Entwurf und Entwicklung eines Visualisierungssystems für eine interaktive Verkehrsplanungsplattform

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Science (M. Sc.)

an der

Beuth Hochschule für Technik Berlin
Fachbereich Informatik und Medien
Studiengang Medieninformatik

Erstbetreuer: Dr. Klaus-Peter Eckert (Fraunhofer FOKUS)
Zweitbetreuer: Dipl.-Inform. Ilse Schmiedecke (Beuth Hochschule)

Eingereicht von
Mohamed Amine Mastouri, B. Sc.
Berlin, 15. September 2015

Abstract

The IT-supported design of the city of the future in the context of “smart cities” is a current political issue due to climate change and urban growth. Urban and transport planning have to make informed decisions to meet current and future demands with regard to mobility, emission of pollutants, and quality of life.

The present thesis arose in connection with the Streetlife project which develops an information system for the provision of mobility data for informed and sustainable urban and transport planning. This work’s aim is the design and the development of a visualization system that generically visualizes mobility data with temporal and geographical references from heterogeneous and distributed sources for the Streetlife platform.

The visualization system was designed according to the model of a data warehouse system. It enables a simple federalized connection of new data sources via a push-based websocket communication regardless of the utilized data format and access technology. The developed concept for a lossfree transformation in a generic data format enables a uniform and comprehensive handling of the data irrespectively of their origin. By means of the implementation of a variety of data analysis capabilities, the data is appropriately visualized taking into account their temporal and geographical references, filtered, and evaluated with the aid of common statistic functions. Moreover, a plug-in mechanism for the integration of specific evaluation functions has been developed. It expands the capabilities of the function-based evaluation to application-specific analyses.

The implementation was realized in a structured and modular manner to achieve high maintainability and reusability. The back-end of the visualization system was realized as a Java EE application that provides its functionalities via a RESTful web service. A JavaScript-based single-page web application that was developed using AngularJS provides the functionalities of the Java EE application via a user interface in the front-end.

The developed visualization system constitutes a first functional prototype that was tested by partners of the Streetlife project. In the outlook, ideas for improvement and extension of the functionalities are presented.

Kurzzusammenfassung

Die IT-unterstützte Gestaltung der Städte der Zukunft im Kontext von „Smart Cities" ist aufgrund der Klimaerwärmung und des Städtewachstums ein aktuelles Thema in der Politik. Die Verkehrs- und Stadtplanung muss fundierte Entscheidungen treffen, um den heutigen und zukünftigen Anforderungen moderner Städte hinsichtlich der Mobilität, Schadstoffemission und Lebensqualität gerecht werden zu können.

Diese Arbeit entstand im Zusammenhang mit dem Streetlife-Projekt, welches ein Informationssystem zur Bereitstellung von Mobilitätsdaten für eine informationsbasierte und nachhaltige Verkehrs- und Stadtplanung entwickelt. Das Ziel der Arbeit ist der Entwurf und die Entwicklung eines Visualisierungssystems zur Integration in die Streetlife-Plattform, das Daten mit Zeit- und Ortsbezug aus verteilten heterogenen Quellen generisch visualisiert.

Das Visualisierungssystem wurde nach dem Vorbild eines Data Warehouse Systems entworfen. Es ermöglicht eine einfache föderalisierte Anbindung neuer Datenquellen unabhängig vom verwendeten Datenformat und der verwendeten Zugriffstechnologie über eine Push-basierte WebSocket-Kommunikation. Das entwickelte Konzept zur verlustfreien Transformation in ein generisches Datenformat ermöglicht einen einheitlichen, datensatzübergreifenden Umgang mit den Daten ungeachtet ihrer Herkunft. Durch die Implementierung einer Reihe von Möglichkeiten zur Datenanalyse werden die Daten unter Berücksichtigung ihres Orts- und Zeitbezugs in geeigneter Weise visualisiert, gefiltert und durch allgemeine statistische Funktionen ausgewertet. Es wurde zudem ein Plug-In-Mechanismus zur Integration spezifischer Auswertungsfunktionen entwickelt, der die Möglichkeiten der funktionsgestützten Auswertung um anwendungsspezifische Analysen erweitert.

Die Implementierung erfolgte strukturiert und modular, um eine hohe Wartbarkeit und Wiederverwendbarkeit zu erreichen. Das Backend des Visualisierungssystem wurde als Java-EE-Anwendung realisiert, welche die Funktionalitäten über einen RESTful-Webservice bereitstellt. Eine mit AngularJS entwickelte JavaScript-basierte Single-Page-Anwendung stellt die Funktionalitäten der Java-EE-Anwendung über die Benutzerschnittstelle bereit.

Das entwickelte Visualisierungssystem stellt einen ersten funktionsfähigen Prototypen dar, der durch Partner des Streetlife-Projekts getestet wurde. Im Ausblick werden Ideen zur Optimierung und Erweiterung der Funktionalitäten vorgestellt.

Inhaltsverzeichnis

Inhalte der CD

Dokumentationen:

- JavaScript Dokumentation des Rohdaten-Service
- Java Dokumentation des Backends
- Java Dokumentation der spezifischen Auswertungsfunktion
- Java Dokumentation des spezifischen Transformers für die Rovereto-Daten
- JavaScript Dokumentation des Frontends

Quelltexte:

- Quelltext des Rohdaten-Service
- Quelltext des Backends
- Quelltext der spezifischen Auswertungsfunktion
- Quelltext des spezifischen Transformers für die Rovereto-Daten
- Quelltext des Frontends

Weiteres:

- Schriftliche Ausarbeitung als PDF
- Installationsanleitung des Visualisierungssystems
- Screenshot der Benutzeroberfläche
- Verwendete Onlinequellen

Abkürzungsverzeichnis

API	Application Programming Interface
CDI	Context and Dependency Injection
CRUD	Create, read, update, delete
DAO	Data Access Object
DI	Dependency Injection
DOM	Document Object Model
DW	Data Warehouse
DWS	Data Warehouse System
EAR	Enterprise Application Archive
EJB	Enterprise JavaBeans
ELT	Extract, Load, Transform
ETL	Extract, Transform, Load
FTP	File Transfer Protocol
GIS	Geografisches Informationssystem
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifikationsnummer
IIS	Internet Information Services
ISO	International Organization for Standardization
JAR	Java Archive
Java EE	Java Platform, Enterprise Edition
JavaDOC	Java Dokumentation
JAX-RS	Java API for RESTful Webservices
JPA	Java Persistence API
JPQL	Java Persistence Query Language
JsDOC	JavaScript Dokumenation
JSF	JavaServer Faces
JSON	JavaScript Object Notation
KVP	Key-Value-Pair
MVC	Model-View-Controller
OGC	Open Geospatial Consortium
ORM	Object-Relational Mapping
OSM	OpenStreetMaps
POJO	Plain Old Java Object
POM	Project Object Model
REST	Representational State Transfer
SLF4J	Simple Logging Façade for Java

SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SPA	Single-Page Webanwendung
SQL	Structured Query Language
TC 211	Technisches Kommitee 211
TCP/IP	Transmission Control Protocol/Internet Protocol
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAR	Web Application Archive
XML	Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Wissenschaftliche Motivation

Die städtische Verkehrsplanung der Vergangenheit prägt das Erscheinungsbild der gegenwärtigen Städte. Zu Zeiten des Wirtschaftswunders war die Planung stark auf den Automobilverkehr fokussiert. Durch den Wandel in der Nutzung von Verkehrsmitteln, das wachsende Umweltbewusstsein sowie das Aufkommen multimodaler Mobilität und Sharing-Angeboten muss die urbane Verkehrsplanung neuen Anforderungen gerecht werden. Sie soll nachhaltig sein, für eine ausgewogene Verkehrsauslastung sorgen und den Individualverkehr optimieren. Das private Auto verliert in Metropolen zunehmend an Bedeutung. Die Komplexität des Verkehrs durch das Angebot an unterschiedlichen Verkehrsmitteln, Park & Ride Stationen, Car-Sharing und Leihfahrrädern übersteigt die bisherigen Planungsmöglichkeiten. Der Herausforderung einer integrierten modernen Mobilitätsplanung zur Gestaltung des zukünftigen Stadtbildes kann die Verkehrsplanung nur mithilfe digitaler Lösungen gerecht werden. (Wilde, 2015)

Für eine vorausschauende und nachhaltige Verkehrs- und Umweltpolitik sind die Erfassung, Aufbereitung, Visualisierung und Auswertung von Mobilitätsdaten notwendig. Informationstechnische Werkzeuge können die dafür benötigten Funktionalitäten bieten und somit die Beantwortung diverser Fragestellungen des verkehrs- und stadtplanerischen Kontextes ermöglichen. In dieser Arbeit soll daher eine IT-Infrastruktur zur Visualisierung und Analyse beliebiger heterogener und verteilter Daten mit geografischem und zeitlichem Bezug entwickelt werden, die insbesondere zur Verkehrs- und Stadtplanung eingesetzt werden kann. Angestellte und Entscheidungsträger aus diesen Bereichen bilden die Hauptbenutzergruppe.

1.2 Problemstellung und Zielsetzung

Das Ziel dieser Arbeit ist der Entwurf und die prototypische Entwicklung eines Visualisierungssystems für die interaktive Verkehrsplanungsplattform des Streetlife[1] Projekts. Streetlife ist ein laufendes EU-gefördertes Forschungsprojekt in Kooperation mit dem Fraunhofer-Institut für Offene Kommunikationssysteme FOKUS zur Entwicklung eines multimodalen Informationssystems zur Bereitstellung von Mobilitätsdiensten für Verkehrs- und Städteplaner sowie Verkehrsteilnehmer. Ausgehend von Verkehrs- und Umweltdaten aus den

[1] http://www.streetlife-project.eu/.

