <stop refId="1" departureOffset="00:00:00"/>
        <stop refId="2a" arrivalOffset="00:03:20" departureOffset="00:04:00"/>
        <stop refId="3" arrivalOffset="00:09:00" />
      </routeProfile>
      <route>
        <link refId="11"/>
        <link refId="12"/>
        <link refId="23"/>
        <link refId="33"/>
      </route>
      <departures>
        <departure id="01" departureTime="06:00:00" vehicleRefId="tr_1" />
        <departure id="02" departureTime="06:15:00" vehicleRefId="tr_2" />
        <departure id="03" departureTime="06:30:00" vehicleRefId="tr_1" />
        <departure id="04" departureTime="06:40:00" vehicleRefId="tr_2" />
      </departures>
    </transitRoute>
  </transitLine>
</transitSchedule>
```

5.1.3 Simulationsresultate

Das Hauptresultat der Verkehrssimulation `MATSim` [1] wird als XML-File `events.xml` abgespeichert. Die Struktur ist relativ trivial, denn sie ist durch zeilenweise aufgeführte Events definiert. Jeder Event geschieht zu einem bestimmten Zeitpunkt (Sekunden nach Mitternacht), ist von einem bestimmten Event-Typ und wird von einer Person auf einem Link durchgeführt. Falls ein solcher Event in Zusammenhang mit einer Aktivität (Start oder Ende) steht, dann ist zusätzlich das Attribut `actType` gesetzt, welches den Typ dieser Aktivität definiert.

```
<event time="0.0" type="actend" person="19" link="1121" actType="h" />
<event time="0.0" type="departure" person="19" link="1121" legMode="car" />
<event time="0.0" type="PersonEntersVehicle" person="19" vehicle="159" />
<event time="0.0" type="wait2link" person="19" link="1121" vehicle="159" />
<event time="1.0" type="left link" person="19" link="1121" vehicle="159" />
<event time="1.0" type="entered link" person="19" link="2122" vehicle="159" />
<event time="2.0" type="actend" person="39" link="1314" actType="h" />
<event time="2.0" type="departure" person="39" link="1314" legMode="car" />
<event time="2.0" type="PersonEntersVehicle" person="39" vehicle="39" />
<event time="5.0" type="wait2link" person="39" link="1314" vehicle="39" />
<event time="5.0" type="left link" person="39" link="1314" vehicle="39" />
<event time="5.0" type="entered link" person="39" link="1413" vehicle="39" />
```

5.2 Domainanalyse

Nachfolgendes Domainmodell ist vor allem darauf ausgerichtet, das `Events-File` und dessen Beziehungen darzustellen. Bei der Analyse ist das primäre Element ein einzelner Event. Dieser geschieht immer auf einem bestimmten Link beziehungsweise gehört zu einem bestimmten Link. Daneben beinhaltet ein Event immer auch eine Person, nämlich den Agent, welcher diesen Event verursacht hat. Diese drei Entitäten sind auch so im `Events-File` wiederzufinden.

Die Aktivitäten werden von genau einer Person ausgeführt. Die Beziehung zu einem Link ist analog zu der des Events. Eine Aktivität geschieht entweder direkt auf einem Link oder sie gehört rein geografisch zu einem Link.

Der Trip ist eine künstlich eingeführte Entität und ist nicht explizit in einem der Resultate-Files von `MATSim` zu finden. Ein Trip repräsentiert eine Aggregation von Events einer bestimmten Person über einen bestimmten Zeitabschnitt. Er beginnt mit dem Beenden einer Aktivität und endet mit dem Start einer nachfolgenden Aktivität. Die Beziehung zwischen Aktivität und Trip ist vorläufig nicht von Bedeutung und wird deshalb auch nicht modelliert.

Die Entität `AnalysisPassenger` steht in Verbindung mit dem öffentlichen Verkehr. Die Entität erweitert die `AnalysisPerson` um zusätzliche Attribute wie beispielsweise die Route, auf welcher der Passagier unterwegs ist oder das Fahrzeug, das momentan benutzt wird. Wenn eine Person auf verschiedenen Fahrzeugen verschiedene Strecken fährt, resultieren daraus mehrere `AnalysisPassengers`.

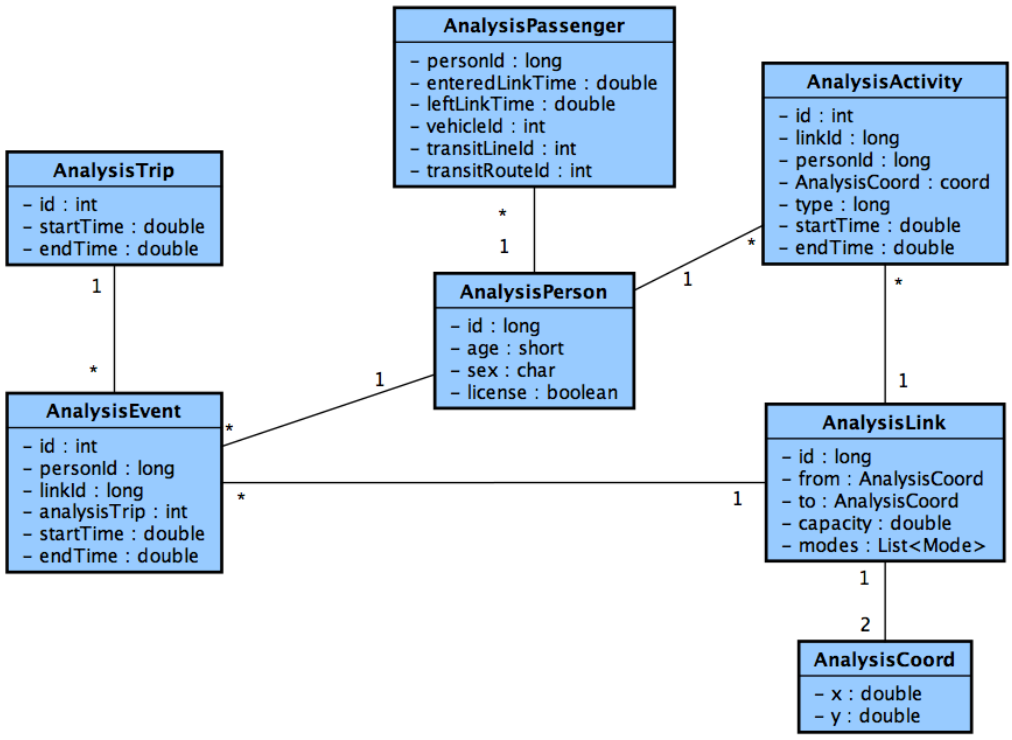


Abbildung 5.2: Domainmodell

5.3 Abfragen (Analyse)

Als Ansatz für ein Datenmodell, welches gewünschte Auswertungen vereinfachen und schneller machen soll, werden relevante Szenarien für die drei wichtigsten Typen von Auswertungen betrachtet. Aus diesen Szenarien soll eine Grundlage für das Modell gefunden werden.

5.3.1 Berechnung von Strassenbelastungen

Ziel

Bei der `Berechnung von Strassenbelastungen` wird aufgezeigt, zu welchem Zeitpunkt welche Links verkehrstechnisch wie stark belastet sind. In die Analyse miteinbezogen wird lediglich der Individualverkehr.

Verwendete Entitäten

`AnalysisLink`, `AnalysisTrip`, `AnalysisEvent`, `AnalysisPerson`, `AnalysisCoord`

Eingabeparameter

Die folgenden Parameter müssen bei einer Analyse für dieses Szenario angegeben werden:

- Link(s): Auflistung eines oder mehrerer Links
- Zeitraum: definiert durch Start- und Endzeit

Filterung

Die Resultate können nach dem Abrufen aus dem Datenspeicher nach verschiedenen Personenattributen gefiltert werden:

- Alter
- Geschlecht
- Fahrzeugausweis

Beispiele

- Strassenbelastung aller Links von 7 - 8 Uhr. Filter: männlich, unter 25 Jahren
- Strassenbelastung des Links X von 7 - 8 Uhr
- Spinnenanalyse
  Zeitraum: 7 - 8 Uhr
  Link: X
  Es interessiert, woher die Personen kommen und wohin sie unterwegs sind (ganzer Trip). Strassenbelastung aller Links, welche durch die erwähnten Trips besucht werden.

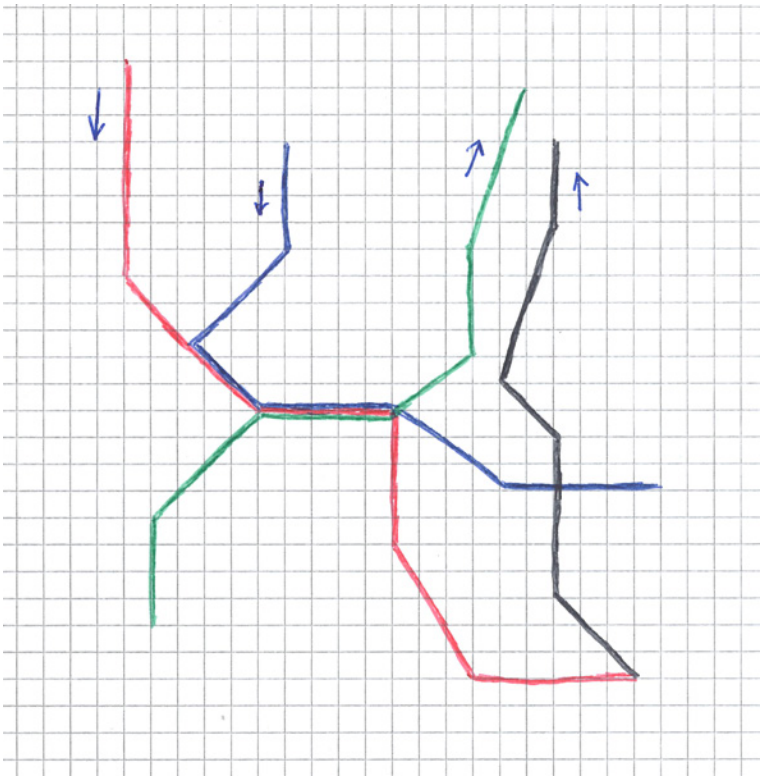


Abbildung 5.3: Modell der Spinnenanalyse

In der Abbildung 5.3 ist schematisch dargestellt, wie die Spinnenanalyse aufgebaut ist. Vier verschiedene Trips (mit unterschiedlichen Farben dargestellt) führen auf verschiedenen Links durch das Netzwerk. Bei der Spinnenanalyse interessiert beispielsweise der mittlere Link, durch welchen 3 Trips (rot, blau, grün) führen. Der Trip mit schwarzer Farbe ist hier nicht weiter interessant. Die Pfeile entlang der Trips markieren die Richtung.

5.3.2 Berechnung von ÖV-Belastungen

Ziel

Dieses Szenario hat sehr grosse Ähnlichkeit mit dem vorangegangenen Anwendungsfall - mit dem Unterschied, dass es sich bei den Verkehrsdaten um solche des öffentlichen Verkehrs handelt.

Verwendete Entitäten

`AnalysisLink`, `AnalysisEvent`, `AnalysisPerson`, `AnalysisPassenger`, `AnalysisCoord`

Eingabeparameter

Die folgenden Parameter müssen bei einer Analyse für dieses Szenario angegeben werden:

- Link(s): Auflistung eines oder mehrerer Links

- Zeitraum: definiert durch Start- und Endzeit

Filterung

Die Resultate können nach dem Abrufen aus dem Datenspeicher nach verschiedenen Personenattributen gefiltert werden:

- Alter
- Geschlecht
- Fahrzeugausweis

Beispiele

- Anzahl Studenten unter 25 Jahren, welche von 7 - 8 Uhr auf einem der Links (X, Y, Z) unterwegs sind.
- Alle berufstätigen Personen (beinhalten Tagesaktivität `Arbeiten`), welche auf einem der Links (X, Y, Z) unterwegs sind.

5.3.3 Berechnung von Anzahl Personen im Umkreis einer Koordinate

Ziel

In diesem Szenario soll ersichtlich werden, wo von welchen Personen ausgewählte Aktivitäten durchgeführt wurden.

Verwendete Entitäten

`AnalysisActivity`, `AnalysisPerson`, `AnalysisCoord`

Eingabeparameter

Die folgenden Parameter müssen bei einer Analyse für dieses Szenario angegeben werden:

- Mittelpunkt und Radius
- Zeitraum: definiert durch Start- und Endzeit

Filterung

Die Resultate können nach dem Abrufen aus dem Datenspeicher nach verschiedenen Personenattributen sowie Aktivitätstypen gefiltert werden:

- Aktivitätstyp
- Alter
- Geschlecht
- Fahrzeugausweis

Beispiele

- Anzahl Personen, welche zwischen 9 und 11 Uhr im Umkreis von 250m um Koordinate X die Aktivität `Einkaufen` ausüben.
- Anzahl Personen, welche zwischen 9 und 11 Uhr im Umkreis von 250m um Koordinate X die Aktivität `Einkaufen` ausüben. Person X beginnt um 8 Uhr und hört um 12 Uhr auf.

Kapitel 6

Design

6.1 Datenstrukturen

6.1.1 Konzeptionelle Abbildungen

Mit Hilfe von Hash-Strukturen sollen häufig abgefragte Werte schnell geliefert werden. Aufgrund verschiedener Beispielabfragen (näher beschrieben im Kapitel 5.3) wurde die folgende konzeptionelle Struktur gebildet. Als Eingabeparameter (Key) sind jeweils eine oder mehrere Eigenschaften der Rückgabewerte (Values) gesetzt. Der Aufbau dieser Datenstrukturen ist nicht zeitkritisch.

Nr	Key		Values
1	TimeSlice	$\longmapsto$	List of Activities
2	geografische Angabe	$\longmapsto$	List of Activities
3	Trip	$\longmapsto$	List of Events
4	LinkId	$\longmapsto$	List of Events
5	TimeSlice	$\longmapsto$	List of Events
6	LinkId, TimeSlice	$\longmapsto$	List of Events
7	LinkId, TimeSlice	$\longmapsto$	List of Passengers
8	PersonId	$\longmapsto$	List of Persons

Tabelle 6.1: Konzeptionelle Datenstrukturen

Synthetische Schlüssel

Verschiedene Werte wie zum Beispiel `linkId` oder `person` enthalten neben reinen Zahlenwerten auch alphanumerische Werte. Damit diese Werte nicht redundant in der Datenbank abgespeichert sind - was viel Speicher benötigen würde - haben wir eine Abbildung auf einen `long` Zahlenwert erstellt.

Die Abbildung ist in einer Textdatei gespeichert und wird vor der Analyse in den Arbeitsspeicher geladen, um einen schnellen Zugriff zu gewährleisten. Somit sind anstelle vieler Strings hauptsächlich `long` Werte in der Datenbank vorhanden. Der reale Wert wird

über die Abbildung geladen.

Wir haben uns entschieden, die häufigsten vorkommenden Werte wie folgt abzubilden:

- Aktivitättyp (Acttype)
- Link (LinkId)
- Person (PersonId)
- ÖV Fahrzeug (VehicleId)
- ÖV Linie (TransitLineId)
- ÖV Route (TransitRouteId)

6.1.2 Aufbau

Während dem Parsen des XML-Files wird eine interne Datenstruktur aufgebaut (intern startet jeweils mit dem Präfix `Analysis`). Ein `AnalysisTrip` startet beim `AgentDepartureEvent` und endet mit dem `AgentArrivalEvent`. Dazwischen geschehen unterschiedliche `AnalysisLinkVisitedEvent`. Dies entspricht einer Zusammenfassung eines `LinkEnterEvent` und dem entsprechenden `LinkLeaveEvent`. Somit kann die Anzahl Events, welche intern verwendet werden, beträchtlich reduziert werden.

Behandelte MATSim-Events

Folgende Events des XML-Files `events.xml` werden behandelt:

- `ActivityStartEvent`
- `ActivityEndEvent`
- `AgentDepartureEvent`
- `AgentArrivalEvent`
- `LinkEnterEvent`
- `LinkLeaveEvent`
- `TransitDriverStartsEvent`
- `PersonLeavesVehicleEvent`
- `PersonEntersVehicleEvent`
- `AgentStuckEvent`

In der internen Analyse-Datenstruktur werden aus den behandelten Events die folgenden Objekte aufgebaut:

- `AnalysisActivity`
- `AnalysisEvent`
- `AnalysisTrip`
- `AnalysisPerson`
- `AnalysisPassenger`
- `AnalysisLink`
- `AnalysisCoord`

In der Abbildung 6.1 ist dargestellt, wie die Events `LinkEnterEvent` und `LinkLeaveEvent` in interne `AnalysisLinkVisitedEvents` gemappt werden, die dann in der Analyse-Datenstruktur abgespeichert werden.

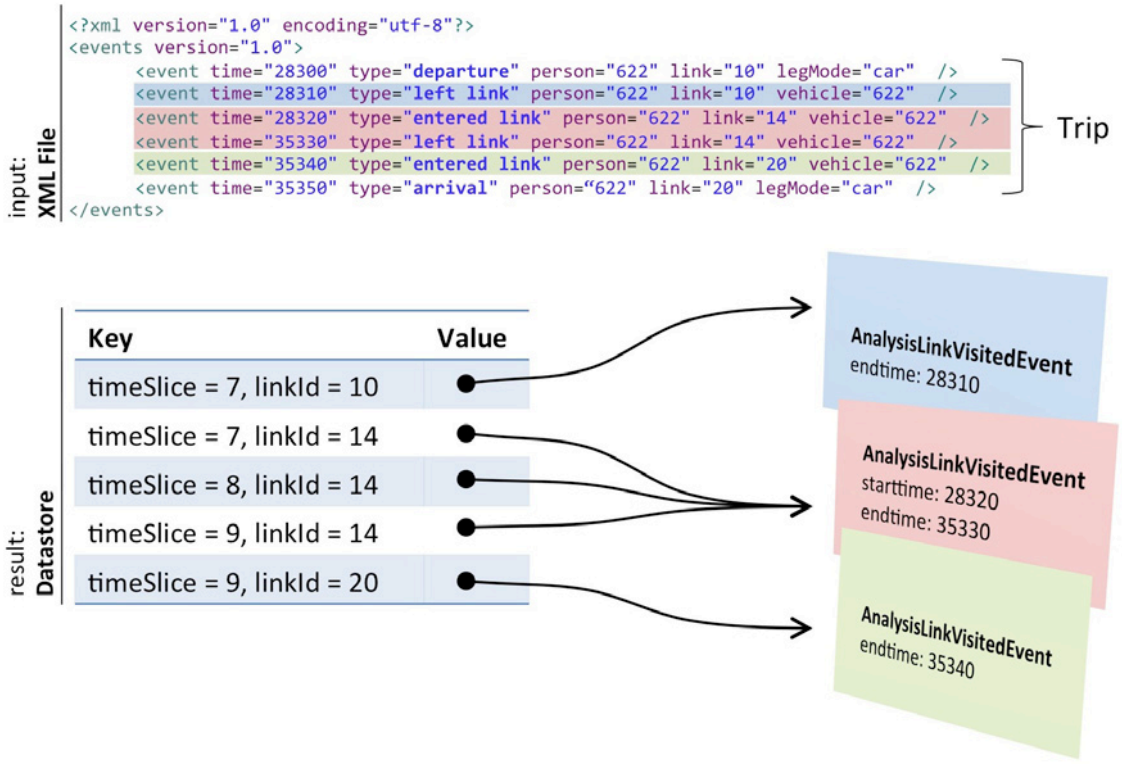


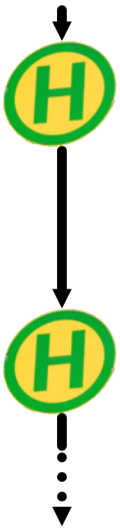
Abbildung 6.1: Aufbau AnalysisLinkVisitedEvent

Öffentlicher Verkehr

Die Verkehrssimulationsdaten umfassen zusätzlich zum Individualverkehr auch Daten zu den Fahrten von öffentlichen Verkehrsmitteln.

In der Datenbank sind pro Link und Zeit alle Passagiere gespeichert, welche sich in einem Fahrzeug befinden.

Sobald ein Passagier in ein Fahrzeug einsteigt, erfolgen bis zum Ausstieg keine weiteren Interaktionen mit der Person. Alle Aktionen beziehen sich dann auf das Fahrzeug. Damit die Passagiere für jeden befahrenen Link gespeichert werden können, müssen diese während einer Fahrt temporär in einer Liste gehalten werden. An jeder Haltestelle werden die aussteigenden Passagiere aus der Liste entfernt und die einsteigenden Passagiere hinzugefügt. Wenn dann ein Fahrzeug einen Link verlässt, werden alle Passagiere aus der temporären Liste in die Datenbank geschrieben. Dabei werden dem Passagier Fahrtdaten hinzugefügt. Dazu gehört Fahrzeugnummer, Liniennummer sowie die Identifikation der Route. Diese drei Eigenschaften identifizieren eine Fahrt mit dem ÖV eindeutig.



Event	Beschreibung	Aktion während Parsing
TransitDriverStart	Neue Fahrt beginnt	Alte Daten des Fahrzeugs werden falls vorhanden entfernt
PersonEntersVehicle	Passagier steigt ein	Passagier wird in die Liste der Passagiere des Fahrzeuges gespeichert.
LeftLink	Fahrzeug verlässt link	Die Passagiere aus der Liste werden in die Datenbank geschrieben.
EnterLink	Fahrzeug befährt link	Speichern der Zeit des Befahrens
PersonLeavesVehicle	Passagier steigt aus	Passagier wird aus Liste entfernt.
PersonEntersVehicle	Passagier steigt ein	

Abbildung 6.2: Parsen des öffentlichen Verkehrs

Zeitabschnitte Eine Fahrt in einem Link über mehrere Zeitabschnitte wird mehrmals pro Zeitabschnitt in der Datenbank erfasst.

6.2 Speicherverwaltung

Das Ziel der Technologiestudie ist es, geeignete Technologien für die Problemlösung zu evaluieren. Dazu werden Technologien und Tools auf verschiedene Kriterien hin getestet. Die Erfahrungen und Erkenntnisse mit diesen verschiedenen Technologien werden nachfolgend festgehalten.

Grundsätzlich gibt es zwei verschiedene Lösungsansätze. Der erstere basiert auf einer reinen In-Memory-Technologie. Es werden alle analysierten Daten im Hauptspeicher gehalten, somit kann bei der Abfrage die bestmögliche Performance erzielt werden. Ein alternativer Lösungsansatz ist das Persistieren der Analyse-Resultate. Nach der Analyse der Szenariodaten werden alle Ergebnisse persistiert und die nachfolgenden Abfragen setzen auf diesem Persistenzlayer auf.

6.2.1 In-Memory

Die Daten werden hierbei im Arbeitsspeicher gehalten. Mit Hilfe einer HashMap kann effizient auf einen bestimmten Value zugegriffen werden. Als weiterer Vorteil ist zu sehen, dass bei der Verwendung von HashMaps mit Referenzen gearbeitet werden kann. Die einzelnen Value Objects müssen nur einmal instantiiert werden, was erheblich ressourcenschonender ist als mehrere Value Objects mit denselben Werten in den Instanzvariablen.

Hashfunktion

Als Beispiel besteht das Value Object `AnalysisEvent` unter anderem aus folgenden Instanzvariablen:

```
private int id;
private long personId;
private long linkId;
private double startTime = -1;
private double endTime = -1;
private int analysisTrip;
```

Aus diesen lässt sich folgende Hashfunktion ableiten (generiert durch Eclipse). Sie verwendet alle Instanzvariablen zur Generierung des Hashcodes, was die Wahrscheinlichkeit für eine Kollision sehr stark reduziert.

```
@Override
public int hashCode() {
  final int prime = 31;
  int result = 1;
  result = prime * result + analysisTrip;
  long temp = Double.doubleToLongBits(endTime);
  result = prime * result + (int) (temp ^ (temp >>> 32));
  result = prime * result + id;
  result = prime * result + (int) (linkId ^ (linkId >>> 32));
  result = prime * result + (int) (personId ^ (personId >>> 32));
  temp = Double.doubleToLongBits(startTime);
  result = prime * result + (int) (temp ^ (temp >>> 32));
  return result;
}
```

Skalierung

Der Arbeitsspeicherverbrauch durch die Einträge in der HashMap wächst linear zu deren Grösse. In einem Test mit 1.6 Millionen Value Objects in der HashMap wurde ca. 2.5 GB an Arbeitsspeicher benötigt. Dies kann zu einem Problem werden, da je nach Szenario erheblich mehr Daten im Arbeitsspeicher gehalten werden müssen.

Performance

Die Zugriffsgeschwindigkeit auf Einträge der HashMap ist bei vielen Daten (1.6 Millionen Value Objects) im einstelligen Mikrosekundenbereich, was genügend performant für die Abfragen ist.

Fazit

Eine In-Memory-Lösung liefert zwar eine sehr hohe Performance, benötigt aber für grosse Szenarien zu viel Speicher. Aus diesem Grund ist eine Persistenzlösung nötig.

6.2.2 Key/Value Stores

Unter Key/Value Stores werden Datenspeicher verstanden, welche nicht über ein Schema definiert sind wie zum Beispiel relationale Datenkbanken. Key/Value Stores sind Speicher, welche aus einer Key-Spalte sowie einer Value-Spalte bestehen. Optimiert sind sie für den Zugriff über den Key. Es wird unterschieden zwischen In-Memory, On-Disk sowie Hybrid-Lösungen. [2] Aus Performance-Sicht wäre ein In-Memory-Key/Value Store optimal, jedoch muss getestet werden, wie gross die Datenmengen sind und ob diese komplett im Arbeitsspeicher zu halten sind. Falls nicht, ist es wichtig zu wissen, auf welche Daten oft zugegriffen wird und welche eher selten benötigt werden. So kann einfach entschieden werden, welche Daten auf die Disk geschrieben werden, wenn der Arbeitsspeicher knapp wird.

Memcached

Memcached [3] ist ein In-Memory-Key/Value Store, welcher als eigenständiger Prozess läuft. Darauf verbunden werden kann via Telnet oder Sockets. Um aus einem Java-Programm darauf verbinden zu können, benötigt es einen Memcached-Client.

Im Anhang unter Kapitel 11.1.1 ist beschrieben, wie Memcached benutzt werden kann.

Spymemcached Spymemcached [4] wird dafür eingesetzt, um aus einer Java-Applikation auf Memcached zuzugreifen.

```
MemcachedClient c = new MemcachedClient(new InetSocketAddress("127.0.0.1",
    11211));
c.set("key", 60, "value");
String value = (String)c.get("key");
```

Neben der Abfrage eines Values mittels Keys gibt es auch die Möglichkeit, eine ganze Anzahl Values abzufragen. Zu diesem Zweck übergibt man eine Collection mit allen Keys:

```
ArrayList<String> keys = new ArrayList<String>(Arrays.asList("a", "b"));
Map<String, Object> map = c.getBulk(keys);
```

Redis

Redis [5] ist ebenfalls ein Key/Value Store, welcher aber im Gegensatz zu Memcached einen zusätzlichen Persistenzlayer anbietet. In der aktuellen Version ist dieser Persistenzlayer (von Redis als `Virtual Memory` bezeichnet) jedoch nicht mehr unterstützt [6]:

> „Redis VM is now deprecated. Redis 2.4 will be the latest Redis version featuring Virtual Memory (but it also warns you that Virtual Memory usage is discouraged). We found that using VM has several disadvantages and problems. In the future of Redis we want to simply provide the best in-memory database (but persistent on disk as usual) ever, without considering at least for now the support for databases bigger than RAM. Our future efforts are focused into providing scripting, cluster, and better persistence.“

Fazit Key/Value Stores

Memcached und Redis hinterliessen einen sehr positiven Eindruck. Es sind beides sehr leichtgewichtige, simple Datenbanken mit einfachen Zugriffsmöglichkeiten. Bei Bedarf können sie auch verteilt auf verschiedenen Rechnern betrieben werden (vorgängig jedoch kein Thema). Leider bieten beide Key/Value Stores keinen Persistenzlayer und somit verunmöglicht dies den Einsatz aus den ähnlichen Gründen wie In-Memory-Lösungen.

6.2.3 Big Data Stores

Big Data Stores sind darauf optimiert, grosse Datenmengen zu verarbeiten. Hauptsächlich handelt es sich dabei um Klartext in verschiedenen Text- oder CSV-Dokumenten. Diese Files können durch die Verteilung auf verschiedene Systeme sehr schnell analysiert werden. Ein Anwendungsbeispiel ist die Websuche von Google, welche auf Big Data Stores aufgebaut ist.

Hive/Hadoop

Hive [7] ist ein auf Java basierendes Abfragesystem, welches auf dem Hadoop [8] Datawarehouse aufsetzt. Hadoop wurde von Google dafür entwickelt, grosse Textdateien mittels MapReduce Algorithmus zu verarbeiten.

Mit Hilfe einer JDBC-Hive Bridge können aus Javaprogrammen Abfragen auf die Daten durchgeführt werden. Die Abfragesprache ist an SQL orientiert. Daten können aber nur von einem Textfile eingelesen werden und nicht direkt in der Datenbank eingefügt, verändert oder gelöscht werden.

Testumgebung Die Firma Cloudera [9] bietet vorbereitete Virtuelle Linux-Images welche Hadoop und Hive bereits installiert und konfiguriert haben. Mit dem bei Hive mitgelieferten JDBC Treiber konnte so die Verbindung zwischen Java Code auf dem Hostrechner und Datenbank auf der virtuellen Maschine hergestellt werden.

Weitere Technologien

Da die Technologien Lucene [10], Cassandra [11] und HBase [12] mit der detaillierter betrachteten Hive/Hadoop-Lösung verwandt sind, werden diese nicht näher betrachtet. Gründe dafür sind folgende:

- keine verteilte Infrastruktur vorhanden/geplant
- keine riesige Datenmengen vorhanden, um Overhead zu rechtfertigen
- sehr hoher Initial-Aufwand, MapReduce-Algorithmen nicht optimal

Exemplarisch folgendes Zitat [2]:

> „Durch das Hybrid-Modell und die gute Skalierbarkeit bietet sich Cassandra für Situationen an, in denen große, sich ständig ändernde Datenmengen gespeichert werden sollen. Das beste Beispiel hierfür dürften die bereits genannten sozialen Netzwerke sein."

Fazit Big Data Stores

Hive eignet sich sehr gut, um grosse Textdateien auf verteilten Rechnern parallel zu bearbeiten. Dies wird durch den MapReduce Algorithmus ermöglicht. Daraus resultiert jedoch ein Overhead mit dem Starten der Jobs und dem Zusammenfassen der Ergebnisse. Dies zeigt sich dadurch, dass bereits einfache Abfragen mit wenigen Daten eine lange Initialisierungs- und Abarbeitungsphase aufweisen. Hadoop unterstützt `consistency` des `CAP Theorems` nicht, was für diese Arbeit nicht geeignet ist.

Dies führt uns wieder in die gleiche Ausgangslage, welche wir mit den grossen XML Dateien bereits haben. Deshalb sehen wir von dieser Lösung ab, da sie für uns keine Performanceverbesserungen bringt.

Die Analyse der Input-Daten muss sequentiell erfolgen, da die Reihenfolge der einzelnen Datensätze essentiell ist. Dies ist bei Big Data Stores nur bedingt möglich, da durch die Verteilung auf verschiedene Systeme die Sequentialität verloren geht. Auch die Vorteile von verteilten Systemen können wir in diesem Fall nicht nutzen, da eine Anforderung an die Anwendung ist, dass diese auf einem einzigen Rechner lauffähig ist.

Des Weiteren sind Big Data Stores auch zu schwergewichtig für diesen Anwendungsfall. Die Dateneinheiten sind hier lediglich kleine Events und keine grössere zusammenhängende Objekte wie beispielsweise Webseiten oder Textdokumente.

6.2.4 Berkeley DB

Hintergrund / Entstehung

`Berkeley DB` [13] war das Hauptprodukt der 1996 gegründeten Firma `Sleepycat Software` [14]. Die Entwicklung begann bereits 1991 an der Berkeley Universität in Kalifornien. Die

Grundlagen der Datenbank wurden in einem Bericht [15] an der USENIX Konferenz im Jahr 1999 vorgestellt. Im Jahr 2006 übernahm `Oracle` die Firma inklusive sämtlicher Produkte.

`Berkeley DB` unterliegt einem dualen Lizenzsystem. Für Open Source Projekte (unter GPL) sowie für den internen Gebrauch wird keine Lizenz benötigt. Wird `Berkeley DB` kommerziell in einer proprietären Software verwendet, muss dafür bei Oracle eine Lizenz gelöst werden.

Stärken

`Berkeley DB` bietet die Möglichkeit, grosse Datenmengen in einem Key/Value Store abzuspeichern. Die Daten werden stückweise auf der Festplatte gespeichert. Durchschnittlich ergibt sich dadurch eine Grösse von ungefähr 10 MB pro Datei. Die Daten sind indexiert und als B-Tree organisiert abgespeichert. Dadurch können auch Daten aus grossen Datenmengen besonders schnell und effizient geladen werden. Die Suche, das Einfügen, sowie das Löschen (in diesem Fall nicht relevant) beträgt `O(log n)`. Das Einfügen kann mitunter länger dauern, da je nach Value-Grösse viele Daten umkopiert werden müssen.

Überdies besitzt Berkeley einen LRU-Cache (Least recently used) welcher bei wiederholten Abfragen auf die gleichen Daten schnellere Zugriffe ermöglicht.

Einsatz

Für die Verwendung im Java Umfeld empfiehlt es sich, die `Berkeley DB Java Edition` [16] zu verwenden. Mit dieser Variante kann `Berkeley DB` als JAR File eingebettet werden. Das aktuelle Datenblatt von Oracle [17] beschreibt die Funktionalität und Einsatzmöglichkeiten von `Berkeley DB Java Edition` im Detail.

Die Datenbank kann zusätzlich für verschiedene Anwendungszwecke konfiguriert werden. Für diese Semesterarbeit wird die Konfiguration als `Data Store (DS)` verwendet. Die Gründe dafür sind, dass keine transaktionale Datenbank benötigt wird und die Daten nicht auf verschiedene Systeme repliziert werden müssen. Ausserdem besteht durch das einmalige Parsen der Daten ein WORM (Write once read many) Ansatz, welcher nach dem seriellen Einlesen keine Concurrency Probleme verursacht.

Um Objekte in die Datenbank zu persistieren, müssen diese mit der Annotation `@Persistent` versehen sein.

```
@Persistent
```

Daten werden durch die `put()` Methode in die DB geschrieben und mittels `get()` von der DB geladen. Ausserdem ist eine `putNoReturn()`-Methode vorhanden, welche die Daten einfügt und selbst keinen Rückgabewert hat.

Einschränkungen

Beim Verändern von Werten eines Schlüssels in der Datenbank gibt es keine Möglichkeit, Daten anzuhängen. Wird ein Wert verändert, muss dieser somit zuerst von der DB geladen und aktualisiert werden. Danach kann der alte Wert mit dem neuen überschrieben werden.

Da die Datenbank darauf spezialisiert ist, Daten schnell zu laden, organisiert sie sich jeweils beim Speichern neu. Diese Neuorganisation benötigt beim Einfügen von Daten zusätzliche Zeit, da diese Daten auf den Sekundärspeicher (Disk) gespeichert werden müssen. Die Priorität liegt in dieser Arbeit im schnellen Lesen der Daten. Auch wenn das Schreiben etwas länger dauert, sind die Zahlen im akzeptierten Bereich.

Berkeley DB und Memcached

In einem Test wurde `Berkeley DB` auf dem Betriebssystem installiert (im Gegensatz zu der Einbindung mittels JAR-File). Das Testsystem bestand aus folgender Konfiguration: System mit SSD, Intel Core i7 3615QM und 6 GB Arbeitsspeicher. Als Cache-Speicher wurde Memcached (mehr im Kapitel 6.2.2) eingesetzt und folgende Zugriffszeiten im Vergleich zu dem Zugriff ohne Memcached gemessen:

Test	**Berkeley DB**	**Berkeley DB + Memcached**
Key 0	2.98 ms	18.9 ms
Key 0 (wiederholt)	0.53 ms	0.8 ms
Key 1	0.24 ms	0.6 ms
Key 1 (wiederholt)	0.20 ms	0.4 ms
Key 10'000	0.82 ms	0.6 ms
Key 10'000 (wiederholt)	0.17 ms	0.4 ms

Tabelle 6.2: Performance-Vergleiche

Wir vermuten, dass durch die Verwendung von Memcached der `Berkley DB` Cache weniger bis gar nicht mehr genutzt wird. Dies resultiert in schnelleren Abfragen, erhöht aber den Initialaufwand (erster Zugriff auf Block). Der Performancegewinn ist aber in einem Bereich, welcher für diese Arbeit nicht relevant ist.

Der LRU Cache von `Berkeley DB` zeigt sich in den kürzeren Abfragezeiten in der Tabelle bei wiederholtem Zugriff.

6.2.5 Relationale Datenbanken

Relationale Datenbanken erlauben einen vielfältigen Einsatz mit einfachen und komplexen Tabellen. Weiter ist es möglich, komplexe Abfragen über mehrere Tabellen und Spalten zu machen.

Einen sehr grossen Teil an Funktionalität, welcher eine relationale Datenbank mit sich bringt, ist für den angestrebten Datenspeicher nicht anwendbar und auch nicht nötig. Exemplarisch seien hier Transaktionssicherheit, Locking oder weitere Mechanismen aufgeführt, welche teilweise zu Overhead und somit zu Performanceverlusten führen.

6.2.6 Protocol Buffers

Falls eine Lösung in Frage kommt, welche auf der Serialisierung von Datenstrukturen basiert, muss überlegt werden, ob es Optimierungsmöglichkeiten in diesem Bereich gibt. In einer im Rahmen dieses Projektes durchgeführten Machbarkeitsstudie wurde festgestellt,

dass mit `Protocol Buffers` [18] eine Performance-Steigerung von Faktor 10 - 15 gegenüber der traditionellen Objektserialisierung mit Java erreicht werden kann.

6.2.7 Designentscheid

Die Technologiestudie zeigte, dass eine Lösung, welche alle Simulationsdaten im Arbeitsspeicher halten kann, nicht umsetzbar ist. Aus diesem Grund musste eine persistente Lösung gefunden werden. `Berkeley DB` [16] zeigte sehr gute Resultate.

6.3 Webapplikation

Die Webapplikation soll folgende Möglichkeiten anbieten:

- Übersicht Szenario
- Kartendarstellung
- Interaktion mit Karte
- Eingabemöglichkeiten Analyseparameter

Als kritisches Element wird die Interaktion mit der Karte angesehen. Das Darstellen von Objekten sowie Zeichnen von Strassenverbindungen ist nicht trivial. Ziel ist es, eine Komponente einzusetzen, welche Unterstützung bei dieser Art von Kartenmodifikationen bietet. Mit der der `GMap` (`http://www.primefaces.org/showcase/ui/gmapHome.jsf`) der Komponentenbibliothek `Primefaces` [19] wurde eine Komponente gefunden, die genau diese Interaktionen anbietet. Es ist möglich, `Markers` auf die Karte zu setzen sowie `Polygone` und `Kreise` zu zeichnen. Da `Primefaces` [19] auf `JSF` [20] basiert, ist somit klar, dass die darunterlegende Java Server Technologie `JSF` ist.

Es besteht auch die Option, die `Google Maps API` zu verwenden, um die Karte einzubinden sowie Modifikationen auf dieser vorzunehmen. Der hohe Anteil an `Javascript-Code` ist jedoch als Nachteil gegenüber der Lösung mit `JSF` zu sehen.

Kapitel 7

Implementierung

7.1 Architectural Overview

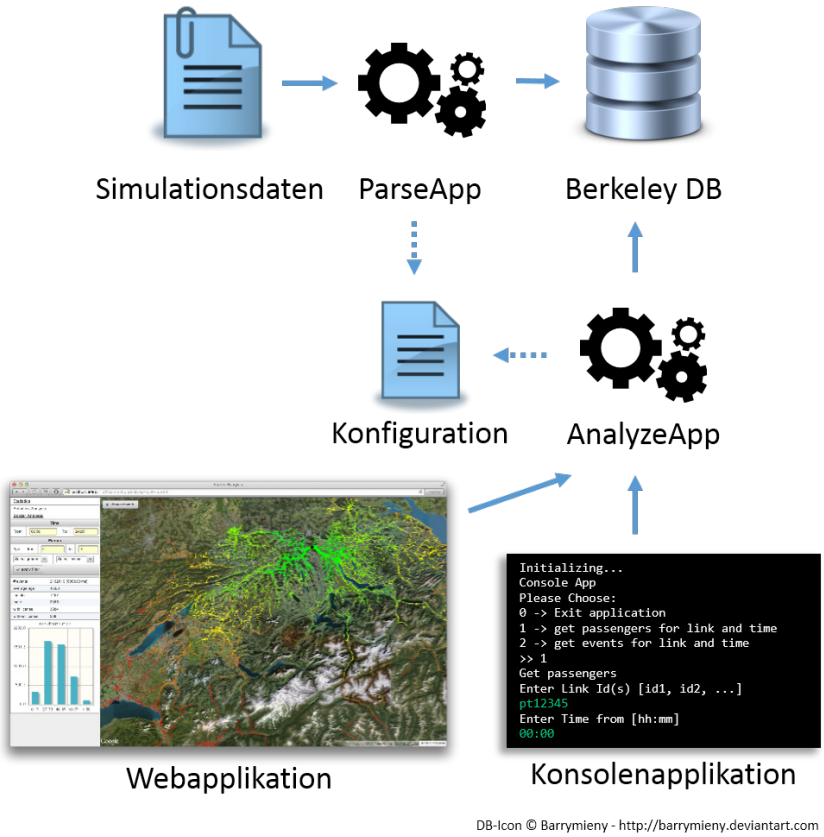


Abbildung 7.1: Architectural Overview

Die Simulationsdaten werden von der `ParseApp` in die `Berkeley DB` geschrieben. Über die `AnalyzeApp` beziehen die Webapplikation und die Konsolenapplikation die Daten aus der `Berkeley DB`.

7.2 Logical View

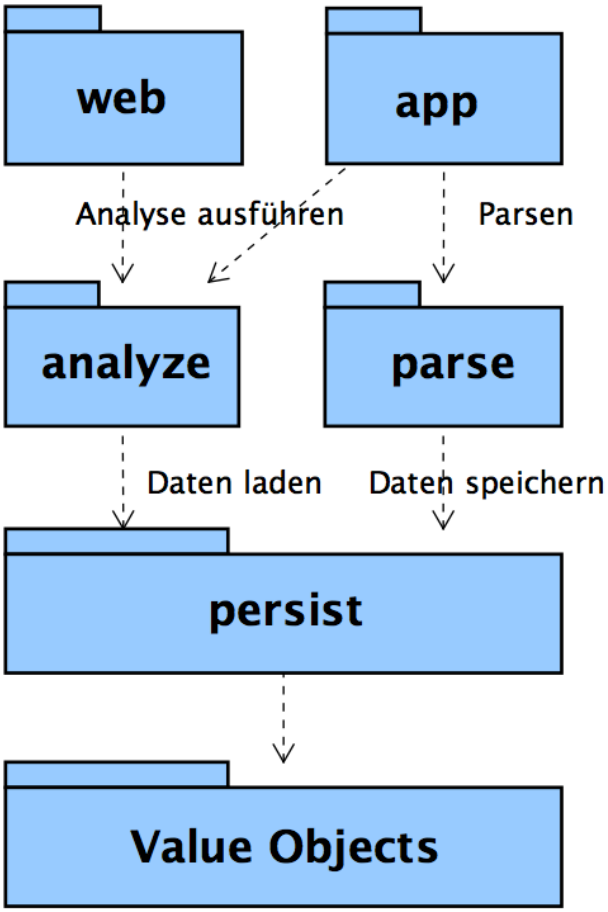


Abbildung 7.2: Logical View

In der `Logical View` zeigen wir, dass die Zugriffe innerhalb der Layer nur vertikal von oben nach unten verlaufen. Die Applikation ist gemäss einer Drei-Schicht-Architektur aufgebaut. Das grafische sowie das textbasierte User Interface bilden den Presentation Layer.

Der Business Layer ist in die beiden Teile `parse` und `analyze` aufgeteilt. `Parse` ist für die Verarbeitung der Simulationsdaten verantwortlich und wird ausgeführt, wenn neue Simulationsdaten vorhanden sind. Der `analyze` Teil wird dazu verwendet, um die verschiedene Analysen durchzuführen und die Daten dem grafischen Interface zur Verfügung zu stellen.

Als Datenschicht fungiert das `persist` Package. Es regelt sämtliche Verbindungen zur Datenbank. Weiter bietet es Methoden an, um Daten in die Datenbank zu speichern und von ihr zu lesen. Die Daten werden als entsprechende `Analysis` Objekte gehalten, welche sich im `Value Objects` Package befinden.

7.2.1 Übersicht

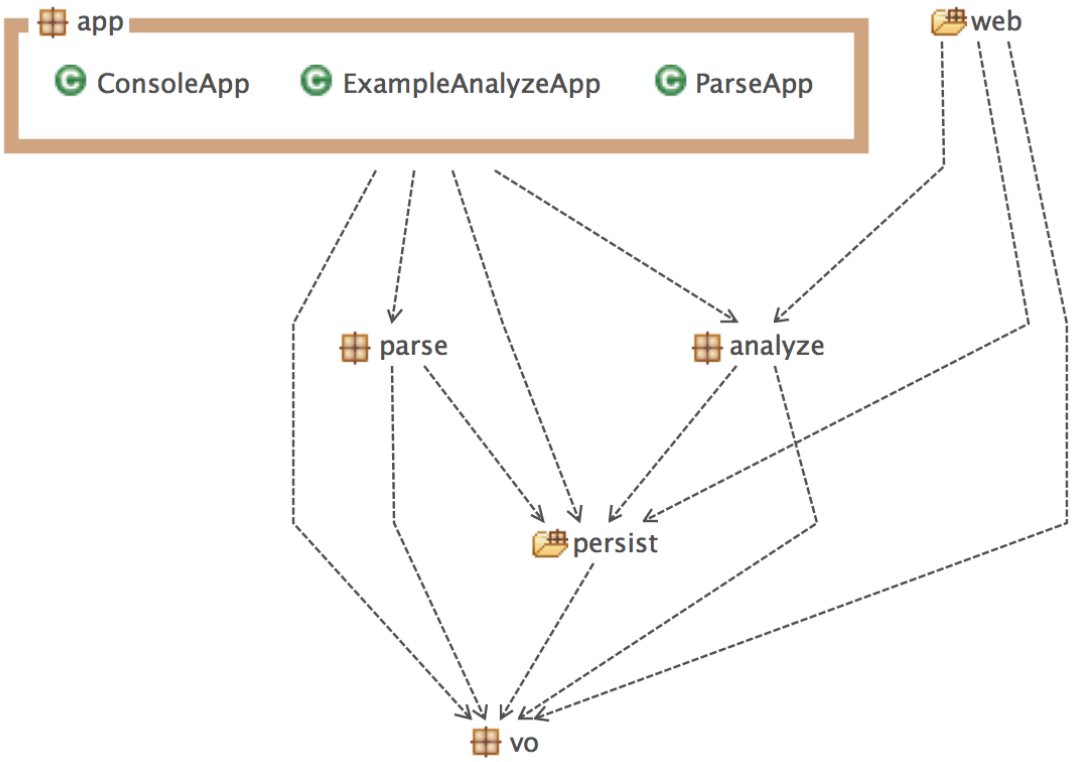


Abbildung 7.3: Übersicht Package-Struktur

In der Abbildung 7.3 ist dargestellt, wie die Umsetzung der einzelnen Packages ist. Dieses Diagramm ist mittels `STAN` [21] generiert und stimmt mit der Abbildung 7.2 überein. Die Abhängigkeiten aus dem Package `app` zeigen auf `parse` und `analyze`, welche wiederum schreibend beziehungsweise lesend auf das `persist`-Package zugreifen. Das Package `web` greift lediglich auf den Analyseteil zu und hat keine Abhängigkeiten zum Parseteil. Es existieren keine zyklischen Abhängigkeiten.

7.2.2 Package parse

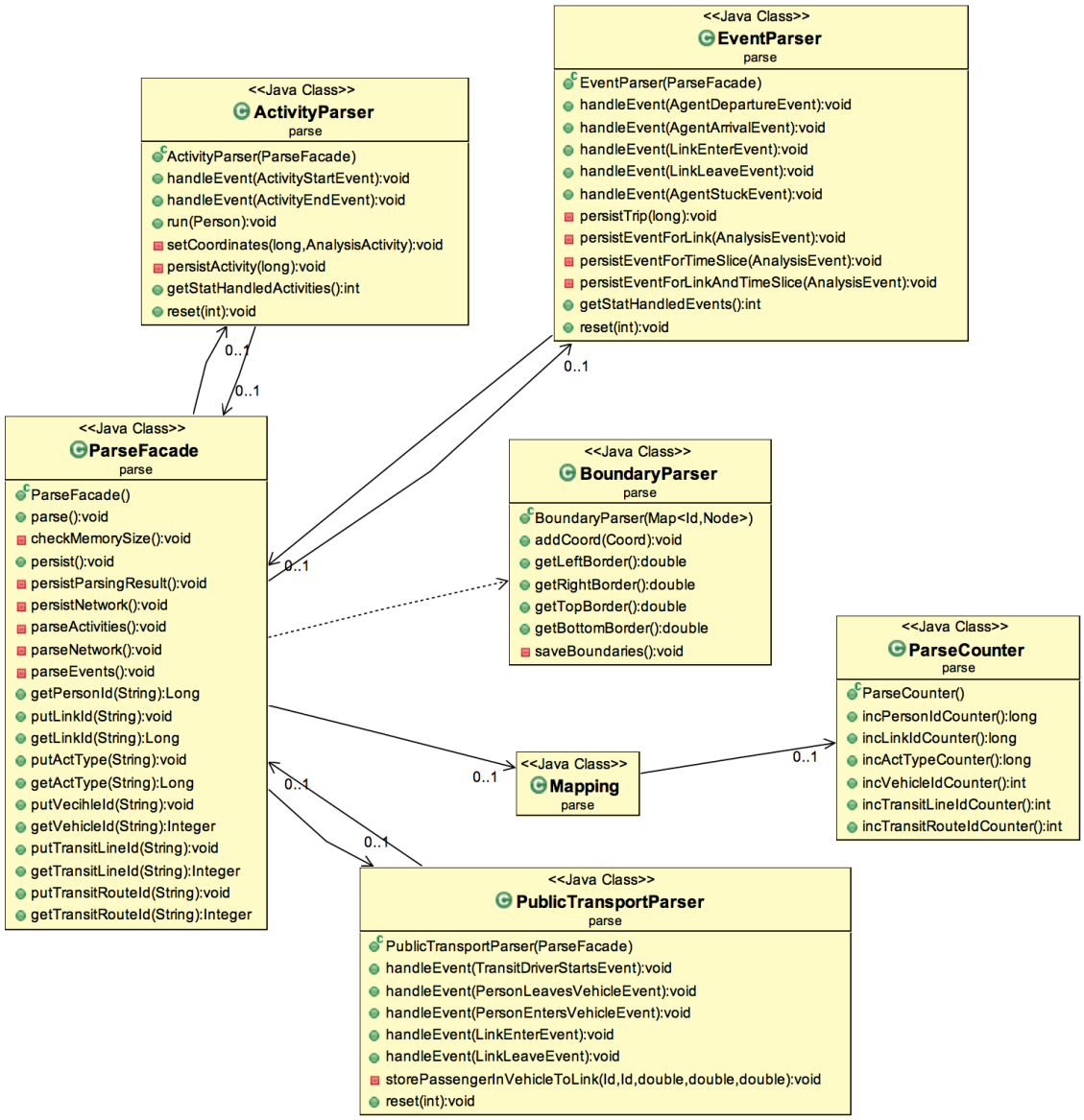


Abbildung 7.4: Aufbau des Packages parse

Als Hauptkomponente dient die Klasse `ParseFacade`. Aus dieser werden die verschiedenen Parser angestossen. Die einzelnen Parser werden als Listener an das Parsen des XML-Files angehängt. Dadurch können die einzelnen XML-Events verwendet werden für den Aufbau der internen Datenstruktur. Schematisch ist der Ablauf der Parse-Methode in der `ParseFacade` folgendermassen:

- checkMemorySize();

- parseNetwork();
- parseActivities();
- parseEvents();

Startparameter

Damit der virtuellen Maschine von Java genügend RAM zur Verfügung steht, muss als Startparameter ein grösserer Wert definiert sein als in einer Applikation normalerweise eingestellt ist.

Mit dem Parameter `-Xmx` wird der maximale RAM-Verbrauch definiert. Analog legt `-Xms` die Grösse des RAM fest, der beim Start der Applikation reserviert wird. Um den noch freien Platz im RAM von der Applikation her richtig berechnen zu können, sollten beide Parameter auf den gleichen Wert eingestellt sein.

Empfohlen wird ungefähr ⅔ des im System verfügbaren RAM festzulegen, mindestens jedoch 4 GB. Daraus resultiert für einen Computer mit 8 GB RAM folgende Konfiguration: `-Xmx5000m -Xms5000m`.

Je mehr RAM zugewiesen wird, desto schneller verläuft das Parsen. Falls jedoch mehr RAM zugewiesen wird als physikalisch vorhanden ist, kann sich dies negativ auswirken. Dabei wird ein Teil des RAM in eine Auslagerungsdatei geschrieben, wodurch sich die Dauer für das Parsen erheblich verlängern kann.

7.2.3 Package analyze

Die `Matcher` Klassen ermöglichen das Filtern der Resultate nach verschiedenen Kriterien.

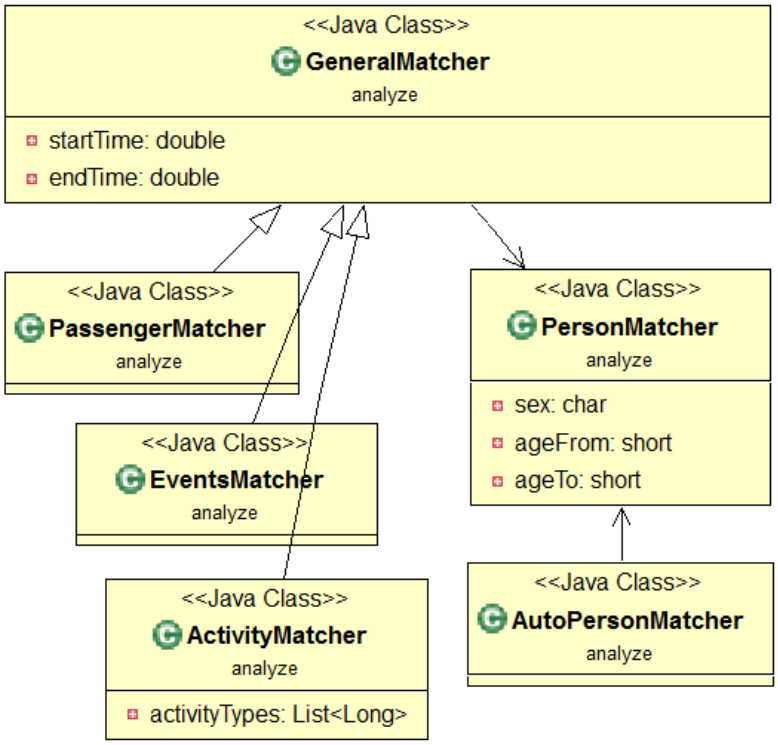


Abbildung 7.5: Aufbau des Packages analyze

Das Filtersystem basiert auf der `lambdaj` [22] Bibliothek. `Lambdaj` erlaubt es, einfach und effizient, Filter auf Listen anzuwenden. Ausserdem ist es möglich, mehrere Filter mit der "Und-" sowie "Oder-"Bedingung zu verknüpfen.

Zur Anwendung auf Simulationsdaten dient die Klasse `GeneralMatcher`, welche als Grundfunktionalität das Filtern nach Zeit und nach Personen unterstützt. Die Abgeleiteten Klassen erweitern diesen `GeneralMatcher` und können weitere, spezifischere Filter definieren.

Der `PersonMatcher` ist ein Spezialfilter um Personen, beispielsweise im `ActivityMatcher`, auf Basis Ihrer `Id` nach verschiedenen Attributen zu filtern. Dazu wird die `AnalysisPerson` während dem Filtern aus der Datenbank geladen und der `PersonMatcher` Filter darauf angewendet.

7.2.4 Package persist

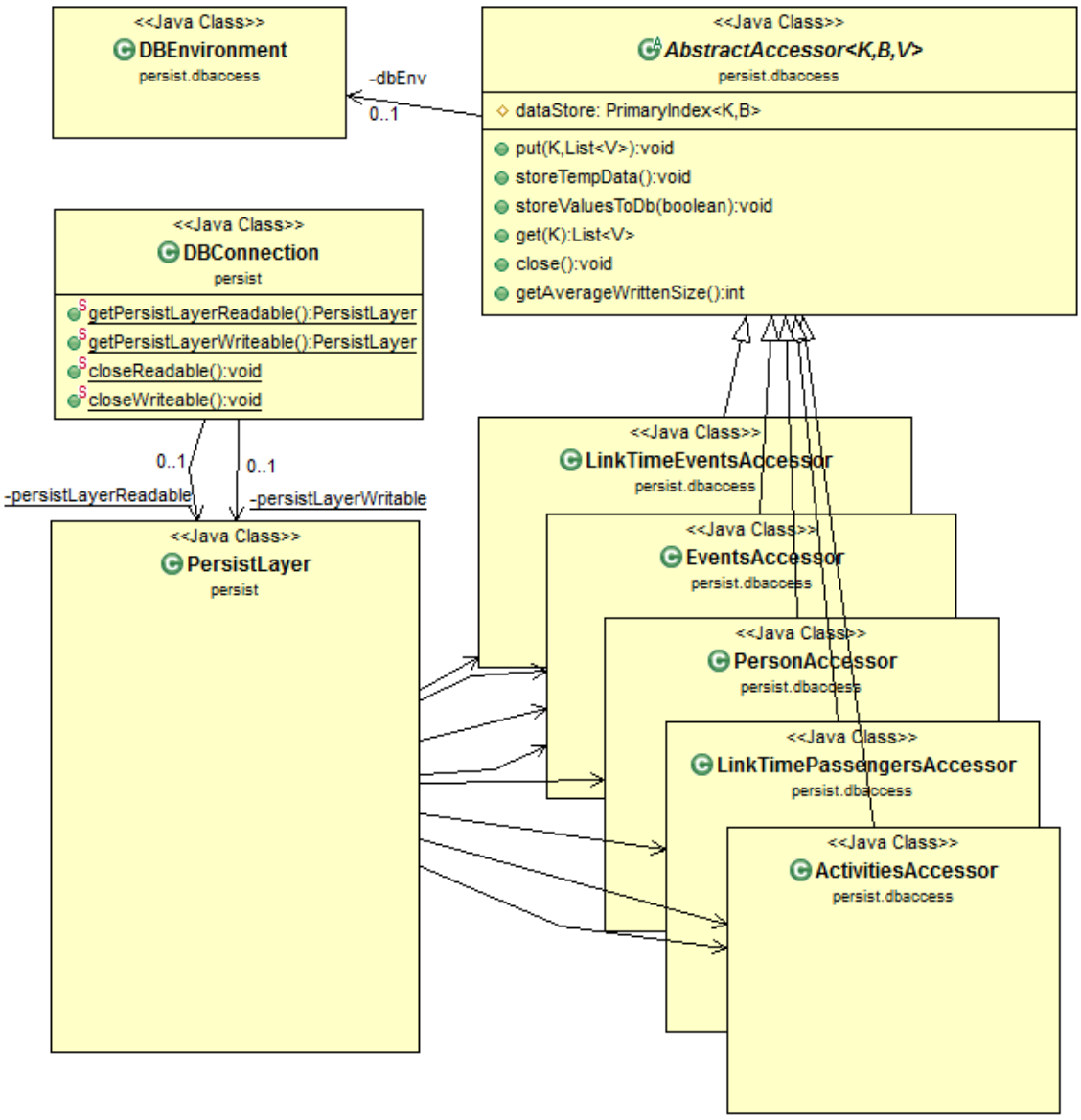


Abbildung 7.6: Aufbau des Packages persist

Die Klasse `PersistLayer` bildet die Schnittstelle zur Datenbankschicht. Programmweit gibt es nur eine Instanz dieser Klasse. Sie ist in der `DBConnection` Klasse gekapselt. Abhängig vom Zugriff auf die Datenbank (schreibend oder nur lesend), wird der entsprechende `PersistLayer` zurückgegeben.

Der `PersistLayer` besitzt Referenzen auf die verwendeten Data Stores. Pro Data Store gibt es eine Instanz von `AbstractAccessor` und `DBEnvironment`.

7.2.5 Package web

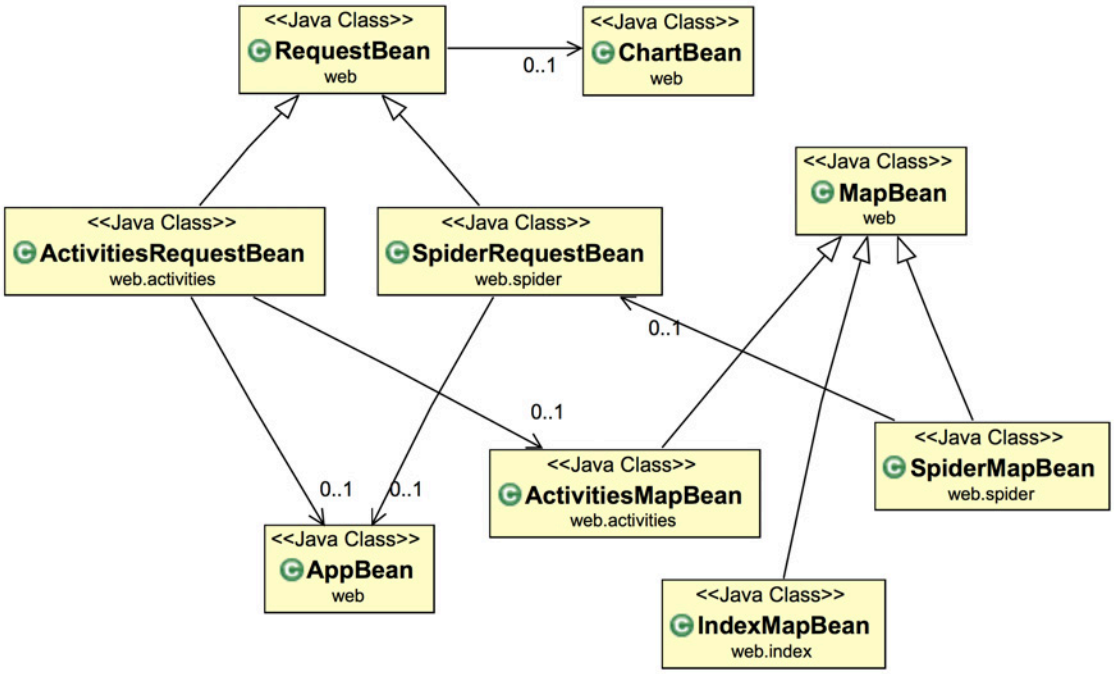


Abbildung 7.7: Struktur des Packages web

Die beiden wichtigsten Klassen im Package `web` sind `RequestBean` und `MapBean`. Diese sind allgemeine Klassen, welche jeweils die Basisfunktionalität zur Verfügung stellen. Aus den einzelnen Szenarien werden Erweiterungen dieser Klassen verwendet. Die sind jeweils durch den Präfix des Szenarios gekennzeichnet. Die Klasse `AppBean` stellt die Schnittstelle der Webapplikation mit dem `analyze`-Package dar. Alle Requests der Webapplikation werden über diese Klasse abgewickelt. Beim Activity-Szenario werden diese aus dem Formular abgeschickt, bei der Spinnenanalyse aus der Karte.

7.3 Deployment View

Nachfolgend sind die Lieferobjekte aufgelistet, welche bei einem Maven-Build [23] erstellt werden.

7.3.1 Lieferobjekte

- WAR-File (beinhaltet Webapplikation)
- Javadoc
- Cobertura Test Coverage Report

Ein JAR-File der Konsolenapplikation (beschrieben im Kapitel 6.1.2) wird nicht erstellt. Da für dieses Projekt keine grafische Oberfläche für den Teil des `Parsen` erstellt wurde, müssen die Szenario-Konfigurationen über einen Texteditor erstellt werden. Das WAR-File kann automatisch auf einen `Apache Tomcat` [24] ausgeliefert werden. Nach dem Starten des Webservers wird die Methode `contextInitialized()` ausgeführt, welche einzelne Applikationsteile initialisiert. Diese ist im Kapitel 7.5.3 detaillierter beschrieben.

In einem Test wurde das WAR-File auch auf einem Glassfish-Webserver [25] installiert und funktionierte identisch wie auf dem `Apache Tomcat`.

7.3.2 Buildprozess

Der Buildprozess wird mit Hilfe von Maven [23] automatisiert. Es wurde gewählt, da das `MATSim`-Projekt ebenfalls Maven für den Buildprozess verwendet. Zudem hatten wir wenig bis keine Erfahrung und wollten ein zusätzliches Buildtool neben `Apache Ant` [26] ausprobieren.

Abhängigkeiten

In Abbildung 7.8 sind die Abhängigkeiten aufgelistet, welche im `Project Object Model` (pom.xml) definiert sind. Diese Abhängigkeiten existieren alle aufgrund der verwendeten Bibliotheken (näher beschrieben im Kapitel 7.8).

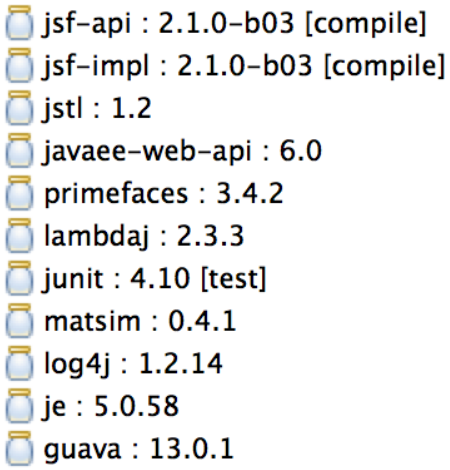


Abbildung 7.8: Project Object Model

7.4 Datenspeicher

7.4.1 Datenbankstruktur

Events und Activities sind in mehreren Kombinationen gespeichert. Die Liste mit den verwendeten Speicherstrukturen ist im Kapitel 6.1.1 detailliert beschrieben.

Der Value entspricht einer Liste von Events, Aktivitäten oder Passagieren. Mit den verschiedenen Speicherstrukturen werden verschiedene kleine Datenstücke geladen und mittels Union kombiniert. Dies ermöglicht eine bessere Performance, da es kein unnötiges Laden von Daten aus der Datenbank ins Memory benötigt.

Personen

Alle Personen sind mit einer 1:1 Abbildung zwischen synthetischer Personen Id und Personenobjekt gespeichert. Das Personenobjekt enthält weitergehende Informationen wie beispielsweise das Alter oder das Geschlecht.

7.4.2 Speicherobjekte (Buckets)

Für die Verwendung der `put()` und `get()` Methoden müssen speziell ausgeprägte Objekte verwendet werden. Diese Speicherobjekte nennen wir `Buckets`. Ein `Bucket` muss mit `@Entity` annotiert werden. Diese Annotation stammt aus `Berkeley DB`. Zusätzlich muss er eine Klassenvariable beinhalten, welche mit `@PrimaryKey` annotiert ist. Sie bezeichnet das Feld, welches als Schlüssel verwendet wird.

Ein `Bucket` kann beliebige Objekte und Werte speichern. Die einzige Voraussetzung besteht darin, dass die verwendeten Klassen mit `@Persistent` annotiert sein müssen.

7.4.3 Accessors

Es ist mit der Java Bibliothek von `Berkeley DB` nicht möglich, generische Typen als Schlüssel und `Buckets` zu definieren. Grund dafür ist, dass `Berkeley DB` zur Laufzeit von diesen Klassen Instanzen erzeugt. Von generischen Typen ist dies nicht möglich, weil Java diese zur Laufzeit nicht mehr kennt.

Durch diese Einschränkung wurde die Struktur mit einem abstrakten Accessor und verschiedenen konkreten Accessors aufgebaut. Die konkreten Accessors definieren die zu verwendenden Typen für den Schlüssel und den Wert. Gewisse Accessors können mehrmals verwendet werden, wenn die Struktur der gespeicherten Daten identisch ist.

Ein Accessor ist jeweils für einen Data Store zuständig. Er stellt die Verbindung zum Datastore her und übernimmt das lokale Zwischenspeichern der Daten. Die Funktionsweise des lokalen Zwischenspeichers ist im nächsten Kapitel beschrieben. Überdies erstellt er aus einem Schlüssel und Wert Paar ein `Bucket` und speichert diesen in die Datenbank.

Lokaler Zwischenspeicher

Einzelne Werte in die Datenbank zu schreiben ist zeitaufwändig. Vor allem, wenn mehrere Werte für einen Schlüssel vorhanden sind, weil dann wegen des Fehlens einer `append`-Methode immer alle Daten des Schlüssels aus der Datenbank geladen werden müssen. Ein Speichern der Daten in eine HashMap im Speicher ist wesentlich schneller.

Aus diesem Grund speichert eine `put` Operation auf dem `PersistLayer` die Werte in eine HashMap, solange genügend Speicher vorhanden ist. Wird eine Grenze an vorhandenem Speicher überschritten, löst der `PersistLayer` das Speichern der temporären Daten in die `Berkeley DB` aus.

7.4.4 Datenbankkonfiguration

In der Klasse `DBEnvironment` ist die Konfiguration der Datastores hinterlegt. Es wird dabei der Ablageort der Daten angegeben und festgelegt, ob Daten nur gelesen oder auch geschrieben werden können. Des Weiteren ist es möglich, verschiedenste Konfigurationsparameter festzulegen. Ein Beispiel dafür ist das Festlegen der maximalen RAM Grösse.

7.4.5 Kartenmodell

Das Zählen der Aktivitäten soll über einen geografisch bestimmten Bereich möglich sein. Dieser Bereich kann verschiedene geometrische Formen annehmen. Dabei wird über alle Aktivitäten iteriert, so dass festgestellt werden kann, ob sich eine Aktivität in einem bestimmten Bereich befindet oder nicht. Um nicht über die gesamten Aktivitäten iterieren zu müssen, ist die Karte in virtuelle Sektoren aufgeteilt. Ein Sektor hat die Form eines Quadrates. Beim Auslesen der Daten werden dadurch nur die Aktivitäten aus den entsprechenden Sektoren geladen. In der Szenario-Konfiguration wird angegeben, wieviele Sektoren in der Breite erstellt werden. Da die Sektoren quadratisch sind, ergibt sich die Anzahl Sektoren in der Höhe automatisch.

Sektorenanzahl festlegen

Eine gute Wahl der Sektorenanzahl hängt von der Anzahl der Aktivitäten und vom Netzwerk des Szenarios ab. Um eine optimale Performance aus der Sektoraufteilung zu gewinnen, sollte ein Sektor zwischen mehreren hundert und 2-3 Millionen Aktivitäten enthalten. Werden mehr als 5 Millionen Aktivitäten pro Sektor gespeichert, kann es in der `Berkeley DB` zu Problemen beim Speichern kommen, da der Value des Sektors zu gross ist.

Um genügend kleine Werte zu erreichen, muss beachtet werden, wie breit das Netzwerk der Links und wie gross die Streuung der Aktivitäten ist. Bei einem grossen Netzwerk, bei welchem alle Aktivitäten dicht in der Mitte angesiedelt sind, ist die Anzahl der Sektoren grösser zu wählen, als wenn die Aktivitäten breit gestreut im ganzen Netzwerk verteilt sind.

Tendenziell wird empfohlen, eine Sektoranzahl zwischen 100 und 300 zu wählen (abhängig von der Netzwerkgrösse). Diese Anpassungen können im szenario-abhängigen Konfigurationsfile getätigt werden:

```
sectorwidthdelimiter=100
```

7.4.6 Netzwerk

Das Netzwerk umfasst alle Links des Szenarios. Es wird benötigt, um in der Webapplikation die Links darzustellen. Um die Links genügend schnell zeichnen zu können, wird das ganze Netzwerk beim Start der Webapplikation in den Arbeitsspeicher geladen. Die Grösse des Netzwerkes beträgt bei grossen Szenarien ungefähr 150 MB.

7.4.7 Statistik

Die statistischen Daten, welche während dem Parsen erhoben und in der Webapplikation angezeigt werden, sind als serialisiertes Objekt pro Szenario abgelegt.

7.5 Webapplikation

Ziel der Webapplikation ist die Abbildung von zwei Showcases. Es soll für den Benutzer möglich sein, einige ausgewählte Abfragen zu tätigen und Informationen über die Simulationsresultate von `MATSim` zu erhalten. In Absprache mit `Senozon AG` wurden folgende Szenarien umgesetzt:

- Darstellung von Aktivitäten sowie deren Metadaten
- Grafische `Spinnenanalyse` eines bestimmten Links

In beiden Szenarien ist es möglich, nach folgenden Dimensionen zu filtern:

- Zeit
- Ort

- Personen

Die Filterung geschieht teilweise (Zeit, Ort) innerhalb der Abfrage auf den Datenspeicher (7.4), bei den Personen jedoch erst auf den Resultaten der Abfrage.

7.5.1 Komponenten

Im Folgenden sind die unterschiedlichen Komponenten (vorwiegend HTML-Seiten mit den dazugehörenden Backing Beans) beschrieben.

Statistics

Diese Seite (Abbildung 7.9) fungiert als Einstiegsseite und liefert einige Informationen über den Datenzustand. Auf der linken Seite ist zu sehen, um welches Szenario es sich handelt zusammen mit einigen Metadaten zum Szenario sowie dem Parse-Vorgang. Der rot markierte Bereich stellt dar, in welchem geografischen Bereich Netzwerk-Daten vorhanden sind (weitere Details unter 7.4.6). Dieses Beispiel ist ein Szenario der Schweiz, welches jedoch Zugverbindungen beinhaltet, die sogar fast bis nach Moskau reichen.

Die Informationen stammen aus einem serialisierten Objekt, welches im Kapitel 7.4.7 näher beschrieben ist.

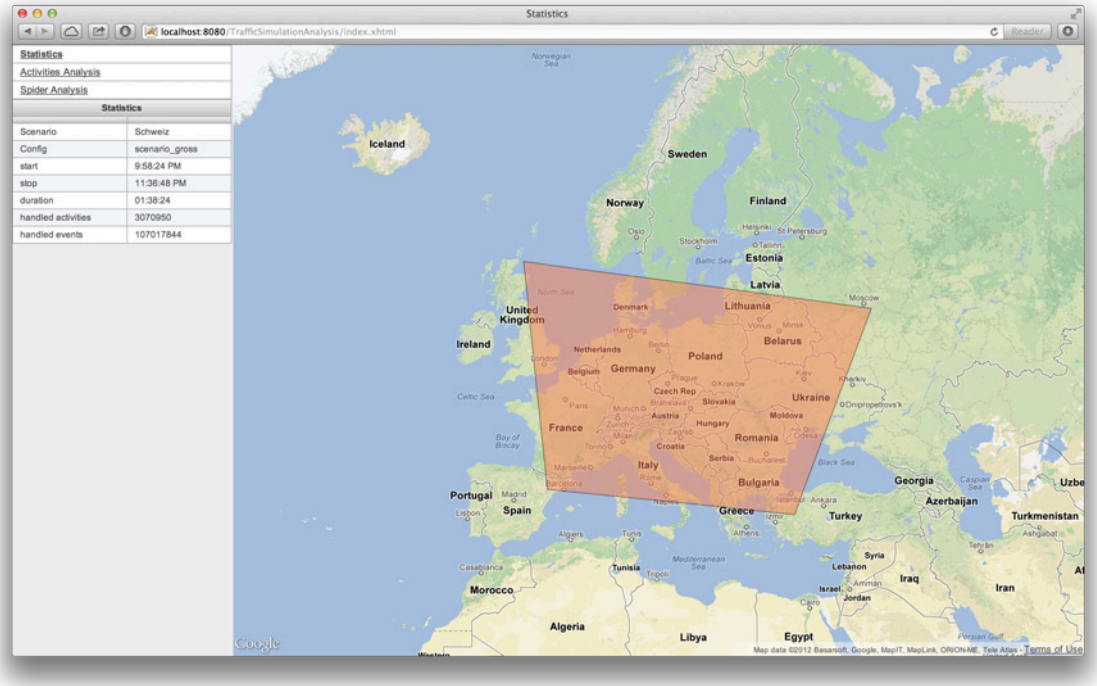


Abbildung 7.9: Einstiegsseite der Applikation

Activities Analysis

Wie im Verkehrsmodell (Kapitel 5.1) beschrieben ist, führen die Agenten der `MATSim`-Simulation während eines Tages verschiedene Aktivitäten aus.

Um eine sinnvolle Darstellung zu ermöglichen, muss vom Benutzer eine geografische Einschränkung vorgenommen werden. Diese geschieht durch Eingabe von Koordinaten oder Klicken auf die Karte sowie der Angabe eines Radius. Mit Angabe von Start- und Endzeit werden die Aktivitäten zeitlich begrenzt. Aufgrund von diesen Informationen werden die Aktivitäten vom Datenspeicher geladen und anschliessend nach Aktivitätstypen und Personen-Kategorien gefiltert.

Die einzelnen Aktivitätstypen werden mit unterschiedlicher Farbe dargestellt und bei einem Klick auf einen Marker werden die Metadaten zu einer spezifischen Aktivität angezeigt. Diese umfassen folgende Informationen:

- activity id
- person id
- duration
- type
- link id
- coordinate

Alle dargestellten Informationen entsprechen den realen Werten, das heisst die synthetischen Werte aus dem Datenspeicher wurden mit dem realen Wert gemappt. (Mehr Informationen zu synthetischen Keys und dem Mapping dazu im Kapitel 6.1.1)

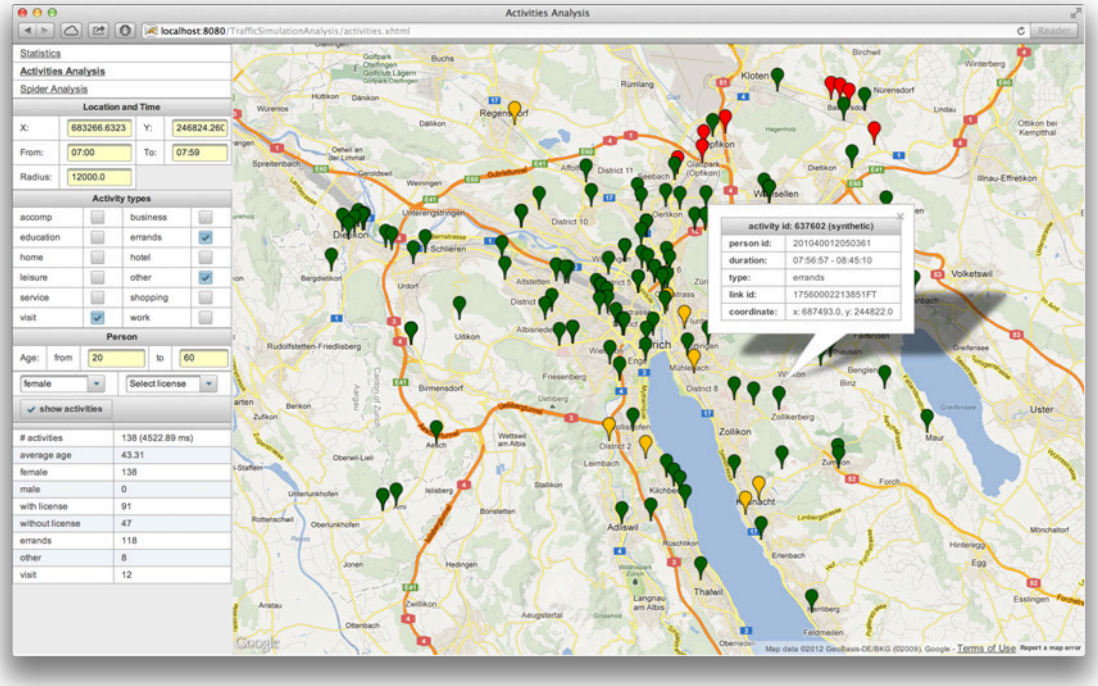


Abbildung 7.10: Aktivitäten der Agenten

Spider Analysis

Das Ziel der `Spider Analysis` ist es, für einen ausgewählten Link den Verkehr über einen bestimmten Zeitraum darzustellen. Ausgelöst wird eine Spinnenanalyse durch einen Klick auf eine Netzwerkkante. Interessant sind vorerst lediglich die Netzwerkkanten, auf welchen Privatverkehr fliesst. Die Verkehrsflüsse vom öffentlichen Verkehr werden in dieser Analyse nicht ausgewertet.

In der Abbildung 7.11 ist zu sehen, wie die Netzwerkdaten, welche aus `MATSim` stammen, über den aktuell dargestellten Bereich der Google Map dargestellt werden. Aus Gründen der Performance wird das Verkehrsnetz nicht über den gesamten analysierten Bereich gelegt, denn das Zeichnen der einzelnen Links ist aufwändig und resultiert in langen Ladezeiten.

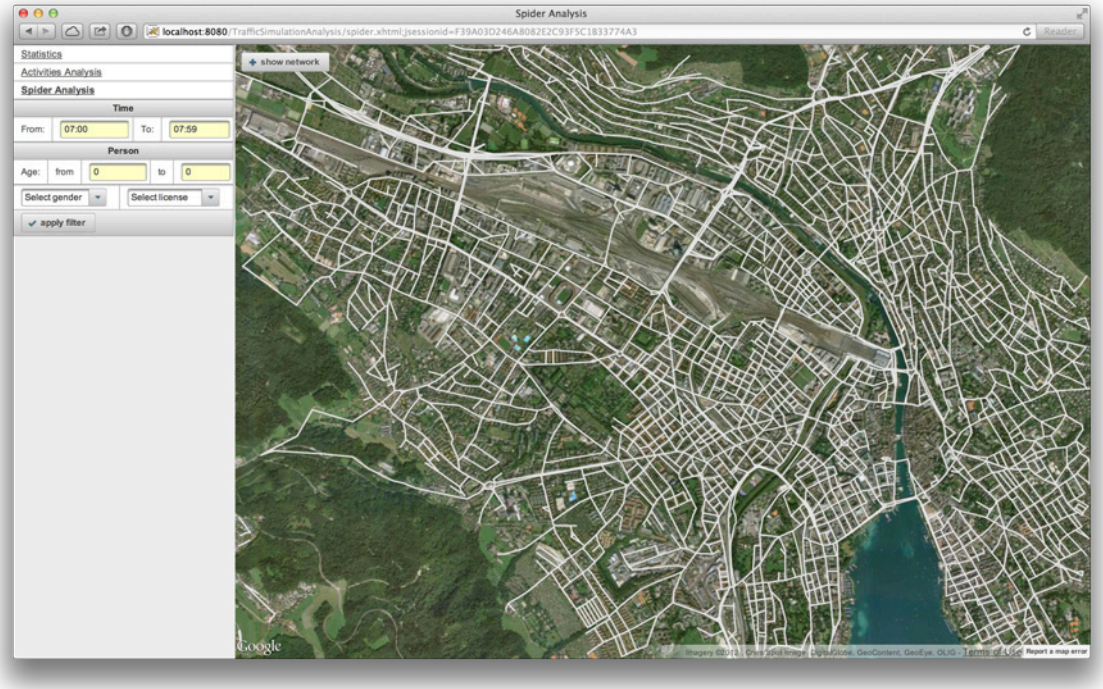


Abbildung 7.11: Darstellung Netzwerk-Overlay

In der Abildung 7.12 ist eine Spinnenalyse für den Bareggtunnel dargestellt. Im linken Bereich werden analog zur Analyse über die Aktivitäten (Kapitel 7.5.1) einige Informationen zu den Resultaten dargestellt. Zusätzlich wird eine Grafik über die Altersverteilung der betroffenen Personen angezeigt. Ebenfalls ist es möglich, die Ergebnisse nach zusätzlichen Dimensionen (vor allem im Personenbereich) zu filtern.

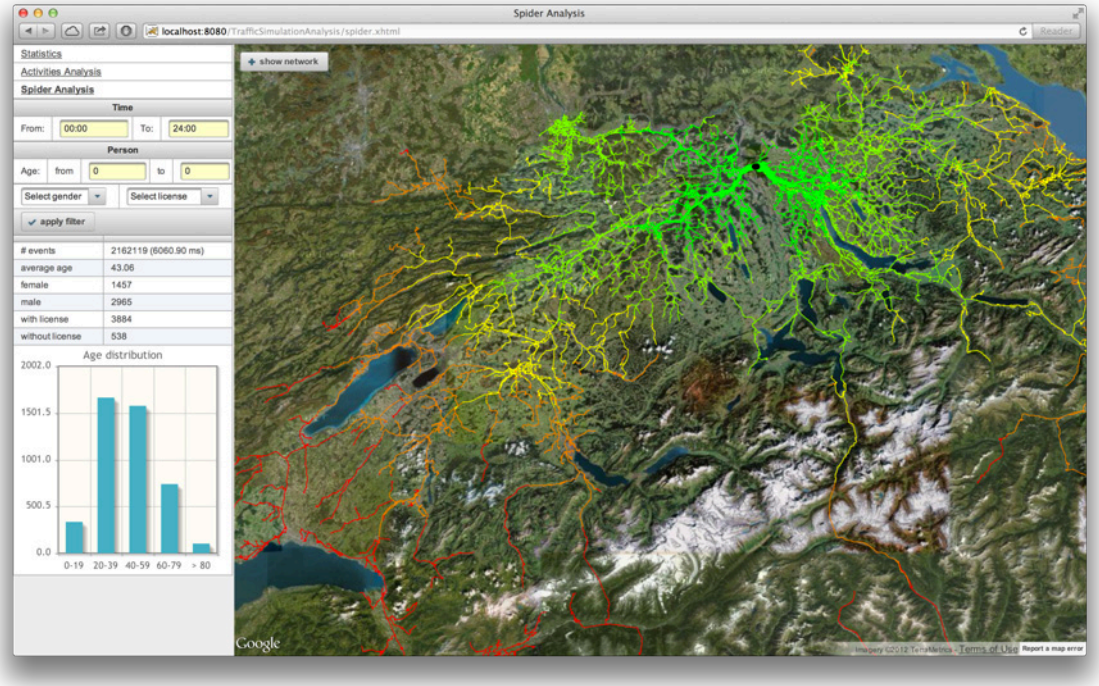


Abbildung 7.12: Spinnenanalyse Bareggtunnel

7.5.2 Technologie

Die Webapplikation wurde mit `JSF 2.1` sowie der Komponentenbibliothek `Primefaces 3.4.2` umgesetzt. Der Vorteil liegt darin, dass die gesamte Entwicklung auf Java basiert und neben `Javascript` und etwas `HTML` keine zusätzlichen Technologien verwendet werden.

Kartenmaterial Als Kartenmaterial wurde Google Maps verwendet. `Primefaces` liefert durch die Komponente `GMap` eine sehr einfache Möglichkeit, Google Maps einzubinden sowie Modifikation (Markers, Polygons, Lines) auf dieser vorzunehmen.

Der für die Analyse vorbereitete Datenspeicher ist unabhängig von der Darstellung einer bestimmten Karte. Auch die Koordinaten werden im Originalformat des Szenarios belassen. In der Webapplikation werden dann die darzustellenden Objekte mit den dazugehörigen Koordinaten in das Google Maps Format (`WGS84`) transformiert. Verwendet wird dazu die von `MATSim` zur Verfügung gestellte Utility-Klasse `CoordinateTransformation`:

```
private static CoordinateTransformation coordinateTransformationACToLatLng =
    TransformationFactory.getCoordinateTransformation(Helpers.
    getScenarioProperty("coordformat"), Constants.TARGETCOORDSYSTEM);
public static LatLng convertAnalysisCoordToLatLng(AnalysisCoord analysisCoord)
    {
  Coord coord = new CoordImpl(analysisCoord.getX(), analysisCoord.getY());
  coord = coordinateTransformationACToLatLng.transform(coord);
  return new LatLng(coord.getY(), coord.getX()); }
```

Bei einer Änderung des Kartenproviders muss überprüft werden, welche Möglichkeiten bestehen, auf der Karte Modifikationen vorzunehmen. Konkret geht es darum, ob es möglich ist, Markers (Aktivitäten) zu setzen sowie einzelne Netzwerkkanten zu zeichnen. Im Showcase ist die Kopplung zwischen den Eingabeformularen und der Karte relativ lose. Anstelle der Google Map können dort auch Tiles von Open Street Map [27] angezeigt werden.

7.5.3 Initializer

Beim Starten der Webapplikation wird das Netzwerk geladen und die Verbindung zur Datenbank (`DBConnection.getPersistLayerReadable();`) geöffnet. Der PersistLayer wird im lesenden Modus geöffnet, so dass bei Bedarf mehrere Prozesse auf die Datenbank zugreifen können. Beim Undeployen der Webapplikation wird der PersistLayer wieder geschlossen (`DBConnection.closeReadable();`).

```
/**
 * Initialize class. Methods are called while starting and stopping the
    webapplication.
 */
@WebListener
public class Initializer implements ServletContextListener {
  private static final Logger LOGGER = Logger.getLogger(Initializer.class);

  @Override
  public void contextInitialized(ServletContextEvent arg0) {
    LOGGER.info("Start preloading: Network and connection to BerkeleyDB");
    DBConnection.getPersistLayerReadable();
    try {
      Network.getNetwork();
    } catch (ParseResultException e) {
      throw new RuntimeException(e);
    }
    LOGGER.info("Preloading done.");
  }

  @Override
  public void contextDestroyed(ServletContextEvent arg0) {
    LOGGER.info("Close connection to BerkeleyDB");
    DBConnection.closeReadable();
  }
}
```

7.6 Konsolenapplikationen

Im Rahmen dieser Arbeit wurden zwei Konsolenapplikationen erstellt, welche dazu dienen, die Abfragemöglichkeiten zu demonstrieren. Sie werden als Ergänzung zur Webapplikation (Kapitel 7.5) gesehen, sind aber komplett unabhängig von dieser.

7.6.1 ConsoleApp

Die Klasse `ConsoleApp` bietet die Funktionalität an, mit einfachen Konsoleneingaben interaktiv einige Beispielszenarios zu benutzen.

```
Please Choose:
0 -> Exit application
1 -> get passengers for link and time
2 -> get events for link and time
>> 2
Get events
Enter Link Id(s) [id1, id2, ...]
12, 13
Enter Time from [hh:mm]
08:00
Enter Time to [hh:mm]
13:00
# of results 7
Print results? [y/n]
y
AnalysisEvent [id=3194277, personId=16384, linkId=17932, startTime=45178.0,
    endTime=45182.0, analysisTrip=146465]
AnalysisEvent [id=1562144, personId=30273, linkId=16843, startTime=31422.0,
    endTime=31426.0, analysisTrip=79349]
AnalysisEvent [id=2361590, personId=30273, linkId=17932, startTime=37755.0,
    endTime=37759.0, analysisTrip=116667]
AnalysisEvent [id=3005819, personId=18690, linkId=17932, startTime=43670.0,
    endTime=43674.0, analysisTrip=143234]
AnalysisEvent [id=2979603, personId=30717, linkId=17932, startTime=43449.0,
    endTime=43453.0, analysisTrip=142007]
AnalysisEvent [id=3005816, personId=18690, linkId=16843, startTime=43669.0,
    endTime=43670.0, analysisTrip=143234]
AnalysisEvent [id=3209690, personId=52454, linkId=17932, startTime=45294.0,
    endTime=45298.0, analysisTrip=144821]
```

7.6.2 ExampleAnalyzeApp

In der Klasse `ExampleAnalyzeApp` sind verschiedene Beispiele für die Analyse eines Szenarios implementiert:

- getAllActivitiesForCoordAndRadiusAndTimeSlices
- getAllActivitiesForCoordAndRadiusAndTimeSlicesViaSectors
- getAllEventsForLinkAndTimeSlice
- getAllEventsForLinks
- getAllEventsForTimeSlices
- getAllPassengersForLinkAndTimeSlice

Die implementierten Beispielszenarios sind nur ein Teil der Funktionalität, die von der Klasse `Analyzer` zur Verfügung gestellt wird.

7.7 Konfiguration

Die Applikation `Traffic Simulation Analysis` kann mit mehreren Konfigurationsfiles an die Bedürfnisse des Benutzers angepasst werden.

Das File `config.properties` konfiguriert die Datei-Bezeichnungen der Dateien, welche im Parse-Vorgang produziert und später im Zug der Analyse verwendet werden.

```
parsingresult=parsingResult.ser
network=network.ser
linkidmapping=linkIdMapping.txt
personidmapping=personIdMapping.txt
activitytypemapping=activityTypeMapping.txt
vehicleidmapping=vehicleIdMapping.txt
transitlineidmapping=transitLineIdMapping.txt
transitrouteidmapping=transitRouteIdMapping.txt

scenario=scenario_gross
```

Das File `scenario.properties` wird pro Szenario erstellt. Es können die spezifischen Input-Dateien angegeben werden, das Koordinatenformat sowie weitere Parameter für die Verwendung in der Webapplikation.

```
name=Schweiz

# input
input=TrafficSimulationAnalysis/input/gross/
events=events.xml
network=network.xml
population=plans.xml
coordformat=CH1903_LV03