[2] http://www.goberlin-projekt.de.

Partnerstädten Rovereto, Tampere und Berlin sollen Daten beispielweise zu Verkehrsaufkommen, Parkraumnutzung oder Schadstoffbelastung erfasst und für die spätere Auswertung und Visualisierung bereitgestellt werden. Aktuell liegen nur Daten zur Parkraumbewirtschaftung aus Rovereto vor, die für die Durchführung dieser Arbeit zur Verfügung gestellt wurden. Daher wird ein weiterer Datensatz aus einem anderen Projekt in die Arbeit einbezogen. Im BMWi-geförderten Projekt goBerlin[2] wurde unter anderem ein Bürgerdienst zur Visualisierung der Kenndaten Berliner Schulen entwickelt. Die erwähnten Projekte und die vorliegende Arbeit beschäftigen sich mit der Entwicklung IT-unterstützter Lösungen zur Gestaltung des zukünftigen urbanen Lebensraums und ordnen sich daher in den Forschungsschwerpunkt „Smart City“ ein.

Die zur Verfügung gestellten Datensätze stammen aus verteilten Datenquellen und liegen in unterschiedlichen Formaten vor. Das Streetlife-Projekt stellt die Daten als Dateien im JSON-Format bereit. Die Datenbasis des Bürgerdienstes liegt im XML-Format vor und ist alternativ über ein Webservice API ansprechbar. Die Inkonsistenz der Datenformate stellt ein Problem bei der Verarbeitung und Aufbereitung der Daten dar. Wichtige und zentrale Aspekte der Arbeit sind daher der anpassbare Zugriff auf in verschiedenen Formaten und Technologien gespeicherte Rohdaten sowie die Transformation der Datenformate in ein im Rahmen dieser Arbeit zu entwickelndes generisches Datenmodell. Dies ermöglicht den einheitlichen Umgang des Visualisierungssystems mit Daten unterschiedlicher Datenformate.

Die zur Verfügung gestellten Daten enthalten Informationen mit geografischem und zeitlichem Bezug. Daten mit geografischem Bezug sollen auf einer Karte verortet und durch Geometrien visualisiert werden. Die zugehörigen zeitbezogenen Informationen sollen nach Zeiträumen filterbar sein und in einem geeigneten Diagramm visualisiert werden. Ferner soll das im Rahmen der Arbeit zu entwickelnde Visualisierungssystem eine funktionsgestützte Datenanalyse der zeitbezogenen Informationen eines Kartenobjekts ermöglichen, deren Ergebnisse angezeigt werden. Elementare statistische Auswertungsfunktionen zur Bestimmung des Maximums, des Minimums und des Durchschnittswerts werden fest implementiert. Darüber hinaus wird ein Mechanismus zur Anbindung und Verwendung spezifischer Auswertungsfunktionen entwickelt, um anwendungsspezifische Analysen zu ermöglichen.

Die Analyse der visualisierten Daten bildet eine vom Anwender leicht zu erschließende informationsbasierte Entscheidungsgrundlage, auf welcher fun-

[2] http://www.goberlin-projekt.de.

dierte und nachhaltige Entscheidungen in der Politik und der Verwaltung zur Gestaltung der Verkehrs- und Stadtplanung getroffen werden können.

1.3 Struktur der Arbeit

Dem Einleitungskapitel folgt Kapitel 2, welches sich den theoretischen Grundlagen widmet, die für den Entwurf und die Entwicklung des Visualisierungssystems im Rahmen der Aufgabenstellung relevant sind. Ausgehend von der Problematik der Heterogenität von Daten aus unterschiedlichen Quellen werden Konzepte zur Handhabung, Analyse und Visualisierung von uneinheitlich strukturierten Daten mit geografischem und zeitlichem Bezug erläutert. Ein Teil des Kapitels beschäftigt sich mit den Technologien zur Umsetzung des Visualisierungssystems. Es werden die in dieser Arbeit verwendeten sowie alternative Technologien vorgestellt und kritisch gegeneinander abgewogen.

In Kapitel 3 werden im Rahmen einer Anforderungsanalyse die funktionalen und nicht-funktionalen Anforderungen an das Visualisierungssystem identifiziert und beschrieben. Die abgeleiteten Aufgaben des Visualisierungssystems werden anhand von Use Cases veranschaulicht. In diesem Kapitel werden außerdem die Datenstrukturen der zur Verfügung gestellten Datensätze analysiert, da diese durch einen zu entwickelnden Mechanismus zu vereinheitlichen sind.

Im vierten Kapitel zur Konzeption des Visualisierungssystems wird die logische Sicht auf das System dargestellt. Die Konzeption beinhaltet insbesondere das generische Datenmodell zur Vereinheitlichung der unterschiedlichen Datenmodelle und die Grobarchitektur des Systems.

In Kapitel 5 wird der Entwurf des Visualisierungssystems erläutert. Die Spezifikationen und Bestandteile des Systems werden aus funktionaler Sicht technologieunabhängig beschrieben. Darüber hinaus wird begründend auf die getroffenen Designentscheidungen eingegangen.

Die Implementierung ist im sechsten Kapitel dargelegt. Es wird auf aufgetretene Probleme und die gewählten technischen Lösungen eingegangen.

In Kapitel 7 wird das entwickelte Visualisierungssystem evaluiert. Die Ergebnisse und Lösungskonzepte der Arbeit werden bewertend diskutiert. Das letzte Kapitel gibt einen Ausblick auf künftige Weiterentwicklungsmöglichkeiten.

2 Grundlagen

2.1 Daten und Informationen

Daten und Informationen sind fundamentale Bestandteile der Informationstechnologie. Daten sind eine formalisierte maschinenlesbare Repräsentation von Informationen in einem definierten Format (Mertens et al., 2012). Der Inhalt der Informationen wird in Form von Zeichen kodiert und in einer definierten Struktur abgebildet. Daten werden verwendet, um einen Informationsaustausch zu ermöglichen. Informationen können durch die Persistierung von Daten gespeichert werden. (Bartelme, 2005)

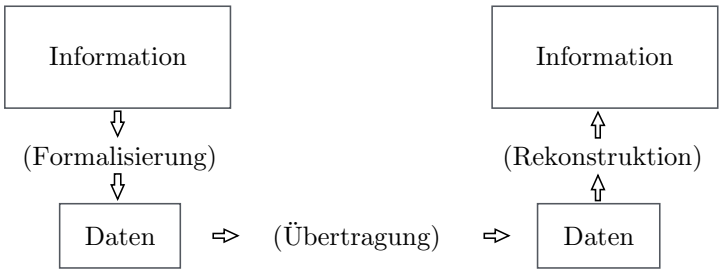

Abbildung 1: Zusammenhang zwischen Daten und Informationen (Bartelme, 2005)

Durch Rekonstruktion können Informationen aus Daten extrahiert werden (Vgl. Abbildung 1). Der Wert der rekonstruierten Information für den Empfänger ist abhängig von der Datenqualität. Die Datenqualität hängt von der Art und Weise der Formalisierung der Information in die Datenstruktur ab. Sie ist ein abstraktes Maß für die Relevanz, den Gehalt und die Korrektheit der Information. (Mertens et al., 2012)

2.1.1 Heterogenität von Daten

Daten werden isoliert in einem speziellen Projektkontext anhand der jeweils spezifischen lokalen Anforderungen erhoben. Aus diesem Grund sind einzelne Datenquellen verschiedenartig gestaltet. Dieser Umstand wird als Datenheterogenität bezeichnet und ist ein zentrales Problem bei der integrierten Nutzung mehrerer Datenquellen. Datenheterogenität kann aus verschiedenen semantischen und syntaktischen Gründen entstehen. (Conrad, 1997; Leser & Naumann, 2007)

Die Syntax bezeichnet die Datenstruktur und die Signatur eines Application Programming Interface (API). Zur syntaktischen Heterogenität zählen die technische und die Schnittstellenheterogenität. (Conrad, 1997)

Die Semantik umfasst das verwendete Datenmodell und das Verhalten des API. Die unterschiedlichen Datenmodelle stellen für die Modellierung von Daten unterschiedlich umfangreiche Strukturen und Semantiken bereit. Jedoch können Sachverhalte auch bei Verwendung desselben Datenmodells durch unterschiedliche Modellierungskonzepte verschieden abgebildet werden. Die Nutzung verschiedener Datenformate resultiert ebenfalls in heterogenen Daten. Des Weiteren besteht eine gewisse Freiheit bei der Benennung von Attributen oder Tabellen. Es können gleiche Bezeichnungen mit unterschiedlicher Bedeutung (Homonyme) oder unterschiedliche Bezeichnungen mit gleicher Bedeutung (Synonyme) verwendet werden. Ferner können Werte dieselbe Eigenschaft unterschiedlich repräsentieren, beispielsweise durch unterschiedliche Datentypen oder durch eine andere Darstellungsweise, Codierung, Maßeinheit oder Genauigkeit. (Conrad, 1997)

Um die Analyse heterogener Daten zu ermöglichen, müssen sie in einem definierten Zielformat vereinheitlicht werden. Die Überführung in ein solches einheitliches Zielformat bezeichnet man als Transformation. (Leser & Naumann, 2007)

Ein zentraler Aspekt dieser Arbeit ist die Transformation orts- und zeitbezogener Daten beliebiger Formate in ein generisches Datenformat, auf dem die entwickelten Filter-, Analyse- und Visualisierungsmechanismen des entwickelten Visualisierungssystems auf einheitliche Art und Weise operieren. Die Zusammenführung heterogener Daten aus unterschiedlichen Datenquellen liegt nicht im Fokus der Arbeit. Sie kann jedoch durch die Bereitstellung zugehöriger Metadaten und Transformatoren einbezogen werden.

2.1.2 Arten von Daten

Daten lassen sich anhand des Informationsgehalts ihrer Merkmale in qualitative und quantitative Daten unterscheiden. Qualitative Daten enthalten beschreibende Werte, welche sich nicht numerisch erfassen lassen, zum Beispiel einen Stadtnamen. Quantitative Daten enthalten Zahlwerte, beispielsweise eine Anzahl, mit denen Rechenoperationen durchgeführt werden können. (Oestreich & Romberg, 2014)

Die in dieser Arbeit betrachteten Daten enthalten statische und dynamische Anteile. Die statischen Daten verändern sich im Laufe der Zeit nicht oder nur selten. Ein Beispiel aus dem Verkehrsbereich ist die Lage eines Parkplatzes. Die dynamische Daten hingegen verändern sich häufig. Ein Beispiel hierfür ist die Anzahl der belegten Stellplätze auf einem Parkplatz. Dynamische Daten

existieren nur im Zusammenhang mit statischen Daten. Ein statisches Datum kann mit beliebig vielen dynamischen Daten verknüpft sein.

Für die Verkehrsplanung sind die dynamischen Aspekte innerhalb eines statischen Kontextes besonders relevant, beispielsweise der sich im zeitlichen Verlauf ändernde Belegungsstatus eines Parkplatzes, welcher ein spezielles statisches Objekt mit Ortsbezug darstellt. Der Belegungsstatus zu einem gegebenen Zeitpunkt kann in Form von dynamischen Daten mit Zeitbezug festgehalten werden.

2.1.3 Skalenniveaus

Das Skalenniveau ist eine Eigenschaft eines Merkmals, das dessen statistische Auswertbarkeit und die zulässigen mathematischen Operationen im Rahmen einer Analyse definiert. Die geläufigste Skaleneinteilung geht auf die Arbeit von Stevens aus dem Jahr 1946 zurück (Stevens, 1946). Im Folgenden werden die von Stevens definierten Skalenniveaus zusammenfassend beschrieben und die entsprechenden zulässigen Operationen angegeben (siehe Tabelle 1), da sie für die Möglichkeiten der Auswertung der betrachteten Daten von entscheidender Bedeutung sind.

Nominalskala: Nominale Merkmale sind kategorial und diskret. Sie beschreiben beispielsweise das Geschlecht. Da sie keine Wertigkeiten ausdrücken, lassen sich nicht in eine Reihenfolge bringen. Es kann jedoch die Häufigkeit, mit der eine Merkmalsausprägung auftritt, gezählt werden.

Ordinalskala: Ordinalskalierte Merkmale besitzen eine diskrete Ausprägung und können in eine Rangfolge gebracht werden. Es kann jedoch keine Aussage über die Abstände getroffen werden.

Intervallskala: Die Werte auf der Intervallskala sind metrisch, weshalb Rechenoperationen mit ihnen durchgeführt werden können. Sie sind in der Regel stetig, können jedoch diskret ausgeprägt sein. Auf der Intervallskala können Differenzen, aber keine Absolutwerte angegeben werden.

Ratioskala: Die Werte auf der Ratioskala sind metrisch und stetig. Die Ratioskala besitzt einen natürlichen Nullpunkt. Es kann daher der absolute Zahlenwert einer Variablen angegeben werden. Dies erlaubt die Verwendung aller Grundrechenarten.

Tabelle 1: Zulässige Operationen der Skalenniveaus

Skala	Operationen	Beispiel
Nominalskala	= / ≠	Geschlecht
Ordinalskala	= / ≠; < / >	Schulnoten
Intervallskala	= / ≠; < / >; + / -	Zeitpunkt
Ratioskala	= / ≠; < / >; + / -; × / ÷	Länge

Qualitative Daten werden auf Nominal- und Ordinalskalen abgebildet, quantitative Daten hingegen auf der Intervall- oder Ratioskala (Kurt, Ihlenburg, & Ott, 2007).

Die zur Anfertigung dieser Arbeit zur Verfügung gestellten dynamischen Daten sollen statistisch ausgewertet werden können. Die Attribute der Daten besitzen unterschiedliche Skalenniveaus, welche die Möglichkeiten zur statistischen Auswertung festlegen. Das Skalenniveau muss ebenfalls bei der Visualisierung der dynamischen Daten berücksichtigt werden. Im Rahmen der Arbeit werden die Möglichkeiten zur Datenanalyse auf die Auswertung und Visualisierung ratioskalierter Daten beschränkt. Sie sind jedoch prinzipiell um die Auswertung und Visualisierung qualitativer Daten erweiterbar. Zur Auswertung der ratioskalierten dynamischen Daten werden elementare statistische Auswertungsfunktion zur Berechnung des Minimums, des Maximums und des Durchschnittwerts fest implementiert. Eine Erweiterung der Analysemöglichkeiten durch anwendungsspezifische Auswertungsfunktionen ist konzeptuell vorgesehen. Ein entsprechender Plug-In-Mechanismus zur Anbindung neuer spezifischer Funktionen wird implementiert.

2.2 Data Warehouse Systeme

Das im Rahmen dieser Arbeit zu entwickelnde Visualisierungssystem orientiert sich am Konzept des Data Warehouse Systems, welches daher im Folgenden beschrieben werden soll.

Ein Data Warehouse (DW) ist eine persistente Datenbank, in der Daten aus unterschiedlichen Quellen in einem einheitlich formatierten, konsistenten Datenbestand für die Entscheidungsfindung integriert werden. Die Zusammenführung der Daten in einem einheitlichen Format ermöglicht die globale Nutzung und Verknüpfung von Daten aus normalerweise inkompatiblen Systemen. Das Konzept des Data Warehouse wurde ursprünglich als „themenorientierte, integrierte, zeitbezogene und nicht-flüchtige Datenbank zur Unterstützung von Managemententscheidungen“ definiert (Inmon, 2005). Ein Data Warehouse System (DWS) umfasst die gesamte technische Infrastruktur,

die benötigt wird, um die Daten zu beschaffen, zu speichern und zu analysieren. (Farkisch, 2011)

2.2.1 Datenintegrationsprozesse

Ein Prozess zur Integration von Daten in das DWS umfasst folgende Schritte (Farkisch, 2011):

Extraktion der heterogenen Daten aus verschiedenen Quellsystemen
Transformation der Daten in ein einheitliches Schema
Laden der transformierten Daten in das DWS

Dieser Prozess wird als ETL (Extract, Transform, Load) bezeichnet (Farkisch, 2011). In Abbildung 2 ist der Ablauf des Prozesses dargestellt. ETL ist der klassische Datenintegrationsprozess eines DWS (Kimball & Caserta, 2004).

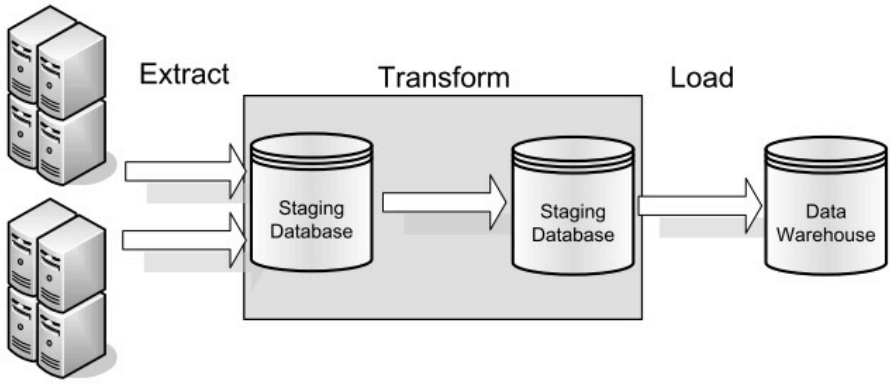

Abbildung 2: Ablauf eines ETL-Prozesses (Davenport, 2008)

Eine Weiterentwicklung des ETL-Prozesses ist Extract, Load, Transform (ELT) (Kakish & Kraft, 2012). Mit ELT werden die Daten direkt auf den Zielserver geladen und erst dort transformiert (siehe Abbildung 3). Dieser Prozess ist zur Prozessierung großer Datenmengen geeignet, da die Zugriffszeit auf die Quelle minimiert wird (GoldenGate Software Inc., 2009). Zudem kann eine exakte Kopie der Rohdaten im DWS abgelegt werden. Bei Änderung der Anforderungen an das DWS können die Transformationsregeln verändert und auf die Rohdaten angewendet werden (Davenport, 2008). Dies ist bei bereits transformierten Daten nach einem ETL-Prozess nicht mehr möglich.