# parse
# in seconds
timesliceinterval=900
# in bytes
memorylimit=500000000
sectorwidthdelimiter=100

# output
output=TrafficSimulationAnalysis/output_gross/
initialviewport=47.367, 8.541
initialzoomlevel=12
x=683266.6323
y=246824.2605
radius=500.0
from=07:00
to=07:59
```

7.8 Verwendete Technologien

Folgende Libraries wurden bei der Umsetzung des Projektes verwendet:

Library	**Version**	**Anwendungsbereich**
Berkeley DB	5.0.58	Persistenzlayer
JSF	2.1	Web Application
Primefaces	3.4.2	GUI Komponentenbibliothek
MATSim	4.10	Traffic Simulation Software
lambdaj	2.3.3	Collection Filter
guava	13.0.1	MultiHashMap
jUnit	4.10	Unit Testing
Apache Maven	3.0.3	Build Process Support
Alertify	11. Dez 12	JavaScript Notification System
Log4j	1.2.14	Logging Framework

Tabelle 7.1: Übersicht verwendeter Libraries

Für den Betrieb wird ein Java-Webserver benötigt. Während des Projektes wurde mit `Apache Tomcat` in der Version 7.0.30 gearbeitet. Der Ablauf des Deployments ist im Kapitel 7.3 detailliert beschrieben.

Das Logging innerhalb der Applikation wurde mit `Log4j` [28] umgesetzt. Somit wird dieselbe Technologie wie beim `MATSim`-Projekt [1] eingesetzt, was zukünftige Anpassungen erleichtert.

7.9 Testing

7.9.1 Beschreibung

Mit Hilfe von Unittests wird die Korrektheit des Parsen sichergestellt. Die verschiedenen Tests stellen sicher, dass die Daten korrekt geparst und gespeichert werden. Zusätzlich existieren Tests, welche das korrekte Filtern von Objekten überprüfen.

Um die Richtigkeit der Daten in der neuen Struktur zu prüfen, wurden Vergleiche mit den verarbeiteten XML-Dateien gemacht. Je nach Dateigrösse wurden die Quelldaten im Texteditor oder in der Konsole mit Hilfe von "`grep`" und "`more`" betrachtet.

7.9.2 Testabdeckung

Das eingesetzte Test Coverage Tool ist Cobertura [29]. Nachfolgende Abbildung zeigt die Testabdeckung über das gesamte Projekt. Ausgenommen von der Analyse sind alle Util-, sowie Exception-Klassen.

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	45	78%	1114/1423	66%	252/378	1.7
analyze	8	74%	159/213	65%	60/92	1.845
parse	7	92%	411/442	90%	90/100	1.934
persist	2	84%	84/99	56%	17/30	1.536
persist.bucket	4	58%	38/65	60%	6/10	1.385
persist.dbaccess	7	94%	131/138	85%	24/28	1.581
util	7	61%	119/192	83%	40/48	1.58
vo	10	62%	172/274	21%	15/70	1.663

Report generated by Cobertura 1.9.4.1 on 12/12/12 8:31 PM.

Abbildung 7.13: Testabdeckung Cobertura

Der Fokus wurde vor allem auf die Packages `parse` und `dbaccess` gelegt, da dies zwei Stellen mit erhöhter Komplexität sind.

7.10 Metriken

7.10.1 STAN

STAN (Structure Analysis for Java) [21] stellt neben verschiedenen Metriken wie beispielsweise Anzahl Zeilen Code oder der zyklomatische Komplexität die Abhängigkeiten zwischen den einzelnen Packages dar. Diese Information ist sehr hilfreich, da so relativ offensichtliche Designfehler in der Architektur sichtbar werden. Nachfolgend sind einige wichtige Metriken aufgeführt, welche Aussagen über das Projekt machen:

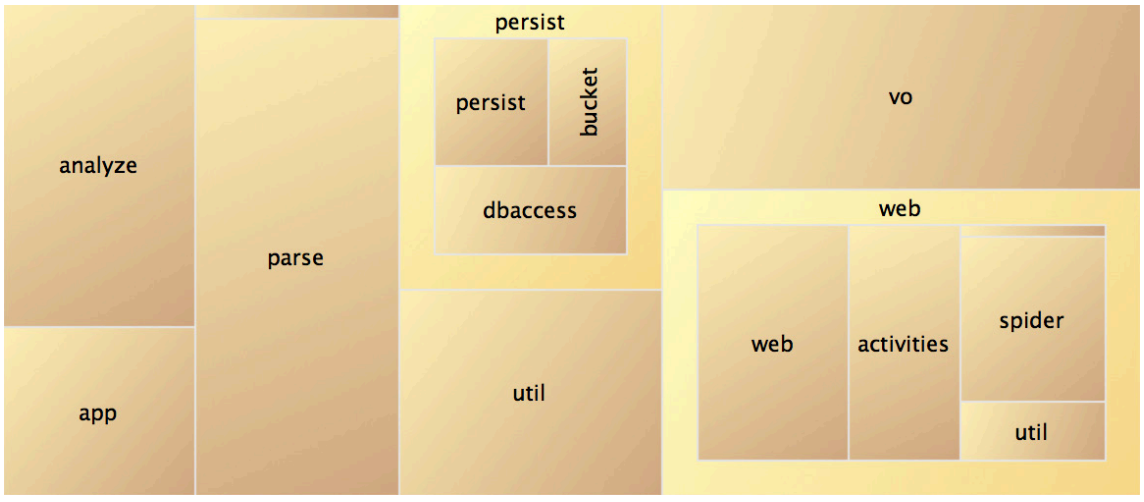


Abbildung 7.14: Verhältnis der Package-Grössen

Metrik	Wert
Packages	14
Units	67
Units/Package	4.79
Methods/Class	8.17
Fields/Class	3.8
ELOC (Estimated Lines Of Code)	4366
ELOC/Unit	65.16
Cyclomatic Complexity	1.5

Tabelle 7.2: Ausgewählte Code-Metriken

7.10.2 Checkstyle

Checkstyle [30] wurde eingesetzt, um einen einheitlichen Programmierstil unter den Entwicklern zu gewährleisten. Die Standard-Konfiguration wurde leicht angepasst und im SVN-Repository für alle Entwickler zur Verfügung gestellt.

7.10.3 UCDetector

`UCDetector` [31] wurde dazu verwendet, unnötigen Code aufzudecken, so dass dieser entsprechend restrukturiert werden konnte. In einigen Klassen wurden Methoden gefunden, welche angedacht wurden, jedoch im finalen Code keine Verwendung fanden. Viele Warnungen stellten sich aber auch als falsch heraus, besonders dann, wenn die Methoden durch die Webapplikation aufgerufen werden. `UCDetector` durchsucht die XHTML-Seiten nicht nach entsprechenden Aufrufen.

7.10.4 FindBugs

FindBugs [32] wurde für die statische Code-Analyse eingesetzt. Typische Probleme, welche gelöst werden konnten, waren beispielsweise:

- nicht serialisierbare Instanzvariablen in serialisierbaren Klassen
- Vergleich von Floating Point Zahlen
- `null`-Überprüfungen nach bereits erfolgtem Zugriff auf Variable

Kapitel 8

Experimentelle Auswertung

8.1 Performance

In mehreren Etappen werden die kompletten Analysestrukturen in die `Berkeley DB` gespeichert. Dies muss mehrmals geschehen, da der Arbeitsspeicher nicht ausreichend ist, um diese erst am Ende des Parse-Vorgangs zu speichern. Sobald ein Zwischenstand in die Datenbank gespeichert wird, muss von `Berkeley DB` Arbeitsspeicher genutzt werden, um die Werte zu speichern. Falls jedoch schon zu viele Objektinstanzen aufgebaut wurden, hat Berkeley zu wenig Arbeitsspeicher, um diese Objekte zu speichern. Dies kann zu `OutOfMemoryExceptions` oder `Swapping` führen. Dies ist ein unerwünschtes Verhalten, weshalb eine Strategie entwickelt wurde, welche zur Laufzeit prüft, wieviel Arbeitsspeicher zur Verfügung steht (allozierter minus gebundener Arbeitsspeicher). Abhängig davon wird dann die Speicherung ausgelöst. Dieser Wert kann mittels Konfigurationsfiles angepasst werden.

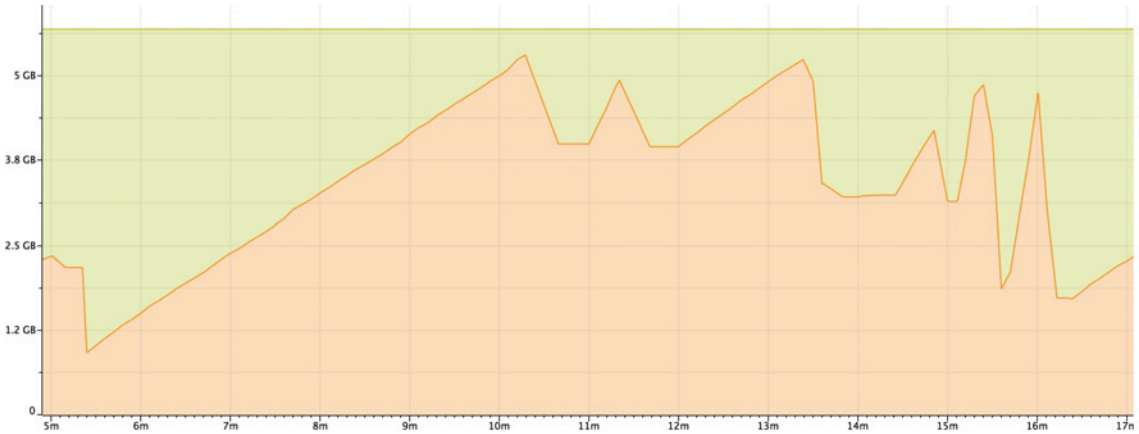


Abbildung 8.1: Ablauf der ersten Speicherphase

In der Abbildung 8.1 ist zu sehen, dass ungefähr nach zehn Minuten die Speicherbelastung so hoch ist, dass das Speichern auf die Datenbank ausgelöst wird. Um die einzelnen Abbildungen (6.1.1) zu speichern, wird von `Berkeley DB` jeweils zusätzlich Ar-

beitsspeicher benötigt (vertikale Schwankungen). In den Einstellungen für den Zugriff auf die `Berkeley DB` besteht die Möglichkeit, den maximalen Arbeitsspeicher zu definieren. In verschiedenen Tests wurde dafür der Wert von `100 MB` pro Datastore als optimal befunden. Mit weniger oder mehr Arbeitsspeicher dauert der Parse-Vorgang jeweils länger.