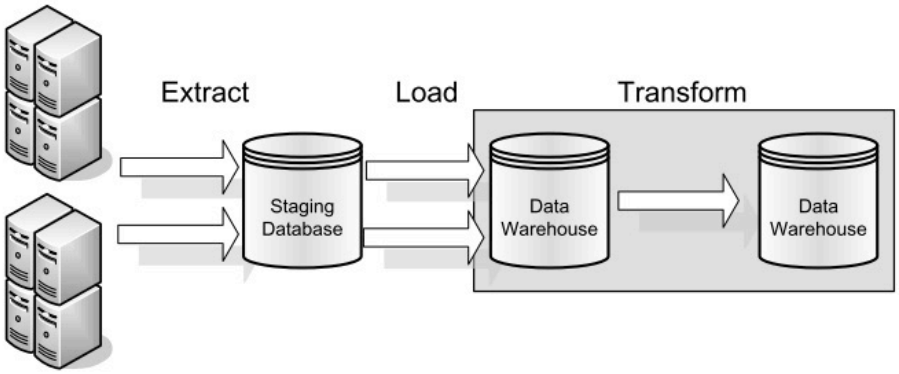

Abbildung 3: Ablauf eines ELT-Prozesses (Davenport, 2008)

2.2.2 Datenextraktion

Die Datenextraktion erfolgt über eine netzwerkbasierte Kommunikation zwischen einem Klienten und einem Server. Für die Extraktion von Daten aus einer Quelle (dem Server) in ein DWS (den Klienten) stehen unterschiedliche Mechanismen zur Verfügung. Die Wahl des Datenextraktionsmechanismus ist abhängig von den Projektgegebenheiten und -anforderungen und beeinflusst maßgeblich die Architektur des Systems. Als für diese Arbeit relevante Kommunikationsmechanismen werden Pull- und Push-basierte Mechanismen vorgestellt.

Wird der Informationsfluss vom Klienten gesteuert, handelt es sich um einen Pull-basierten Mechanismus. Der Klient sendet eine Anfrage (Request) über HTTP[3] an den Server und erhält von diesem eine Antwort (Response). Durch die spezfische Abfrage erhält der Klient nur angeforderte und für ihn relevante Informationen. Ein Beispiel für eine Anfrage mittels Pull ist das Aufrufen einer Webseite. Ein Nachteil des Pull-Mechanismus ist, dass neue Daten zeitlich versetzt zur Verfügung stehen. Dies wird als Datenlatenz bezeichnet. Oftmals reicht es aus, die Abstände zwischen den Anfragen zu verkürzen, um eine hinreichende Datenaktualität zu gewährleisten. Die Abfragefrequenz lässt sich jedoch nicht beliebig erhöhen, da der Quellserver unter der Last zu vieler gleichzeitiger Anfragen zusammenbrechen kann. Zudem ist durch die zeitliche Trennung von Datenaktualisierung und Benutzung des DWS der Zugriff auf das DWS während der Aktualisierung nicht möglich. (Bauer & Günzel, 2013)

Der Informationsfluss kann durch Push-basierte Mechanismen vom Server gesteuert werden. Ein Push-Mechanismus ermöglicht die unaufgeforderte Weiterleitung von Änderungen vom Server an den Klienten und damit einen

[3] Hypertext Transfer Protocol.

kontinuierlichen Datenfluss. Dies bezeichnet man als Streaming Data Flow. Durch eine vorhergehende Anfrage des Klienten wird eine TCP[4]-Verbindung zum Server hergestellt, die im Gegensatz zur HTTP-Verbindung bestehen bleibt und serverseitig aktiviert werden kann, sobald neue Informationen für den Klienten verfügbar sind. Der Klient muss keine neuen Anfragen stellen und die Daten sind quasi in Echtzeit verfügbar. (Goll, 2014)

Bei der Einführung des DWS bestand die Hauptaufgabe in der Analyse vergangener Ereignisse. Der Fokus verschiebt sich jedoch von der Nutzung der Analysen als Entscheidungshilfe in der langfristigen Strategieplanung auf die Verwendung im operativen Management zur schnellen Reaktion auf neue Informationen oder Änderungen in der Umgebung (Kakish & Kraft, 2012; Kimball & Ross, 2002). Dafür müssen neue Daten so schnell wie möglich verfügbar sein. Die Pull-basierten Mechanismen werden daher zunehmend durch Push-basierte Kommunikationsmechanismen abgelöst, welche die Latenzzeit der Daten minimieren. (Kimball & Ross, 2002; Simitsis, Vassiliadis, & Sellis, 2005; Tho & Tjoa, 2003)

Im Rahmen des zu entwickelnden Visualisierungssystems ist sowohl die Verwendung von Pull- als auch von Push-basierten Mechanismen denkbar. In der Konzeption wird dieser Punkt konkretisiert.

2.2.3 Datentransformation

Die extrahierten Daten stammen zumeist aus verschiedenen Quellen und liegen in heterogenen Formaten vor. Im Rahmen der Transformation werden die Daten in ein definiertes Zielformat überführt. Die Transformation kann auf verschiedenen Ebenen stattfinden. Bei der syntaktischen Transformation werden die Daten an die formale Syntax des Zielsystems angepasst. Darunter fällt beispielsweise die Änderung des Datumsformats. Bei der semantischen Transformation werden die Daten auf inhaltliche Aspekte geprüft und gegebenenfalls modifiziert. Hierzu zählen die Anpassungen der Datenkodierung oder die Umrechnung von Einheiten. (Bauer & Günzel, 2013; Kimball & Ross, 2002)

Zur Aufbereitung und Visualisierung der heterogenen Daten im zu entwickelnden Visualisierungssystem ist eine Transformation in ein einheitliches Zielformat notwendig. Die Konzeption des Transformationsmechanismus und des Datenmodells wird in Kapitel 4.2 ausführlich beschrieben.

[4] Transmission Control Protocol.

2.2.4 Datenladung

Die Daten werden in das Zielsystem geladen. Während des Ladens ist der Zugriff auf das DWS blockiert, weshalb dieser Schritt möglichst performant erfolgen muss, um die Sperrzeit zu minimieren. Im DWS wird üblicherweise eine Versionshistorie angelegt, um Änderungen zu protokollieren und auf Versionen früherer Zeitpunkte zurückgreifen zu können. (Bauer & Günzel, 2013; Kimball & Ross, 2002)

2.3 Geografische Informationen

Die zu analysierenden Daten enthalten Informationen mit geografischem Bezug, welcher bei der Visualisierung berücksichtigt werden sollen. Geografische Informationen werden standardisiert in Form von Koordinaten angegeben, welche die Lage eines Punktes auf der Erdoberfläche durch die Angabe eines Längen- und eines Breitengrads festlegen. Der Breitengrad beschreibt, wie weit nördlich oder südlich ein Ort in Relation zum Äquator liegt. Der Längengrad gibt an, wie weit westlich oder östlich ein Ort in Relation zu Greenwich (England) liegt. (Svennerberg, 2010) Koordinaten können auf Karten verortet werden.

2.3.1 Eigenschaften von Geodaten

Als Geodaten bezeichnet man Daten, die einen geografischen Bezug besitzen. Sie sind durch spezifische Eigenschaften charakterisiert, welche geometrischer, topologischer, thematischer oder temporaler Natur sein können. Geometrische Eigenschaften sind zum Beispiel die Form, die Größe und die Lage eines Gebietes auf der Erdoberfläche. Topologische Eigenschaften beschreiben die relative räumliche Beziehung von Objekten zueinander. Die Objekte können überlappen, an- oder ineinander liegen oder durch eine bestimmte Entfernung voneinander getrennt sein. Sie bleiben bei Verzerrung oder Skalierung unverändert und ermöglichen daher beispielsweise eine Routenermittlung. Thematische Eigenschaften sind Sachattribute wie Ortsnamen oder Postleitzahlen. Temporale Eigenschaften legen fest, zu welchen Zeitpunkten oder in welchen Zeiträumen die angegebenen Eigenschaften gültig sind. (Brinkhoff, 2008)

2.3.2 Geografische Informationssysteme

Als geografisches Informationssystem (GIS) bezeichnet man die Gesamtheit der zur Erfassung, Speicherung, Verwaltung, Bearbeitung und Auswertung von Geodaten benötigten Infrastruktur. Objekte, die über Geodaten verfügen, werden als Geoobjekte bezeichnet. Sie können mithilfe eines GIS in einem Ko-

ordinatensystem durch Geometrien visualisiert werden. Ihnen können neben den geometrischen und topologischen auch temporale und thematische Eigenschaften als Attribute zugewiesen werden. (Andrae, Fitzke, & Zipf, 2009)

Zur Visualisierung statischer Daten mit geografischem Bezug soll das zu entwickelnde Visualisierungssystem die Funktionalitäten eines GIS unterstützen.

2.3.3 Standardisierung von Geometriedatenmodellen

Zur Vereinheitlichung und Interoperabilität von Geoinformationssystemen wurden vom Open Geospatial Consortium (OGC) und dem Technischen Kommitee 211 (TC 211) der International Organization for Standardization (ISO) Standards und Normen zur Modellierung und Speicherung von Geodaten eingeführt (Brinkhoff, 2008). Diese Normen müssen bei der Entwicklung des Visualisierungssystem eingehalten werden. Zwei wichtige Modelle sind das Feature Geometry Modell und das Simple Feature Modell. Sie sind durch ISO-Normen definiert.

2.3.3.1 Feature Geometry Modell

Das Datenmodell der Norm ISO 19107 „Geographic Information - Spatial Schema“ beschreibt die räumlichen Eigenschaften eines Geoobjekts durch eine Vektorgeometrie und eine Topologie von maximal drei Dimensionen (ISO/TC 211, 2003). Im Modell sind räumliche Standardoperationen für Zugriff, Anfrage, Verwaltung, Verarbeitung oder den Austausch von Geoobjekten definiert.

2.3.3.2 Simple Feature Modell

Das Simple Feature Modell ist eines der wichtigsten Datenmodelle für geografische Datenbanksysteme. Das Modell beruht auf dem Feature Geometry Modell und ist in zwei ISO-Normen mit der Bezeichnung „Simple Feature Access” festgelegt. Die ISO 19125-1 stellt das allgemeine Modell vor, während die ISO 19125-2 dessen Umsetzung in einer SQL[5]-Datenbank erläutert (ISO/TC 211, 2004a, 2004b). Das Simple Feature Modell beschreibt zweidimensionale Vektorgeometrien und definiert den Datentyp sowie Operationen für den Zugriff, die Anfrage und die Verarbeitung dieser Geometrien.

Abbildung 4 gibt einen Überblick über das Geometrieschema des Simple Feature Modells, welches die Klassen zur Repräsentation von Geometrien enthält. In der Oberklasse `Geometry` werden die gemeinsamen Attribute und Methoden aller Unterklassen gebündelt. Die vier Unterklassen `Point`, `Curve`, `Surface` und

[5] Structured Query Language.

GeometryCollection leiten sich aus der Oberklasse Geometry ab. Jeder Geometrie kann ein räumliches Bezugssystem, SpatialReferenceSystem genannt, zugeordnet werden.

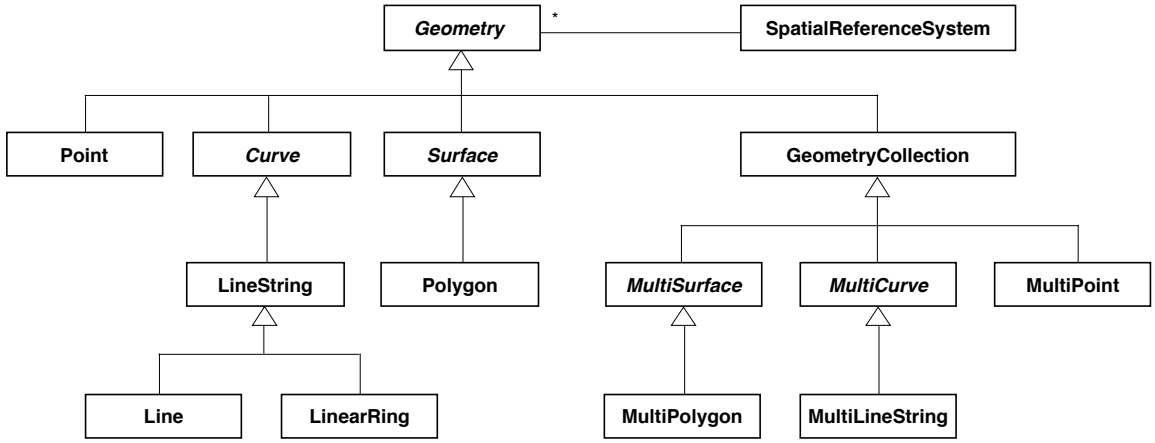

Abbildung 4: Geometrieschema des Simple Feature Modells (Brinkhoff, 2008)

Das Geometrieschema des Simple Feature Modell bildet im Entwurf eine Teilmenge des abstrakten Modells, mit dessen Hilfe die Visualisierung der Daten mit Ortsbezug ermöglicht werden soll. Die zur Verfügung gestellten Geodaten weisen nur zwei Dimensionen auf, da sie nur einen Längen- und einen Breitengrad besitzen. Auf die Betrachtung von Höheninformation wird im Rahmen der Arbeit verzichtet. Die Gebiete werden auf Grund ihrer relativ geringen Ausdehnung auf der Erdoberfläche als eben betrachtet, weshalb sie als Polygone dargestellt werden können. Das Simple Feature Modell ist ausreichend für die Darstellung dieser Daten.

2.4 Visualisierung von Informationen

Die Interpretation von Daten und Analyseergebnissen sowie die Ableitung von Entscheidungen und Handlungen aus selbigen obliegt dem Anwender des Visualisierungssystems. Obwohl es in den letzten Jahren große Fortschritte in den Bereichen des Machine Learnings und der Künstlichen Intelligenz gab, ist noch nicht das nötige Niveau erreicht, um solche komplexen kognitiven Aufgaben einem Computer zu überlassen. Die Zugänglichkeit und Interpretierbarkeit der Daten für den Benutzer sind daher die entscheidenden Aspekte für die Verwendbarkeit des Systems. Sie werden durch eine geeignete Visualisierung der Daten ermöglicht.

2.4.1 Ziel der Visualisierung

„The purpose of visualization is insight, not pictures."
(Card, Mackinlay, & Schneidermann, 1999)

Große Datenmengen in Form von Text sind für einen Anwender schwer erfassbar und auswertbar. Zur Unterstützung der Datenanalyse sollen die Daten daher visualisiert werden. Als Visualisierung bezeichnet man die computergestützte Generierung von grafischen Repräsentationen aus Daten mit dem Ziel, einen besseren kognitiven Zugang zu den enthaltenen Informationen zu ermöglichen (Card et al., 1999). Eine angemessene Visualisierung unterstützt den Erkenntnisgewinn durch die grafische Darstellung der Daten.

2.4.2 Visualisierungskonzepte

In der deskriptiven Statistik existieren zahlreiche Diagrammtypen zur grafischen Repräsentation von Daten. Sie besitzen oftmals Untertypen wie gruppierte oder gestapelte Varianten.

Die verschiedenen Diagrammtypen sind unterschiedlich gut zur Darstellung von Daten eines bestimmten Skalenniveaus geeignet (Akremi, Baur, & Fromm, 2011). Tabelle 2 zeigt, inwiefern unterschiedliche Diagrammtypen für die Visualisierung von Daten eines bestimmten Skalenniveaus geeignet sind, wobei ++ für „sehr geeignet", + für „geeignet" und – für „ungeeignet" steht.

Tabelle 2: Geeignete Diagrammarten zur Darstellung von skalierten Daten

	Ratioskala	Intervallskala	Ordinalskala	Nominalskala
Symbole	–	–	+	++
Kreisdiagramm	–	–	++	++
Streifendiagramm	+	+	++	++
Balkendiagramm	+	++	+	–
Punktdiagramm	++	+	–	–

Nominalskalierte Werte werden am besten durch Symbole, Streifendiagramme oder Kreisdiagramme dargestellt. Ordinalskalierte Werte werden abhängig von der Anzahl ihrer Merkmalsausprägung durch Balken-, Streifen- oder Kreisdiagramme anschaulich visualisiert. Intervallskalierte Werte lassen sich am geeignesten in Balkendiagrammen abbilden, aber auch die Darstellung durch Streifen- oder Punktdiagramme ist möglich. Punktdiagramme und deren Variationen wie Punkt-Linien-Diagramme eignen sich besonders zur Darstellung von ratioskalierten Werten.

Digitale Visualisierungen können interaktiv und dynamisch gestaltet werden. So können Daten mit gemeinsamen Merkmalen zum Beispiel in Cluster gruppiert und durch den Benutzer gefiltert werden. Ferner können die Grafiken bei Verfügbarkeit neuer Daten aktualisiert werden.

2.5 Technologien

In diesem Kapitel folgt die zusammenfassende Beschreibung der im Rahmen dieser Arbeit verwendeten Technologien. Es werden außerdem alternative Technologien vorgestellt und die für die Implementierung getroffene Auswahl durch eine kritische Abwägung der Vor- und Nachteile begründet.

2.5.1 Enterprise-Plattformen

2.5.1.1 Java-EE-Framework

Java ist eine der bedeutendsten Entwicklungsplattformen zur Erstellung von Anwendungen. Die Java-Technologie besteht aus der Laufzeitumgebung Java Runtime Environment, der objektorientierten Programmiersprache Java und in Form von API bereitgestellten Bibliotheken. Durch die Kompilierung in Bytecode vor der Übersetzung in Maschinencode sind Java-Programme plattform- und maschinenunabhängig mit der Java-Laufzeitumgebung ausführbar.

Die Java Platform, Enterprise Edition (Java EE) ist eine Spezifikation zur Entwicklung und Ausführung von Java-Applikationen und -Webanwendungen. Eine Java-EE-Anwendung wird prinzipiell mit einer Drei-Schichten-Architektur entworfen, welche beispielhaft in Abbildung 5 dargestellt ist. Das System ist dabei in Schichten gegliedert, welche spezifische Funktionen übernehmen. Diese Strukturierung reduziert die Komplexität des Systems und erleichtert dadurch das Verständnis der Systemarchitektur. Die Schichten sind hierarchisch aufgebaut. Eine Schicht greift nur auf die Funktionalität der ihr unmittelbar untergeordneten Schicht zur Ausführung der eigenen Aufgaben zu. (Weil, 2012)

Abbildung 5 zeigt das Drei-Schichten-Architekturmodell einer Java-EE-Anwendung. Das Frontend bildet die oberste Architekturschicht, die sogenannte Präsentationsschicht. Sie enthält die Präsentationslogik, die auf die Geschäftslogikschicht, die zweite Schicht, zugreift. Diese zweite Schicht enthält Komponenten zur Umsetzung der Geschäftslogik der Anwendung. Die Datenhaltungsschicht ist die unterste Schicht und enthält die Daten. (Weil, 2012)