```
envConfig.setConfigParam(EnvironmentConfig.MAX_MEMORY, "100000000");
```

Zusätzlich hat diese Einstellung zur Folge, dass keine Memory Leaks seitens der `Berkeley DB` entstehen. In der Abbildung 8.2 ist dargestellt, wie der RAM-Verbraucht ohne Limit sowie mit Limit mit 100 MB variiert.

Abbildung 8.2: Vergleich Speicherverbrauch ohne und mit RAM Limit

8.2 Skalierung

8.2.1 Parsen

Nachfolgend wird aufgezeigt, wie die Lösung für den Parse-Vorgang skaliert. Interessant ist die Betrachtung von Szenarien unterschiedlicher Grösse. Die verschiedenen Szenarien wurden von `Senozon` zur Verfügung gestellt. Die folgenden Tests wurden auf einem System mit SSD, Intel Core i7 3615QM sowie 6 GB Arbeitsspeicher ausgeführt. Jedes Szenario wurde einmal geparst.

Szenario	Anzahl Events	Grösse Events-File	Parse-Vorgang (hh:mm:ss)
klein	34'239	3 MB	00:00:03
mittel	32'858'495	3.3 GB	00:04:53
gross	250'080'746	30 GB	01:22:25

Tabelle 8.1: Skalierung des Parse-Vorgangs

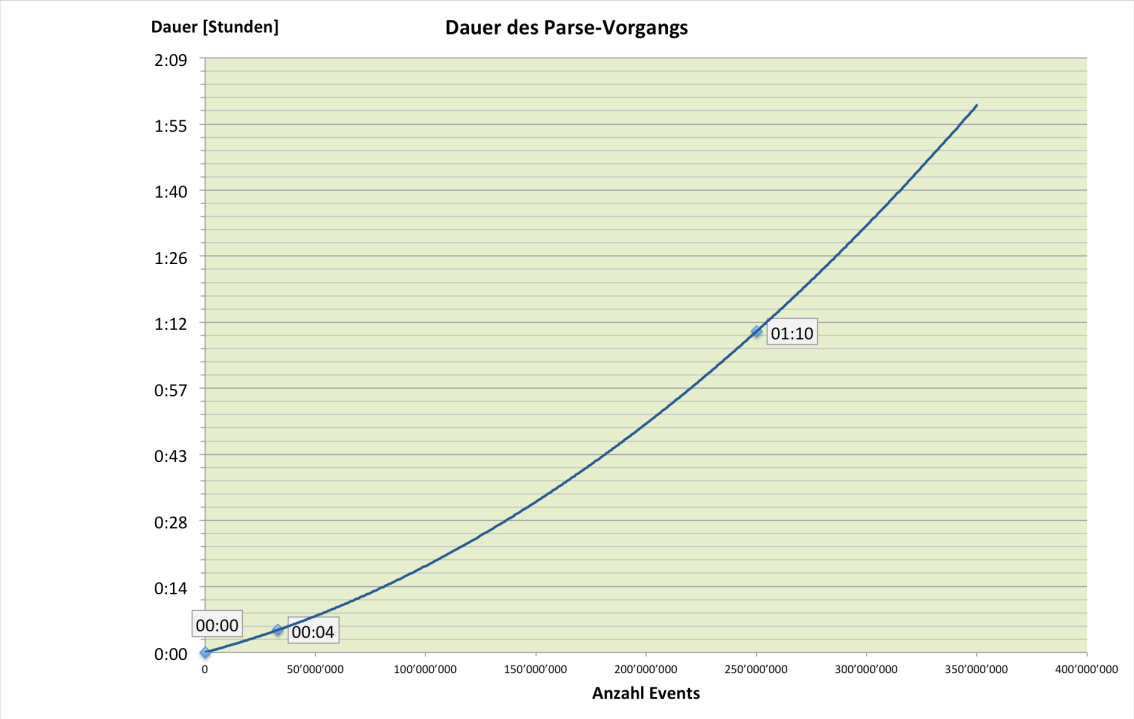


Abbildung 8.3: Skalierung der verschiedenen Szenarien

In der Abbildung 8.3 sind die Zeiten für das Parsen des Event-Files für verschiedene Szenario-Grössen dargestellt. Die Kurve verläuft polynomial, da sich die Zeit für die Zwischenspeicherung der Daten nicht linear verhält, da bei einem Speichervorgang die gesamten Daten aus dem Key/Value Store geladen werden müssen.

Die implementierte Lösung skaliert auch von den Datentypen mit viel grösseren Szenarien. Alle in `Berkeley DB` benutzten Keys sind `Longs`, was für aktuelle Szenarien nicht notwendig wäre (250 Millionen Events), in Zukunft aber durchaus relevant werden könnte.

In einem Test (System mit Raid 0, 7200 RPM, Intel Core Quad Q9400 und 5.5 GB RAM) dauerte der komplette Parse-Vorgang 2 Stunden und 50 Minuten. Damit ist gezeigt, dass das Parsen auch auf etwas älteren Systemen gut skaliert.

8.2.2 Analyse

Kritisch für dieses Projekt sind die Zeiten, welche bei der Abfrage der Daten erreicht werden. Diese sind tabellarisch für ein System mit SSD, Intel Core i7 3615QM sowie 6 GB RAM aufgelistet:

Nr	Abfrage	Grösse Events-File	Analyse
1	Aktivitäten (Radius 500, 7 - 8 Uhr)	3.3 GB	639.97 ms
2	wiederholte Abfrage	3.3 GB	68.08 ms
3	wiederholte Abfrage	3.3 GB	43.31 ms
4	Spinnenanalyse Link 63998 (7 - 8 Uhr)	3.3 GB	172.27 ms
5	wiederholte Abfrage	3.3 GB	21.32 ms
6	wiederholte Abfrage	3.3 GB	29.05 ms
7	Aktivitäten (Radius 500, 7 - 8 Uhr)	30 GB	5346.22 ms
8	wiederholte Abfrage	30 GB	873.27 ms
9	wiederholte Abfrage	30 GB	826.87 ms
10	Spinnenanalyse Link 1038399 (7 - 8 Uhr)	30 GB	530.14 ms
11	wiederholte Abfrage	30 GB	171.54 ms
12	wiederholte Abfrage	30 GB	169.54 ms

Tabelle 8.2: Skalierung der Analyse auf System A

Zu erkennen ist, dass bei der Abfrage #7 der Zugriff auf die `Berkeley DB` relativ lang dauert. Grund ist, dass zu diesem Zeitpunkt noch keine Abfragen abgesetzt wurden und gewisse Initialisierungsarbeiten mehr Zeit brauchen. In späteren Abfragen ist der Zugriff erheblich schneller.

Auch bei der Abfrage der Daten haben wir mehrere Systeme getestet. Die folgenden Ergebnisse entstanden auf einem System mit Raid 0, 7200 RPM, Intel Core Quad Q9400 und 5.5 GB RAM:

Nr	Abfrage	Grösse Events-File	Analyse
1	Aktivitäten (Radius 500, 7 - 8 Uhr)	3.3 GB	741.81 ms
2	wiederholte Abfrage	3.3 GB	73.84 ms
3	wiederholte Abfrage	3.3 GB	58.89 ms
4	Spinnenanalyse Link 63998 (7 - 8 Uhr)	3.3 GB	1780.20 ms
5	wiederholte Abfrage	3.3 GB	30.92 ms
6	wiederholte Abfrage	3.3 GB	32.54 ms
7	Aktivitäten (Radius 500, 7 - 8 Uhr)	30 GB	9712.66 ms
8	wiederholte Abfrage	30 GB	1161.70 ms
9	wiederholte Abfrage	30 GB	1142.54 ms
10	Spinnenanalyse Link 1038399 (7 - 8 Uhr)	30 GB	6649.79 ms
11	wiederholte Abfrage	30 GB	149.10 ms
12	wiederholte Abfrage	30 GB	155.20 ms

Tabelle 8.3: Skalierung der Analyse auf System B

Kapitel 9

Schlussfolgerungen

9.1 Nutzen

Die Applikation `Traffic Simulation Analysis` ermöglicht, dass Simulationsdaten nur einmal zeitaufwändig geparst werden müssen. Anschliessend können auf diesen Ergebnissen mehrfach interaktive Analysen durchgeführt werden. Diese Analysen umfassen neben textuellen Resultaten vor allem auch visuelle Darstellungen. Diese können parametrisiert und gefiltert werden. Es muss somit beim Parse-Vorgang der Simulationsdaten noch nicht eindeutig definiert sein, welche Aspekte von Interesse sind.

9.2 Erweiterungsmöglichkeiten

Nachfolgend sind ausgewählte Bereiche beschrieben, in welchen Optimierungen beziehungsweise Erweiterungen möglich sind.

9.2.1 Performance

Im Bereich der Performance des Parse-Vorgangs gibt es wenig Optimierungsmöglichkeiten. In einem Test (System mit SSD, Intel Core i7 3615QM und 6 GB) wurde nur das Events-File geparst, aber keine Daten auf die Festplatte geschrieben. Dieser Parse-Vorgang dauerte 19 Minuten. Der wichtigste Faktor in der benötigten Zeit für die Speicherung der Daten ist der Festplattenzugriff. Falls die Input-Daten auf einer anderen physikalischen Festplatte liegen würden, könnte dieser Prozess parallelisiert werden. Mit dieser Lösung könnte parallel zum Einlesen der Daten auch bereits mit dem Schreiben begonnen werden.

9.2.2 Datenspeicher

Im Bereich des Datenspeichers gibt es folgende Optimierungsmöglichkeiten:

- Adaptive Sektor-Einteilung, so dass in dichter besiedelten Gebieten mit vielen Aktivitäten die Sektorgrösse kleiner gewählt wird.
- detailliertere Speicherung der Daten, welche den öffentlichen Verkehr betreffen (Streckenabschnitt)

9.2.3 Webapplikation

Die Webapplikation kann durch mehrere Szenarien ergänzt werden. Technologisch gibt es folgende Optimierungsmöglichkeiten:

- Mehr Requests per Ajax absetzen, so dass die Kartenanwendung interaktiver wird.
- Fehlermeldungen vom Server in der Webapplikation anzeigen.

Kapitel 10

Projektmanagement

10.1 Projektplan

10.1.1 Projektorganisation

Betreuung

Die Studienarbeit wird von Herrn Prof. Dr. Luc Bläser betreut. Die Bewertung erfolgt ebenfalls durch ihn.

Auftraggeber

Auftraggeber ist die Firma `Senozon AG` mit Sitz in Zürich. `Senozon` wird durch die folgenden Personen mit den jeweiligen Spezialgebieten vertreten:

- Dr. Michael Balmer, CEO (Kundensicht, Abfragen)
- Dr. Marcel Rieser, CTO (File-Formate, API, techn. Aspekte)

Entwicklung

Umgesetzt wird das Projekt durch die beiden Studenten

- Mirco Widmer, m1widmer@hsr.ch
- Matthias Zürcher, mzuerche@hsr.ch

mit den folgenden Kompetenzbereichen:

Mirco Widmer	**Matthias Zürcher**
Abfragen der Daten	Einlesen der Daten
GUI-Komponente	Tuning, Optimierungen

Tabelle 10.1: Kompetenzbereiche

10.1.2 Management-Abläufe

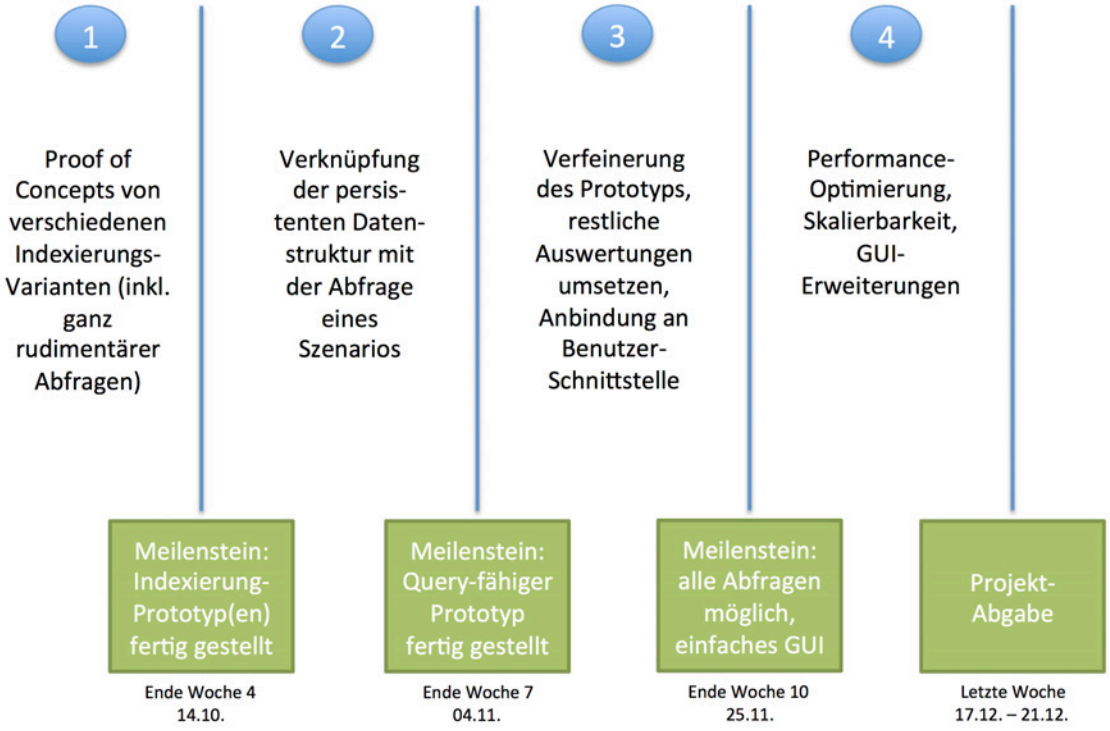


Abbildung 10.1: Aufteilung in 4 Sprints à je 3 Wochen

Zeitplanung Von den zur Verfügung stehenden 14 Wochen, sind 12 in der Zeitplanung berücksichtigt. Die erste sowie auch die letzte Woche werden speziell behandelt und sind für Initialisierungs- beziehungsweise Abschlussarbeiten vorgesehen. Die 12 Wochen Arbeitszeit werden in 4 Sprints à je 3 Wochen aufgeteilt. Zu jedem dieser Sprints gibt es einen Meilenstein, welcher durch eine lauffähige Version der Software überprüft werden kann.

Termine

17.09.12 Beginn der Studienarbeit, Ausgabe der Aufgabenstellung durch die Betreuer.

17.12.12 Die Studierenden senden folgende Dokumente der Arbeit per Email zur Prüfung an ihre Betreuer:

- Kurzfassung
- A0-Poster

Vorlagen sowie eine ausführliche Anleitung betreffend Dokumentation stehen unter den allgemeinen Infos Diplom-, Bachelor- und Studienarbeiten zur Verfügung.

21.12.12 Die Studierenden senden die vom Betreuer abgenommene und freigegebene Kurzfassung als Word-Dokument an das Studiengangsekretariat (cfurrer(at)hsr.ch).

21.12.12 Abgabe des Berichtes an den Betreuer bis 17.00 Uhr.

Besprechungen

- Skype-Konferenz alle 2 Wochen von 13 - 14 Uhr, beginnend ab 24.09.12, alternativ auch vor Ort
- Besprechung zwischen L. Bläser, M. Zürcher und M. Widmer über Arbeitsfortschritt, alle 2 Wochen von 13 - 14 Uhr, beginnend ab 1.10.12, bei Bedarf auch häufiger.

10.1.3 Risikomanagement

Beschreibung	**Vorbeugung**	**Verhalten beim Eintreten**
Die gewünschten Auswertungen dauern viel länger als geplant	- Detaillierte Überlegungen zur Datenstruktur um eine hohe Performance zu gewährleisten.	- Optimierungen einbauen - Besprechung mit Auftraggebern
Die Entwicklungsumgebung oder Teile davon sind nicht verfügbar. (SVN, Buildserver, Notebooks, Server)	- Wöchentlich Backups erstellen - Alternative Hardware bereithalten (Schul-PCs)	- Backups zurückspielen - Wechsel der Infrastruktur
Es wird mehr Zeit als geplant benötigt um die neuen Technologien kennenzulernen (Matsim, Datenablagestrukturen, Frontend Frameworks)	- Einarbeitung in die Themen in den ersten Wochen - Technische Einführung	- Zusätzliche Bücher beschaffen - Drittpersonen zu Themen anfragen
Die Entwicklung der Applikation benötigt mehr Zeit als zur Verfügung steht.	- Regelmässige Überprüfung des Fortschritts - Regelmässige Statusmeetings	- Erweitertes GUI nicht umsetzen
Teammitglied kann aus verschiedenen Gründen (Krankheit, Unfall, etc.) länger als 1 Woche nicht am Projekt arbeiten.	- Täglich Code einchecken - Häufiger Informationsaustausch zwischen Teammitgliedern, damit Aufgaben vom anderen übernommen werden können.	- Besprechung mit Auftraggebern und Neupriorisierung

Tabelle 10.2: Risikoübersicht

10.1.4 Infrastruktur

M. Widmer und M. Zürcher arbeiten beide mit ihrem privaten Laptop. (Mac OS X 10.8, Windows 7)

10.1.5 Projekttools

- Zeiterfassung (Tickets) mit Redmine
- Dokumentation mit LaTeX

Entwicklungsumgebung

- Eclipse IDE for Java Developers
- Als Code-Repository wird das bestehende SVN-Repository der Firma `Senozon AG` verwendet. Somit haben alle beteiligten Personen Zugriff auf den Entwicklungs-Code.
- Verwendung von via [33]
- XML-Editoren, Texteditoren nach Bedarf
- Jenkins-Buildserver nach Bedarf

10.2 Projektverlauf

Als Resultat des Sprint 1 konnte ein erster Prototyp mit einer In-Memory Lösung geliefert werden. Dieser hatte bereits die Struktur der Abbildungen, welche bis zum finalen Stand nur noch kleine Änderungen erfuhr.

Da die In-Memory Lösung einen zu hohen Speicherbedarf hatte, musste eine Alternative gefunden werden. Analysen und Tests von unterschiedlichen Datenbankprodukten brachten dann in `Berkeley DB` eine brauchbare Lösung, die Daten auf der Festplatte speichert und trotzdem schnelle Zugriffe ermöglicht. Dabei trat das Risiko 3 (10.1.3) ein und wir benötigten für diese Analysen mehr Zeit als geplant.

Dadurch, dass wir in einer frühen Phase auf diese Problem gestossen sind, konnten wir die zusätzlich gebrauchte Zeit im weiteren Verlauf der Arbeit wieder aufholen.

Andere Risiken wie im Risikomanagement (10.1.3) beschrieben sind während der Arbeit nicht eingetroffen.

Im dritten Sprint entstand auf Basis des ersten Prototypen eine neue Lösung. Diese beinhaltete das Persistieren der Daten auf der Festplatte sowie ein bereits sehr komplexes GUI mit einer schnellen Darstellung der Simulationsdaten.

Der letzte Sprint nutzen wir für Optimierungen und Verbesserungen am Code. Das sehr ausführliche Code-Review von Herrn Bläser wurde umgesetzt und zusätzlich wurde der technische Bericht finalisiert.

10.3 Persönliche Berichte

10.3.1 Mirco Widmer

Für die Studienarbeit war mir wichtig, dass ich an einem Projekt arbeiten konnte, welches mich interessiert und herausfordert. Durch die frühe Festlegung des Themas fanden wir mit der Firma `Senozon AG` einen ausgezeichneten Auftraggeber. Bei der Projektplanung teilten wir die Kompetenzen grob auf: Ich war verantwortlich für die Darstellung der Showcases sowie den Zugriff auf die interne Datenstruktur.

Zur Festlegung der einzusetzenden Technologie untersuchte ich in einer Technologiestudie vor allem Key/Value Stores und stellte fest, dass diese einerseits sehr verständlich aufgebaut sind, andererseits aber oft keine Persistenzschicht zur Verfügung stellen. In unserem ersten Prototyp verzichteten wir denn auch auf die Persistenzschicht und konzentrierten uns auf den Aufbau der internen Datenstruktur. Dabei stellte ich fest, dass Details bei einem Architekturprototyp zu vernachlässigen sind und man sich darauf konzentrieren sollte, alle Architekturkomponenten zu testen.

Eine der herausforderndsten Phasen des Projektes war aus meiner Sicht das Bilden der internen Datenstruktur. Ich musste erst herausfinden, wie ich aus den `MATSim`-Daten eine optimierte interne Datenstruktur für das Szenario "Berechnung von Strassenbelastungen" aufbauen konnte. In dieser Zeit war eine enge Zusammenarbeit zwischen Matthias und mir einer der Schlüsselfaktoren für ein erfolgreiche Lösung.

Als das Projekt weiter fortgeschritten war, konnte ich mit dem Aufbau der GUI-Komponenten beginnen. Die einzelnen Visualisierungen, besonders der Kartendarstellung, forderten mich heraus, da ich mit `JSF` bis jetzt noch nicht solche komplexe Webapplikationen konzipiert und umgesetzt hatte. Ich lernte, dass ein einfaches User Interface von einem Anwender ganz anders verwendet werden kann als von mir erwartet. Zudem habe ich sehr viel im Bereich des Architektur-Designs einer Applikation gelernt.

Zusammenarbeit

Die Zusammenarbeit mit Matthias war sehr angenehm. Da wir bereits in früheren Software Engineering Projekten an der HSR zusammen arbeiteten, wusste ich ziemlich genau wie wir uns ergänzten. Unser Auftraggeber war immer sehr hilfsbereit und meist hatte ich innerhalb von kurzer Zeit eine Antwort oder einen Hinweis auf ein aktuelles Problem. Besonders schätzte ich auch die Statusmeetings, in welchen ich oft den aktuellen Stand der Showcases zeigen konnte und Feedback erhielt. Die Betreuung durch Prof. Dr. Bläser war sehr angenehm und konstruktiv.

Lessons Learned

- Es ist wichtig, in einem Architekturprototyp alle kritischen Bereiche einer Software abzudecken. Während dieses Schrittes kann man unwichtige Details ausblenden.
- Trotz den vielen Kommunikationsmöglichkeiten ist ein Face-to-Face-Austausch mit anderen Entwicklern die effizienteste Form, um anspruchsvolle Probleme zu lösen.

- Das Erreichen der physikalischen Grenze eines Systems bezüglich Speicherauslastung war äusserst lehrreich. Ich habe daraus gelernt, wie wichtig ressourcenschonende Entwicklung ist.

10.3.2 Matthias Zürcher

Von Anfang an war ich am Thema dieser Arbeit interessiert. Dies liegt einerseits am Bereich der Verkehrssimulation, welchen ich sehr spannend finde - andererseits aber auch an der Tatsache, dass nicht sicher war, ob das Problem überhaupt mit den gegebenen technischen Vorgaben zu lösen ist.

Zu Beginn mussten wir neben dem Einarbeiten in die Problemdomäne verschiedene Datenspeicher untersuchen. Dies fand ich etwas zermürbend. Immer wenn wir den Eindruck hatten, ein passendes Produkt gefunden zu haben, fehlten zwingend benötigte Eigenschaften. Als wir mit `Berkeley DB` ein Produkt gefunden hatten, konnten wir aber endlich gut vorwärts arbeiten und die geforderten Analysen implementieren.

Durch die Einteilung in dreiwöchige Sprints waren wir in der Lage, unseren Auftraggebern jeweils eine lauffähige Version der Software zu demonstrieren. Dies empfand ich als sehr hilfreich, um unsere bisherigen Ergebnisse aufzuzeigen und Feedback zu erhalten.

Neben der Bewältigung der grossen Datenmengen, fand ich auch die Berechnung der Koordinaten eine Herausforderung. Die Implementierung der Analyse via Sektoren, welche die Anzahl der zu durchsuchenden Aktivitäten beschränkt, brauchte mehrere Versuche, bis sie zuletzt schnellere Resultat lieferte.

Etwa drei Wochen vor Abgabe schlossen wir die funktionale Implementierung ab. Dies gab uns genügend Zeit die Dokumentation zu finalisieren sowie Probleme, welche beim Code-Review aufgetaucht sind, zu beheben.

Zusammenarbeit

Da Mirco und ich bereits verschiedene Projekte zusammen durchgeführt hatten, gab es bei mir keine Bedenken bezüglich der Zusammenarbeit. Es hat sich dann auch in diesem Projekt einmal mehr gezeigt, dass wir uns gut ergänzen und ohne grössere Probleme zusammenarbeiten können.

Die Zusammenarbeit mit unseren Auftraggebern, Marcel Rieser und Michael Balmer von der `Sonozon AG`, fand ich von Beginn an angenehm und entspannt. Sie haben uns im Verlauf der Arbeit sehr gut unterstützt und wertvolle Hilfe geboten.

Aus meiner Sicht sehr gut betreut hat uns auch Prof. Dr. Bläser. In den wöchentlichen Meetings hat er sich Zeit für unsere Arbeit genommen und uns konstruktives Feedback gegeben. Sehr hilfreich war auch das detaillierte Code-Review, welches wir gegen Ende des Projektes erhalten hatten.

Lessons Learned

- Es lohnt sich genügend Zeit in die Technologiestudie zu investieren, damit das bestmögliche Produkt gefunden werden kann.
- Gute Testfälle ermöglichen das schnelle Überprüfen der Funktionalität der Applikation nach einer Änderung.

10.4 Zeiterfassung

10.4.1 Auswertung

Aufwand pro Sprint

In der Abbildung 10.2 sind die Schätzungen sowie die realen Aufwände aggregiert pro Sprint aufgelistet. Speziell ist, dass die Sprints 0 und 5 jeweils nur eine Woche lang dauerten und für Start- und Abschlussarbeiten genutzt wurden. Die Sprints 1 - 4 sind jeweils 3 Wochen lang.

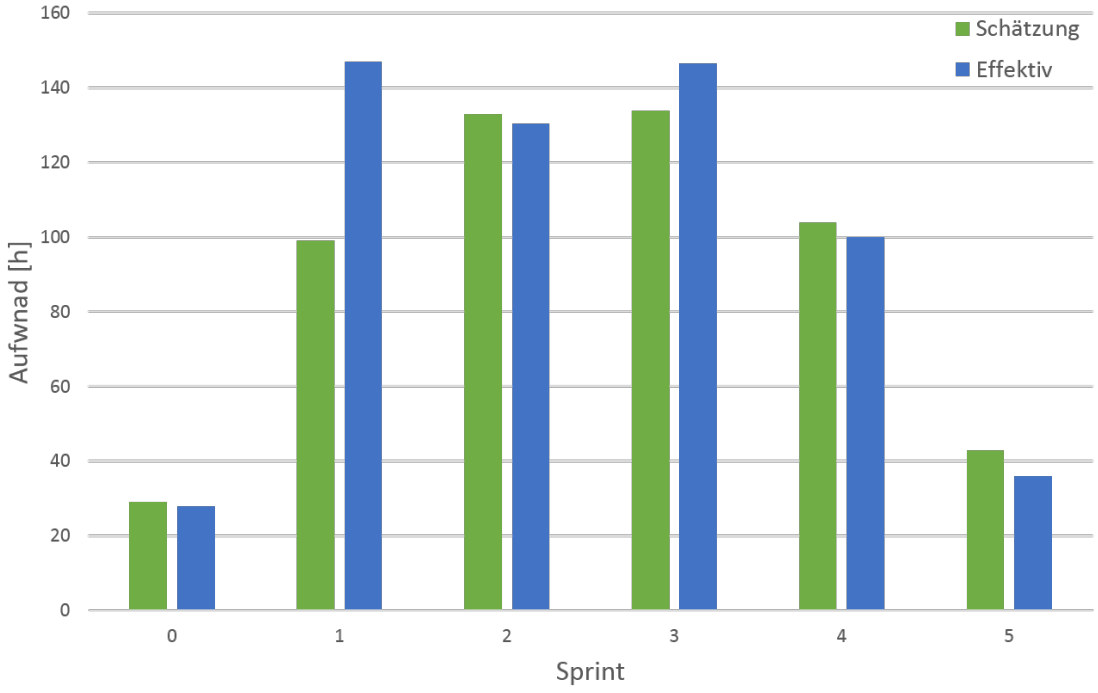


Abbildung 10.2: Aufwand pro Sprint

Aufwand pro Woche

In der Abbildung 10.3 sind die Aufwände aggregiert pro Mitglied und Arbeitswoche aufgeführt.

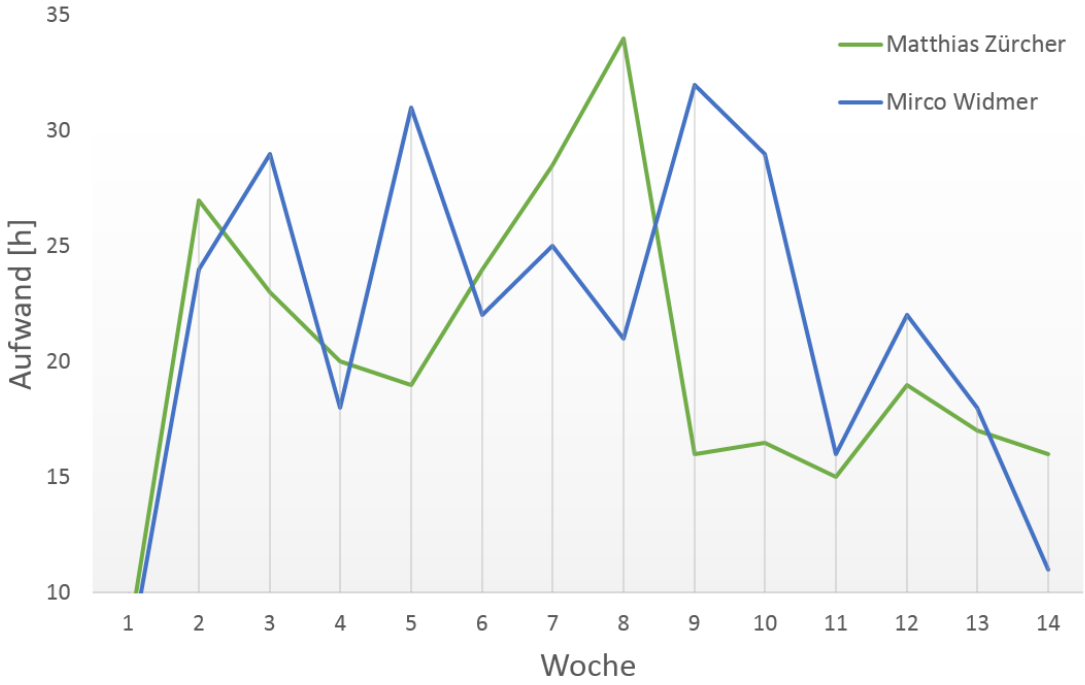


Abbildung 10.3: Aufwand pro Teammitglied und Woche

10.4.2 Übersicht Arbeitspakete

In der nachfolgenden Tabelle sind alle Arbeitspakte aufgelistet mit den geschätzten, sowie den tatsächlichen Aufwände.

#	Titel	Sprint	Schätzung [h]	Realität [h]
28	Projektplan erstellen	0	4	4
29	MATSim-Tutorial durcharbeiten	0	20	18
30	Projekt-Setup	0	5	6
31	Besprechungen	1	6	6
32	Memcached Prototyping	1	8	17
33	Hive Prototyping	1	10	16
34	Cassandra Prototyping	1	10	1
40	Besprechungen	1	8	13
41	Modellierung der Datenstruktur	1	20	36.5
42	Native Prototyping	1	32	47.5
43	Aufsetzen Maven	1	5	10
44	Besprechungen	2	16	4.5
45	Technischer Bericht	2	20	35
46	Modellierung der Datenstruktur	2	20	26
47	Implementierung	2	40	30

#	Titel	Sprint	Schätzung [h]	Realität [h]
48	Key-Value-Store-Setup	2	20	13
52	realistische Szenarien (mittel)	2	2	7
54	Filter auf Activities (Demo)	2	5	2
68	Darstellung analysierter Bereich (Grid)	2	10	4
49	Besprechungen	3	10	5
53	realistische Szenarien (gross)	3	3	4
55	GUI Redesign	3	10	9
56	Verwendung von eigenen Keys	3	10	28
57	Intelligentes Abspeichern der Events	3	20	5
58	Erstellung Szenario-Konfigurationsfiles	3	5	15
59	Refactoring Persist Package	3	10	15
60	GUI Darstellung Netzwerk	3	10	6
61	Parsing Networkfile	3	4	3
62	TimeSlices erweitern um Zeitrange	3	3	4.5
63	Filter-Mechanismus	3	5	6
64	GUI einfache Spider-Analyse	3	10	10
65	GUI erweiterte Spider-Analyse	3	10	10
66	Datastore-Erweiterung	3	8	15
67	Sektor-Aufteilung Activities	3	10	15
69	WAR-Export auf Tomcat	3	4	4
70	Aktualisierung POM	3	2	1
71	ÖV von Spider Analyse ausschliessen	4	2	2
72	Aggregierte Werte Activities darstellen	4	5	5
73	Spider erweitern um Personen-Filter	4	6	4
74	Visualisierung Teil-Netzwerk	4	10	6
75	Lade-Hinweis darstellen (GUI)	4	3	5
76	Koordinate durch Map-Klick setzen	4	2	3
78	Bericht - Implementierung	4	15	5
80	Parsen des öffentlichen Verkehrs	4	20	17
82	Implementierung der ConsoleApp	4	10	14
83	Spider Analysis - Altersverteilung	4	4	20
85	Gestaltung Plakat	4	9	4
86	Erweiterung der Filter	4	3	8
87	Javadoc vervollständigen	4	5	5
88	Code Review Feedback umsetzen	4	10	13
77	Bericht - Design	5	6	8
79	Bericht - Diverses	5	15	4
81	Bericht - Review	5	5	4
84	Implementierung Beispielanalyse	5	17	9

10.5 Sitzungsprotokolle

PROTOKOLL KICK-OFF 17.09.2012

BETEILIGTE PERSONEN

Dr. Michael Balmer, Dr. Marcel Rieser, Prof. Dr. Luc Bläser, Matthias Zürcher, Mirco Widmer

ANSPRECHSPERSONEN SENOZON AG

M. Rieser: File-Formate, bestehende API, technische Aspekte

M. Balmer: Kundensicht, Abfragen

NÄCHSTE SCHRITTE

M. Zürcher, M. Widmer

- Einarbeitung Aufgabenstellung
- Projektplan erstellen (Risikoanalyse)
- Aufteilung der Kompetenzen
- Aufsetzen SVN-Repository (HSR)
- Bestellung eines Jenkins-Build-Server (HSR)

M. Rieser

- Lieferung Beispiel-Files
- Bereitstellung Austauschplattform für grosse Files

TERMINE

- Skype-Konferenz alle 2 Wochen von 13 – 14 Uhr, beginnend ab 24.09.2012, alternativ auch vor Ort
- Besprechung zwischen L. Bläser, M. Zürcher und M. Widmer über Arbeitsfortschritt, alle 2 Wochen von 13 – 14 Uhr, beginnend ab 1.10.2012, bei Bedarf auch häufiger.

PROTOKOLL STATUSMEETING 24.09.2012

BETEILIGTE PERSONEN

Dr. Michael Balmer, Dr. Marcel Rieser, Prof. Dr. Luc Bläser, Matthias Zürcher, Mirco Widmer

ENTSCHEIDUNGEN

- Verwendung von Java 6 (falls nötig und mit Begründung Java 7)
- SVN Backup wird durch Senozon sichergestellt

NÄCHSTE SCHRITTE

M. Zürcher, M. Widmer

- Redmine Account für Marcel und Michael erstellen
- Architekturstudie
    - Lucene
    - Hive
    - …
- MATSim Analysen vertiefen

M. Rieser, M. Balmer

- Generierung Via Lizenz
- Zusammenstellung wichtiger MATSim Event Typen/Attribute
- Informationen über Analyse innerhalb MATSim
- Ergänzende Technologietypen (nicht dringend)

TERMINE

- Technische Besprechung: 2. Oktober 2012 14:00 Uhr bei Senozon (ETH Hönggerberg)

PROTOKOLL STATUSMEETING 08.10.2012

BETEILIGTE PERSONEN

Dr. Michael Balmer, Dr. Marcel Rieser, Prof. Dr. Luc Bläser, Matthias Zürcher, Mirco Widmer

ENTSCHEIDUNGEN

- Domainmodell
  - Verbindung zwischen Trip und Person erstellen. (Eine Person hat mehrere Trips und ein Trip gehört zu genau einer Person)
  - Eine Spinnenanalyse behandelt immer nur einen Trip und nicht nur einen Leg (Trip besteht aus mehreren Legs)
- Überprüfung der Ergebnisse auf Korrektheit
  - Falls vorhanden bestehende Analysen aus Matsim verwenden.
  - Am Anfang einfache und kleine Analysen verwenden.
  - „grep“ oder „sed“ verwenden um manuell in den xml Files die Resultate zu zählen und mit der Analyse zu vergleichen.
- Aktivitäten
  - Ob eine Aktivität zur Analysefläche gehört entscheidet die Koordinate und nicht der Link welchem die Aktivität zugeordnet ist.
  - Gleiche Aktivitäten derselben Person können durch die Reihenfolge im xml und durch die Zeit (falls angegeben) unterschieden werden.
  - Räumliche Abfragen
    - Beim Aufbauen der Analysen berücksichtigen, dass später auch andere Flächen als nur einen Kreis gewünscht sein könnten. Beispielsweise ein Polygon oder ein Quadrat.
- Übergreifende Szenarien
  - Der Spezialfall Park+Ride (Reise mit MIV und ÖV) kann vorläufig vernachlässigt werden.

NÄCHSTE SCHRITTE

M. Zürcher, M. Widmer

- Technischer Bericht mit Szenarien ausarbeiten und versenden
- Für Milestone 1: Prototyp für Szenario 1
  Zuerst Machbarkeit analysieren. Später können dann weitere Optimierungen eingebaut werden.

M. Rieser, M. Balmer

- Informationen zu räumlichen Abfragen (Bereits erhalten. Vielen Dank!)
- Feedback zu Szenarien (beschrieben in technischem Bericht) geben

PROTOKOLL STATUSMEETING 05.11.2012

BETEILIGTE PERSONEN

Dr. Michael Balmer, Dr. Marcel Rieser, Prof. Dr. Luc Bläser, Matthias Zürcher, Mirco Widmer

ENTSCHEIDUNGEN

- Parsing
  - Aktuelle Zeit für ca. 3 GB Events ist 10-15 Minuten. Dies erfüllt die Anforderung.
  - Granularität der Time Slices in Berkeley DB bis zu 15 Minuten.
- Priorisierung
  - Priorität 1
    - GUI für Spinnenanalyse
    - Refinement der Architektur
    - CLI für Public Transport
  - Priorität 2
    - GUI Public Transport
- Technologien
  - JSF für Webapplikation ist in Ordnung.

TERMINE

- Kein Statusmeeting am 19. November 2012
- 22. November 2012 16:00 Uhr bei Senozon
  - API (Analyzer Klasse)
  - Datenstruktur (wichtigste Abbildungen)
  - Besprechung der Anforderungen an die Showcases

NÄCHSTE SCHRITTE

M. Zürcher, M. Widmer

- GUI für Spinnenanalyse
- Refinement der Architektur
- CLI für Public Transport

M. Rieser, M. Balmer

- Anforderungen an Showcases
- Bereitstellung eines grossen Szenarios

PROTOKOLL STATUSMEETING 22.11.2012

BETEILIGTE PERSONEN

Dr. Marcel Rieser, Prof. Dr. Luc Bläser, Matthias Zürcher, Mirco Widmer

BESPROCHENE THEMEN

- Demonstration
  - Analyzer API
  - GUI Showcases
- Entscheidungen
  - Im grossen Datensatz (Schweiz) sind LinkIds und Streckenabschnitte 1:1 abgebildet.
    Zu diesem Zweck werden zum Passagier erweiterte Informationen gespeichert, damit für zukünftige Datensätze eine verfeinerte Unterteilung möglich ist.
  - ÖV wird von Spider Darstellung ausgeschlossen.
  - Fahrzeuglenker im ÖV werden nicht mitgezählt.
  - Es werden keine zusätzlichen Abfragen im GUI abgebildet.

TERMINE

- 3. Dezember 2012 – Skype Statusmeeting mit Senozon

NÄCHSTE SCHRITTE

M. Zürcher, M. Widmer

- Visualisierung Netzwerk (nur Links ab einer gewissen Capacity werden dargestellt)
- Durch User-Aktion gesamtes Netzwerk in Ausschnitt laden
- vorbereitete Filter umsetzen
- Klick setzt Koordinate
- Summe Aktivitättypen in Tabelle
- Lade-Hinweis
- Technischer Bericht
- API dokumentieren

Kapitel 11

Anhang

11.1 Verwendung von ausgewählten Technologien

11.1.1 Memcached

Start/Beenden

Memcached kann wie folgt gestartet werden:

```
memcached –d –m 1000 localhost –p 11211
```

-d Starten des Prozesses als Dämon

-m xxxx Zuweisung von Arbeitsspeicher in Megabytes

-p xxxxx TCP und Portnummer

Da der Prozess über den Activity Monitor/Taskmanager sichtbar ist, kann dieser so einfach beendet werden.

Telnet

Der Zugriff über Telnet erfolgt standardmässig:

```
telnet localhost 11211
```

Danach können mittels `stats` Informationen über die aktuelle Auslastung von Memcached gewonnen werden.

Um ein Key/Value-Paar zu speichern, gibt es folgenden Befehl:

```
set key 0 60 5
value
```

Der zweite Wert gibt an, wie lange das Paar gespeichert werden soll. [Sekunden]
Der dritte Wert gibt an, wie gross der Value ist. [Bytes]

11.2 Aufgabenstellung Senozon AG

senozon
understanding mobility

Hochschule für Technik
Rapperswil

Studienarbeit «Komplexe aber schnelle Analyse agentenbasierter Simulationsresultate»

Aufgabenstellung
30. August 2012

1 Ausgangslage und Zielsetzung

1.1 Ausgangslage

Die Senozon AG ist ein ETH Spin-off, spezialisiert auf Anwendungen mit der agentenbasierten Verkehrs- und Verhaltenssimulation MATSim [MATSim]. Diese Simulation generiert Resultatedateien (sogenannte Events-Files), die für grosse Szenarien mehrere Gigabyte gross sein können, selbst in komprimierter Form. Diese Daten sind von sehr einfacher Art und Weise. Für Simulationsanalysen sind die Daten deshalb praktisch immer in der einen oder anderen Form zu aggregieren.
Bislang implementierte Analysen lesen jeweils sequentiell das komplette Events-File ein und extrahieren fortlaufend die gewünschten Daten. Komplexere Analysen können zusätzliche Datenquellen voraussetzen (z.B. Informationen zum Strassennetz, zu Personen, Gebäuden, Fahrplänen), oder mehr als einen Lesevorgang der Events benötigen. Entsprechend den Datenmengen und der oftmals sequentiellen Verarbeitung der Daten sind solche Analysen zeitaufwändig und können nicht zur interaktiven Analyse verwendet werden.

1.2 Zielsetzung

Ziel der Studienarbeit ist es, Datenstrukturen und Algorithmen zu entwickeln, um einige vorgegebenen Analysen innert einer Zeitspanne zu berechnen, die auch den Einsatz in interaktiven Analysetools ermöglicht. Zur Demonstration sollte eine einfache Webapplikation entwickelt werden, die für eine oder zwei Analysen eine interaktive Abfragemöglichkeit zur Verfügung stellt und die entsprechenden Resultate darstellt.

2 Detailbeschreibung

2.1 Gewünschte Auswertungen

Berechnung von Strassenbelastungen: Berechnung der Anzahl Fahrzeuge auf allen Kanten oder Teilen des Strassennetzes. Die Belastungen sollten für beliebige Zeiteinheiten berechnet und in beliebige Fahrer-Kategorien unterteilt werden können. Fahrer-Kategorien werden durch ein oder mehrere Personenattribute definiert, z.B. alle männlichen Fahrer unter 25 Jahren, oder alle Fahrzeuglenker zwischen 30 und 40 Jahren die erwerbstätig sind. Denkbar wäre auch eine Fahrer-Kategorie auf Grund räumlicher Unterteilung, z.B. alle Fahrer welche in einer Region oder einer Kante unterwegs sind.
Berechnung von ÖV-Belastungen: Berechnung der Anzahl Passagiere im öffentlichen Verkehr auf Streckenabschnitten oder Netzwerkkanten. Die Belastungen sollten für beliebige Zeiteinheiten berechnet und in beliebige Passagier-Kategorien unterteilt werden können. Passagier-Kategorien werden durch ein oder mehrere Personenattribute definiert, z.B. alle Schüler und Studenten (=alle Personen, die eine entsprechende Aktivität in ihrem Tagesplan haben), oder alle Personen die älter sind als 65 Jahre.
Berechnung der Anzahl Personen im Umkreis einer Koordinate: In der Simulation führen Personen Aktivitäten an vorgegebenen Koordinaten aus. Gegeben eine beliebige Koordinate ist zu bestimmen, wie viele Personen im Umkreis der gegebenen Koordinate eine Aktivität ausüben. Der Radius des Umkreises soll dabei frei gewählt werden können. Die Resultate sollten für beliebige Zeiteinheiten berechnet und nach Aktivitätentyp und Personen-Kategorie unterteilt werden können. Beispiel: Wie viele 25 bis 60 Jahre alte Personen führen zwischen 9:00 und 11:00 im Umkreis von 250m um Koordinate XY die Aktivität „Einkaufen" aus?

2.2 Ideen zur Umsetzung

Auf Grund der grossen Datenmengen wäre in Analogie zu relationalen Datenbanken eine Umsetzung mittels vorberechneter Indizes und passender Datenserialisierungsformaten denkbar. Es wäre dann zu prüfen, welche Indizes notwendig wären, in welcher Struktur die Daten auf der Festplatte vorgehalten

werden und welche Serialisierungstechnologie (z.B. Java Serialisierung, JSON, Protocol Buffer [ProtoBuf], Avro [Avro], ...) am passendsten ist.
Es ist unklar, ob modernere BigData-Technologien (z.B. Hive [Hive], Lucene [Lucene]) existieren, welche für die Aufgabenstellung geeigneter sind.

2.3 Webapplikation

Zur Vorführung der zu erstellenden Analysen sollte eine Webapplikation erstellt werden. Die Webapplikation sollte im Minimum die Auswahl der gewünschten Analyse und deren einfache Konfiguration (z.B. durch Auswahl einiger vorgegebener Filter) ermöglichen. Nach Knopfdruck sollte das Resultat der Analyse in möglichst kurzer Zeit dem Anwender angezeigt werden.
In der einfachsten Version könnte die Webapplikation als einfaches Webformular zur Eingabe und textueller Tabelle als Ausgabe implementiert werden. Falls zeitlich und technisch realisierbar wäre eine graphische Darstellung der Resultate, z.B. in einer Karte, wünschenswert.

2.4 Datensätze

Senozon stellt für die Arbeit die kompletten Datensätze eines einfachen Beispielszenarios zur Verfügung sowie im späteren Verlauf der Studienarbeit auch entsprechende Datensätze eines echten, grossen Szenarios. Dies beinhaltet insbesondere:
- Strassennetz im MATSim-Format (network.xml)
- Simulierte Bevölkerung inkl. einiger demographischer Attribute im MATSim-Format (population.xml)
- ÖV-Fahrplan und –Netzdaten im MATSim-Format (transitSchedule.xml)
- Simulations-Resultate im MATSim-Format (events.xml)

2.5 Technische Vorgaben

Berechnungsdauer der Analysen: Da das Ziel ist, interaktive Analysen zu ermöglichen, sollten die Berechnungen innerhalb weniger Sekunden durchgeführt werden können. Die Aufbereitung der Ursprungsdaten (z.B. Indizierung, Komprimierung, ...) und Initialisierung der Webapplikation (z.B. Erstmaliges Einlesen von Indizes) ist davon ausgenommen und kann auch längere Zeit in Anspruch nehmen.
Programmiersprache: MATSim und praktische alle von Senozon erstellte Software basiert auf Java 6. Entsprechend ist eine Umsetzung der Analysen in Java erwünscht. Für die Webapplikation ist nach Absprache auch eine andere Programmiersprache (bspw. PHP) möglich. Ob und welches Framework für die Webapplikation verwendet werden soll, ist während dem Projekt abzusprechen (mögliche Vorschläge: Google Web Toolkit [GWT], Play Framework 2.x [Play], ownCloud-App [OwnCloud]).
Speicherverbrauch: Im Arbeitsspeicher sollten nur die notwendigsten Daten gehalten werden, um eine hohe Beantwortungszeit zu ermöglichen. Zum aktuellen Zeitpunkt ist der Wunsch, dass ein entsprechender Prozess pro grosses Szenario mit 4 bis 8 GB RAM laufen kann. Festplattenplatz für vorberechnete Daten sollte das Zehnfache des originalen Datensatzes nicht übersteigen.

3 Datenschutz, Rechte

Alle von Senozon zur Verfügung gestellten Daten sind Eigentum der Senozon AG und dürfen nicht weitergegeben werden. Nach Abschluss der Arbeit sind alle erhaltenen Datensätze, inklusiver Kopien auf Backups, zu löschen.
Senozon wird, falls für die Arbeit notwendig oder nützlich, Zugang zu bestehendem internen Code gewähren. Dieser Code ist und bleibt Eigentum der Senozon AG und wird nur für die Bearbeitung der Studienarbeit zur Verfügung gestellt.
Der im Rahmen der Arbeit erstellte Code geht nach Abschluss der Arbeit in den Besitz der Senozon AG über.

4 Referenzen

[Avro]	Avro data serialization system, URL avro.apache.org
[GWT]	Google Web Toolkit, URL developers.google.com/web-toolkit/
[Hive]	Hive data warehouse for Hadoop, URL hive.apache.org
[Lucene]	Open-Source search software, URL lucene.apache.org
[MATSim]	Multi-Agent Transport Simulation, URL www.matsim.org
[OwnCloud]	Private Cloud Web application, URL owncloud.org
[Play]	Web application framework, URL www.playframework.org
[ProtoBuf]	Google Protocol Buffers, URL code.google.com/p/protobuf/

Literaturverzeichnis

[1] MATSim - Multi-Agent Transport Simulation. http://matsim.org. Besucht im September 2012.

[2] NoSQL im Überblick. http://www.heise.de/open/artikel/NoSQL-im-Ueberblick-1012483.html?artikelseite=2. Besucht im Oktober 2012.

[3] Memcached, Key/Value-Store. http://memcached.org. Besucht im September 2012.

[4] Java-Client für memcached. http://code.google.com/p/spymemcached. Besucht im September 2012.

[5] Redis, Key/Value-Store. http://redis.io. Besucht im Oktober 2012.

[6] Redis - Virtual Memory deprecated. http://redis.io/topics/virtual-memory. Besucht im Oktober 2012.

[7] Hive. http://hive.apache.org. Besucht im September 2012.

[8] Hadoop. http://hadoop.apache.org. Besucht im September 2012.

[9] Cloudera. http://www.cloudera.com. Besucht im September 2012.

[10] Lucene. http://lucene.apache.org. Besucht im September 2012.

[11] Cassandra. http://cassandra.apache.org. Besucht im Oktober 2012.

[12] HBase. http://hbase.apache.org. Besucht im Oktober 2012.

[13] Übersicht Berkeley DB. http://www.oracle.com/technetwork/products/berkeleydb/overview/index.html. Besucht im Oktober 2012.

[14] Sleepycat Software Firmeninformationen. http://www.crunchbase.com/company/sleepycat-software. Besucht im Dezember 2012.

[15] Berkeley DB Beschreibung 1999. http://static.usenix.org/events/usenix99/full_papers/olson/olson.pdf. Besucht im Dezember 2012.

[16] Berkeley DB - Java Edition. http://docs.oracle.com/cd/E17277_02/html/index.html. Besucht im Oktober 2012.

[17] Oracle Berkeley DB Java Edition Datenblatt. http://www.oracle.com/technetwork/database/berkeleydb/berkeley-db-je-ds-066564.pdf. Besucht im Dezember 2012.

[18] Google Protocol Buffers. https://developers.google.com/protocol-buffers. Besucht im Oktober 2012.

[19] Primefaces. http://www.primefaces.org. Besucht im September 2012.

[20] Java Server Faces. http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html. Besucht im September 2012.

[21] Structure Analysis for Java. http://http://stan4j.com. Besucht im Dezember 2012.

[22] lambdaj. http://code.google.com/p/lambdaj/. Besucht im November 2012.

[23] Apache Maven - Build Automationstool. http://maven.apache.org. Besucht im September 2012.

[24] Apache Tomcat. http://tomcat.apache.org. Besucht im September 2012.

[25] Glassfish. http://glassfish.java.net/de. Besucht im Dezember 2012.

[26] Apache Ant - Build Automationstool. http://ant.apache.org. Besucht im September 2012.

[27] Open Street Map. http://openstreetmap.org. Besucht im Oktober 2012.

[28] Log4j. http://logging.apache.org/log4j/2.x/. Besucht im November 2012.

[29] Cobertura Test Coverage. http://cobertura.sourceforge.net. Besucht im September 2012.

[30] Checkstyle. http://checkstyle.sourceforge.net. Besucht im November 2012.

[31] UCDetector. http://www.ucdetector.org. Besucht im November 2012.

[32] findBugs. http://findbugs.sourceforge.net. Besucht im November 2012.

[33] via - Visualisierungs- und Analyse-Programm. http://senozon.com/de/produkte/via. Besucht im September 2012.