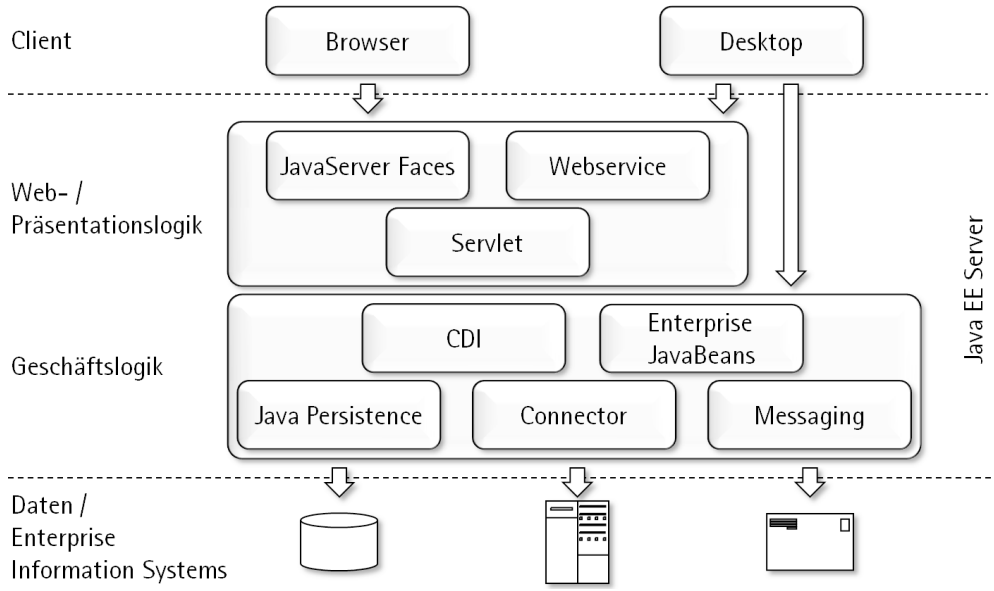

Abbildung 5: Drei-Schichten-Architekturmodell von Java EE (Weil, 2012)

Unter dem Überbegriff Java EE sind mehrere Teilspezifikationen zusammengefasst (Weil, 2012). Im Folgenden sollen diejenigen vorgestellt werden, welche in dieser Arbeit Anwendung finden.

2.5.1.1.1 Java Persistence API (JPA)

Java stellt mit dem Java Persistence API (JPA) ein DAO[6]-Muster zur Verfügung, um die Implementierung der Datenhaltungsschicht zu vereinfachen. JPA übernimmt außerdem objektrelationale Abbildungen und stellt mit JPQL[7] eine SQL-analoge Abfragesprache zur Verfügung. (Balzert, 2011)

In JPA werden Klassen durch spezifische Annotationen gekennzeichnet, beispielsweise mit `@Entity`, um sie persistent zu halten. Ein EntityManager verwaltet den Lebenszyklus der mit `@Entity` annotierten Klassen und übernimmt das Speichern und Lesen von Daten in der Datenbank. Zudem stellt er eine Schnittstelle bereit, die es ermöglicht, SQL-Anweisungen zu erzeugen und auszuführen. (Balzert, 2011)

[6] Das Data Access Object (DAO) Muster ist ein Entwurfsmuster zur Kapselung des Zugriffs auf Datenquellen (Balzert, 2011).

[7] Java Persistence Query Language.

2.5.1.1.2 Enterprise JavaBeans (EJB)

Enterprise JavaBeans (EJB) sind serverseitige Komponenten zur Implementierung anwendungsspezifischer Geschäftslogiken. Sie folgen dem Komponentenmodell der EJB-Spezifikation. (Balzert, 2011)

Eine EJB ist seit EJB 3.0 ein POJO[8] und muss daher keine speziellen technischen Schnittstellen mehr implementieren. Die Java-Objekte werden zur Deklaration mit der Annotation `@EJB` versehen. Während der Laufzeit werden sie in EJB-Containern[9] verwaltet und entsprechend der Spezifikation ausgeführt. (Rupp, 2007)

Man unterscheidet EJB in Session Beans, Entity Beans und Message Driven Beans. In dieser Arbeit werden Session Beans und Entity Beans verwendet. Entity Beans repräsentieren in einer Datenbank persistierte Daten. Session Beans werden zur Umsetzung der Geschäftslogik zur Laufzeit auf dem Server verwendet. Für eine EJB kann eine lokale Schnittstelle definiert werden, die üblicherweise durch die EJB-Klasse implementiert und mit der Annotation `@Local` versehen wird. Session Beans können über eine definierte Remote-Schnittstelle aus Java-Prozessen auf einem anderen Server heraus aufgerufen werden. Die Remote-Schnittstelle, welche mit `@Remote` annotiert ist, stellt eine einfache Schnittstelle der Bean-Klasse dar und definiert die Menge der aufrufbaren Methoden. (Balzert, 2011)

Session Beans werden mit speziellen Annotationen versehen, um anzuzeigen, um welchen Typ von Session Bean es sich handelt (Balzert, 2011). Es existieren drei Typen von Session Beans, die genauer beschrieben werden sollen (Balzert, 2011):

`@Stateless`: Stateless Session Beans sind zustandslos und rein datenverarbeitende Komponenten. Die Instanzen stehen einem Klienten nur für einen Methodenaufruf zur Verfügung. Während der Nutzung kann kein anderer Klient auf die Instanz zugreifen. Nach der Nutzung ist jedoch eine Vergabe an andere Klienten möglich.

[8] Plain Old Java Object.

[9] Ein EJB-Container befindet sich auf dem Java Anwendungsserver. In ihm werden EJB verwaltet (Balzert, 2011).

@Stateful: Stateful Session Beans sind individuell datentragende Komponenten. Die Instanzen stehen einem Klienten exklusiv und für mehrere Methodenaufrufe zur Verfügung, da sie ein sogenanntes „Konversationsgedächtnis" haben, in welchem der Zustand der Interaktion mit dem Klienten gespeichert wird.

@Singleton: Es wird nur eine Instanz einer Singleton Session Bean für eine Anwendung erzeugt. Alle Klienten arbeiten daher mit demselben Objekt.

2.5.1.1.3 Context and Dependency Injection (CDI)

Dependency Injection (DI) ist ein Entwurfsmuster in der objektorientierten Programmierung, das die Abhängigkeiten von Objekten zur Laufzeit definiert. DI überträgt die Auflösungsverantwortung an die Umgebung, die zur Laufzeit ein passendes Objekt zur Verfügung stellt. Die Instanziierung von Objekten wird nicht verteilt in der Applikation programmiert, sondern zentral von einem Container gesteuert. Ein Objekt instanziiert seine Abhängigkeiten folglich nicht selbst, sondern bekommt sie von einer externen Instanz bereitgestellt. Dies wird als Injektion bezeichnet. Durch DI wird eine lose Kopplung zwischen den Komponenten erreicht. (Goncalves, 2013)

Context and Dependency Injection (CDI) ist ein Java-Standard (JSR 346), der die Verwaltung von Modulen und die Instanziierung von Objekten durch die Injektion von Abhängigkeiten ermöglicht, also das Prinzip der DI in Java standardisiert. (Goncalves, 2013)

In Java EE gibt es eine Reihe von Containern, beispielsweise EJB- oder CDI-Container, die den Lebenszyklus ihrer Objekte beziehungsweise Beans verwalten. Eine Bean, deren Lebenszyklus in einem Container verwaltet wird, wird als Managed Bean bezeichnet. Abbildung 6 zeigt den Lebenszyklus einer Managed Bean. Bei der Injektion einer Bean mittels CDI wird vom Container eine neue Instanz erzeugt und die Abhängigkeit aufgelöst. Mit `@PostConstruct` annotierte Methoden werden vor den Geschäftsmethoden der Bean aufgerufen. Diese Annotation wird im Rahmen der Implementierung verwendet. Die `@PreDestroy` Rückrufbenachrichtigung signalisiert, dass die Instanz vom Container entfernt wird. (Goncalves, 2013)

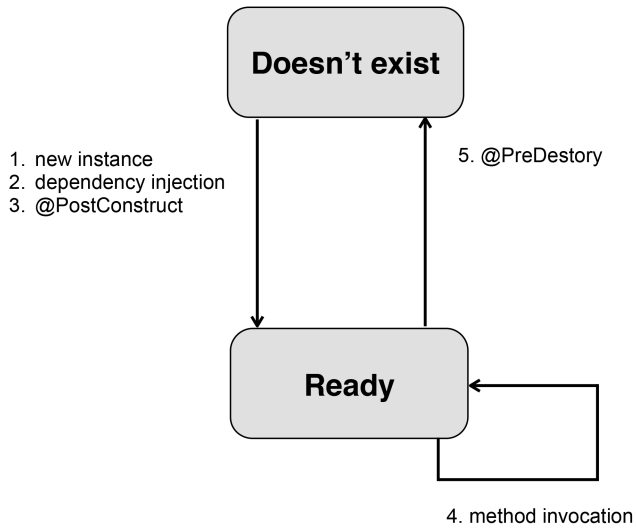

Abbildung 6: Lebenszyklus einer Managed Bean (Goncalves, 2013)

2.5.1.1.4 Java API for RESTful Webservices (JAX-RS)

Java API for RESTful Webservices (JAX-RS) spezifiziert im JSR 311 ein API zur Entwicklung von Webservices und den zugehörigen Klienten, welche mittels REST (siehe Kapitel 2.5.4.2) kommunizieren. Mit der Referenzimplementierung kann ein Servlet-Endpoint definiert werden, wodurch der RESTful Webservice in einem Servlet-Container ausgeführt wird. JAX-RS definiert ferner Annotationen, mit welchen Konfigurationen am RESTful Webservice vorgenommen werden können. (Burke, 2014)

2.5.1.1.5 Java WebSocket API

Die Java WebSocket API Spezifikation JSR 356, welche ab Java EE 7 verfügbar ist, nutzt den View des HTML5-Standards für WebSocket (siehe Kapitel 2.5.4.3), welches das Model in einem Webbrowser und das Protokoll für die Kommunikation definiert. In Bezug auf das Java API ist der WebSocket Endpoint ein Java-Objekt. Für jede Java-Applikation mit einer WebSocket-Verbindung wird eine eigene Instanz des Java-WebSockets erzeugt. (Pilgrim, 2013)

2.5.1.2 .NET Framework

Das Framework .NET ist neben Java EE eine dominierende Entwicklungsplattform zur Erstellung von Anwendungen von Microsoft, welche an Java EE angelehnt ist. Es besteht aus der Laufzeitumgebung Common Language Runtime sowie einer umfangreichen Sammlung von Klassenbibliotheken, einem API und diversen Diensten. Die Plattform unterstützt eine Vielzahl an Programmiersprachen, darunter insbesondere die an die .NET-Plattform angepassten

Sprachen C# und Visual Basic. Die Kompilierung in einen prozessorunspezifischen Zwischencode vor der Übersetzung in Maschinencode ermöglicht die Interoperabilität aller .NET-Anwendungen sowie eine gemischtsprachiger Programmierung. Diese bietet zwar eine hohe Flexibilität bei der Entwicklung, kann jedoch die Wartbarkeit des Systems beeinträchtigen. (Balzert, 2011)

Bei der Gestaltung von .NET wurde viel Wert auf die Entwicklungs- und Ausführungsgeschwindigkeit gelegt, was sich vorteilhaft auf den Ressourceneinsatz auswirkt. Ein Nachteil von .NET ist jedoch seine Plattformabhängigkeit. In vollem Umfang ist es nur für Windows verfügbar. (Balzert, 2011) Aus diesem Grund wird Java EE zur Anfertigung dieser Arbeit bevorzugt.

2.5.2 Anwendungsserver

In diesem Abschnitt werden Anwendungsserver zur Ausführung von Anwendungen vorgestellt, welche mit .NET, Java EE oder JavaScript entwickelt wurden, vorgestellt.

2.5.2.1 Java-EE-Anwendungsserver

Es existieren verschiedene Implementierungen für Java-EE-Anwendungsserver, beispielsweise GlassFish, WebLogic und WildFly (Befeld & Klein, 2010). Die Unterschiede zwischen den Implementierungen sind gering. In dieser Arbeit wird WildFly (früher JBoss[10]) verwendet, welches die bekannteste und verbreitetste Technologie ist. WildFly ist ein in Java geschriebener open-source Anwendungsserver. Er implementiert die aktuellsten Java-EE-7-Spezifikationen, welche in dieser Arbeit verwendet werden. (WildFly, 2015)

2.5.2.2 JavaScript-Anwendungsserver

Ein JavaScript-Anwendungsserver ist eine Plattform für die Entwicklung und den Betrieb von serverseitigen JavaScript-Webanwendungen (Springer, 2013). Ein Beispiel für einen solchen Anwendungsserver ist die Node.js-Plattform. Node.js ermöglicht die Entwicklung und den Betrieb skalierbarer Netzwerkanwendungen und Webserver auf Basis der schnellen JavaScript-Laufzeitumgebung V8. Es stellt dem Entwickler dazu wiederverwendbare JavaScript-Module zur Verfügung. (Joyent Inc., 2015; Springer, 2013)

Das Framework läuft asynchron und ereignisbasiert. Die ereignisbasierte Architektur wird durch die Programmiersprache JavaScript vorgegeben, auf welcher Node.js basiert. Die Architektur ist sehr ressourcensparend, sodass nur

[10] JBoss wurde 2013 in WildFly umbenannt (Red Hat Inc., 2013).

wenig Arbeitsspeicher für die Netzwerkverbindungen benötigt wird. (Springer, 2013)

Es existieren weitere Lösungen, welche sich nicht grundlegend unterscheiden und zumeist auf Node.js basieren. Meteor[11] und io.js[12] sind zwei Beispiele. In dieser Arbeit wird ein HTTP-Server auf Basis von Node.js eingesetzt, der das Frontend auf einem beliebigen Port ausliefert.

2.5.2.3 .NET Anwendungsserver

Ein .NET-Anwendungsserver dient der Bereitstellung und Ausführung von Geschäftsanwendungen, die mithilfe des .NET-Frameworks erstellt wurden. Der eigenständige „Windows Server AppFabric Server" umfasst IIS[13] und weitere Komponenten zur Verbreitung, Verwaltung und Überwachung. AppFabric ist vergleichbar mit den etablierten Java-Servern, existiert jedoch erst seit 2010. (Balzert, 2011)

Aufgrund der hohen Technologieabhängigkeit zu Microsoft Windows wurde .NET in dieser Arbeit nicht verwendet.

2.5.3 Frameworks zur Erstellung von Single-Page-Webanwendungen

Klassischerweise bestehen Webseiten aus mehreren untereinander verlinkten HTML[14]-Dokumenten. Bei jeder Anfrage wird die gesamte Benutzeroberfläche neu geladen, wobei die gesamte klientenseitige Präsentationslogik beendet und auf der nächsten Seite neu gestartet wird. (Mikowski & Powell, 2014)

Single-Page-Webanwendungen (SPA) sind klientseitige JavaScript-Applikation, die im Gegensatz zu klassischen Webseiten aus nur einem einzigen HTML-Dokument bestehen. Beim ersten Seitenaufruf liefert der Webserver die JavaScript-Applikation an den Klienten aus. Die SPA kommuniziert nur dann mit dem Server, wenn Daten gelesen oder geschrieben werden, wodurch die Klient-Server-Kommunikation auf ein Minimum reduziert wird und die SPA schnell und dynamisch wirkt. Die Daten werden vom Webserver beispielswei-

[11] https://www.meteor.com/about.

[12] https://iojs.org/api/.

[13] Internet Information Service.

[14] Hypertext Markup Language.

se im JSON[15]-Format übergeben und von der JavaScript-Applikation in HTML konvertiert. (John Joseph, 2015; Mikowski & Powell, 2014)

Das Rendern der HTML-Templates übernimmt bei einer SPA der Browser. Die verstärkte klientenseitige Ausführung der Webanwendung reduziert die Serverlast. Die Webseite wird zur Laufzeit der Anwendung dynamisch um die benötigten Inhalte erweitert, was die Ladezeiten im Vergleich zum vollständigen Neuladen einer klassischen Webseite reduziert. (Mikowski & Powell, 2014)

Single-Page-Webanwendungen sind „offline-friendly“. Die vollständige Realisierung der Präsentationsschicht auf dem Klienten ermöglicht es, mithilfe der Web-Storage-Funktion einen Zwischenspeicher anzulegen. Bei Nichterreichbarkeit des Nutzdatenservers kann auf die Daten aus dem Zwischenspeicher zurückgegriffen werden. Dadurch kann die Anwendung auch ohne eine bestehende Serververbindung weiterhin betrieben werden, was einen großen Vorteil darstellt. (Mikowski & Powell, 2014)

AngularJS, Backbone.js und Ember.js sind drei der bekanntesten Frameworks zur Erstellung von SPA, die hier vorgestellt werden sollen.

2.5.3.1 AngularJS

AngularJS ist ein im Jahre 2009 von Google veröffentlichtes klientenseitiges opensource JavaScript-Framework für die Entwicklung dynamischer Webapplikationen. Es wurde entwickelt, um die Produktivität, Einfachheit und Schnelligkeit der Anwendungsentwicklung zu erhöhen. AngularJS hat darüber hinaus den Anspruch, die Entwicklung von Anwendungen zu ermöglichen, die wie native Anwendungen wirken. Aufgrund dieser Vorzüge gewinnt das AngularJS-Framework zunehmend an Popularität. AngularJS besitzt eine durchdachte Architektur, eine hervorragende Testbarkeit sowie eine hohe Leistungsfähigkeit und ermöglicht die Skalierung der Anwendung. (Tarasiewicz & Böhm, 2014)

Die Logik der Visualisierung auf der Benutzeroberfläche wird daher mittels AngularJS in Form einer SPA umgesetzt. Da AngularJS in dieser Arbeit verwendet wird, werden die Funktionalitäten von AngularJS anhand von (Tarasiewicz & Böhm, 2014) genauer beschrieben:

[15] JavaScript Object Notation.

Dependency Injection: AngularJS nutzt DI (siehe Kapitel 2.5.1.1.3), wodurch eine gute Testbarkeit, eine Reduktion des Codes und eine einfache Refaktorierung erreicht wird.

Direktiven: Durch die Verwendung von Direktiven lässt sich das HTML-Markup erweitern. Sie ermöglichen die Strukturierung und Reduktion des Codes durch dessen Kapselung in wiederverwendbaren Komponenten, beispielsweise in UI-Widgets.

MVC: AngularJS bietet das Model-View-Controller (MVC) Architekturmuster, welches die Anwendung in drei miteinander verbundene Komponenten aufteilt. Die zentrale Komponente des MVC ist das Model. Es verwaltet die Daten, die Logik und die Regeln der Anwendung. Der View ist als Benutzerschnittstelle eine beliebige ausgegebene Repräsentation von Informationen. Der Controller übersetzt Benutzereingaben in Befehle an das Model.

Dirty-Checking: Dirty-Checking ermöglicht die Beobachtung von Variablen und die Ausführung einer Aktion als Reaktion auf die Änderung der beobachteten Variablen.

Two-Way Data-Binding: Traditionell werden in der JavaScript-Programmierung Änderungen durch One-Way Data-Binding direkt im DOM von einem Event-Handler ausgeführt. Die Daten werden von einem Model bereitgestellt, während ein Template die Darstellung bestimmt. Model und Template werden zu einem View zusammengefasst, welcher an den Benutzer gesendet wird. Problematisch ist hierbei, dass Änderungen im View und im Model nicht automatisch synchronisiert werden. Beim Two-Way Data-Binding, welches AngularJS unterstützt, wird dagegen eine Verknüpfung zwischen View und Model hergestellt. Jedes Element eines Arrays wird einem DOM-Element zugeordnet. Veränderungen im Model werden an den View propagiert und dort übernommen. Dadurch entfällt die Notwendigkeit der aufwendigen manuellen und fehleranfälligen Synchronisation. Die beiden Datenbindungsmodelle sind in Abbildung 7 schematisch dargestellt. (Brink, 2015)

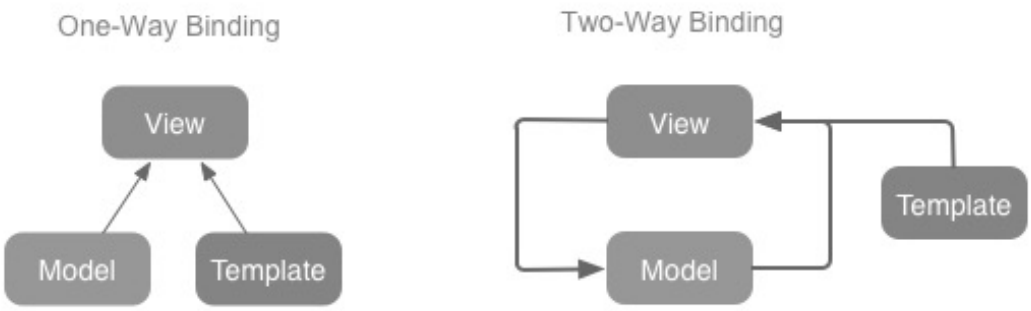

Abbildung 7: Schematische Darstellung von One-Way und Two-Way Data-Binding (Brink, 2015)

Zudem gilt es die technischen Aspekte von AngularJS, die in dieser Arbeit Anwendung finden, genauer zu beschreiben (Tarasiewicz & Böhm, 2014):

Module: Module dienen der Strukturierung einer Anwendung in AngularJS. Ein Modul kapselt zusammengehörige Anwendungskomponenten. Mögliche Anwendungskomponenten sind Controller, Services, Filter und Direktiven. Module können voneinander abhängig sein. Diese Abhängigkeiten werden mittels DI eingebunden.

Scope: Ein Scope ist ein Objekt, das den Gültigkeitsbereich innerhalb einer Ansicht beschreibt. In einem Scope können Eigenschaften und Methoden hinterlegt werden. Scopes dienen der Realisierung der Kommunikation und Logiken zwischen View und Controller.

Controller: Controller sind Konstruktionsvorschriften für Scopes. Sie definieren die für einen View benötigten Daten und die Logik. Ein Controller kann einen neuen Scope erstellen. Jeder Controller hat einen eigenen Gültigkeitsbereich.

Service: Services dienen der Kommunikation zwischen der AngularJS Anwendung und anderen Services. Die Kommunikation kann über einen implementierten REST-Service stattfinden. Ein Service kann von Controllern oder anderen Services mithilfe von DI verwendet werden.

Direktiven: Direktiven ermöglichen die Erweiterung des HTML-Vokabulars durch neue Tags und Attribute, welche im DOM definiert werden. Dieses Feature ist ein Alleinstellungsmerkmal von AngularJS. Mit Direktiven lassen sich zudem erstmals wiederverwendbare Komponenten im Web erstellen.

Model: Models sind datenhaltende Objekte und in AngularJS gewöhnliche JavaScript-Datentypen. Um die Anforderungen an ein Model in AngularJS zu erfüllen, genügen daher die üblichen Definitionen für Objekte, Arrays und primitive Datentypen.

Template: Templates sind in HTML verfasst und enthalten AngularJS-spezifische Elemente und Attribute. AngularJS verwendet die Informationen des Models und des Controllers, um einen dynamischen View im Browser zu erzeugen.

Routen: SPA realisieren mehrere logische Seiten in einem HTML-Dokument. Dies kann zu Problemen bei der Adressierung einzelner Anwendungsteile und der Vorwärts-/Rückwärts-Funktion des Browsers führen. Das Modul `ngRoute` ermöglicht zur Lösung dieses Problems die Definition von Routen. Jeder logischen Seite kann ein URL[16] zugeordnet werden, über welchen sie referenziert werden kann. Routen ermöglichen die Abbildung von URL auf Templates mit den zugehörigen Controllern.

2.5.3.2 Ember.js

Ember.js ist ein sehr umfangreiches Werkzeug zur Realisierung von SPA. Es existieren wenige Unterschiede zwischen Ember.js und AngularJS. Beide Frameworks haben die Zielsetzung, die Entwicklung von Webanwendungen mit dem „Look and Feel" nativer Anwendungen zu ermöglichen. Ember.js basiert ebenfalls auf einem MVC-Pattern und bietet ein deklaratives Two-Way Data-Binding. Es nutzt im Gegensatz zu AngularJS jedoch das Entwurfsmuster des Beobachters mit Accessors (Getter- und Setter-Funktionen) anstelle von Dirty-Checking. Ember.js wurde erst 2011 veröffentlicht und ist damit das jüngste Framework in diesem Vergleich. Die Testmöglichkeiten für AngularJS Anwendungen sind aufgrund dessen früherer Einführung ausgereifter. Die Integration von Drittanbieterbibliotheken in Ember.js gestaltet sich oftmals noch problematisch. Ember.js erfordert ferner die Verwendung von jQuery[17]. (Kelonye, 2014) Der Einstieg in AngularJS ist aufgrund der sehr guten Dokumentation leichter und die Projektstrukturierung ist flexibler gestaltbar, weshalb AngularJS in dieser Arbeit Anwendung findet.

2.5.3.3 Backbone.js

Backbone.js wurde bereits 2010 veröffentlicht und bietet einen ausgereiften, umfassenden Ansatz zur Erstellung von SPA. Es gibt wenige Einschränkungen und Konventionen zur Strukturierung des Projekts vor. Einerseits ermöglicht dies eine freie Projektgestaltung, andererseits ist die Entwicklung deshalb be-

[16] Ein Uniform Resource Locator (URL) ist eine standardisierte Adresse, über welche eine beliebige Webressource adressiert werden kann (Abts, 2015).

[17] jQuery ist eine JavaScript Bibliothek zur DOM-Navigation und -Manipulation (The jQuery Foundation, 2015).

sonders am Anfang sehr aufwendig, da zuerst die Struktur der Anwendung aufgebaut werden muss. Auch die Produktivität wird durch die Flexibilität negativ beeinflusst, da Boilerplate-Code benötigt wird. (Osmani, 2013)

Der größte Nachteil ist der Verzicht auf ein Konzept zur Aktualisierung der Darstellung in Backbone.js, weshalb die Integration einer Drittanbieterbibliothek erforderlich ist. Dies kann langfristig zu Kompatibilitäts- und Wartungsproblemen führen. (Osmani, 2013)

2.5.4 Webservices zur webbasierten Kommunikation

Zur webbasierten Kommunikation werden Protokolle verwendet. Das mit Abstand bekannteste Protokoll ist das Hypertext Transfer Protocol (HTTP). Es handelt sich um ein zustandsloses Protokoll zur Übertragung von Daten. HTTP regelt insbesondere die Kommunikation eines Webbrowsers mit einem Webserver im World Wide Web, ist jedoch nicht auf diese Funktion beschränkt. Es ist auch als allgemeines Dateiübertragungsprotokoll sehr verbreitet. (Abts, 2015)

Das HTTP ist der Anwendungsschicht im TCP/IP[18]-Schichtenmodell zugeordnet und verwendet auf der Transportschicht TCP. Das TCP/IP-Schichtenmodell ist ein Protokollstapel des Internets. TCP/IP ist ein zuverlässiges, paketvermittelndes und verbindungsorientiertes Transportprotokoll. Es ist das wichtigste Protokoll der Protokollfamilie des TCP/IP-Schichtenmodells. (Abts, 2015)

Ein Webservice ist eine Softwarekomponente zur webbasierten Kommunikation. Die Kommunikation zwischen einem Klienten und einem Webservice findet über ein Protokoll wie beispielsweise HTTP statt. Ein Webservice bietet eine oder mehrere Schnittstellen über ein Netzwerk an, über welche seine Operationen extern aufgerufen werden können. Mehrere Webservices sind durch Nachrichtenaustausch lose miteinander gekoppelt. Jeder Webservice ist über einen Uniform Resource Identifier (URI) eindeutig identifizierbar. (Abts, 2015)

Nachfolgend werden einige bekannte Technologien zur Realisierung von Webservices vorgestellt.

[18] Transmission Control Protocol/Internet Protocol.

2.5.4.1 Simple Object Access Protocol (SOAP) Webservices

Einen Webservice, der das Simple Object Access Protocol (SOAP) zur webbasierten Kommunikation nutzt, bezeichnet man als SOAP-Webservice. SOAP ist ein zustandsloses W3C-standardisiertes Netzwerkprotokoll, das den Datenaustausch zwischen Systemen und die Durchführung von Remote Procedure Calls ermöglicht. Die Datenrepräsentation erfolgt im menschen- und maschinenlesbaren XML-Format, wodurch das Protokoll programmiersprachen- und implementierungsunabhängig ist. Zur Nachrichtenübertragung nutzt SOAP Internet-Protokolle der Transport- und Anwendungsschicht. Am weitesten verbreitet ist die Nutzung von HTTP als Transportprotokoll, jedoch können SOAP-Nachrichten auch mithilfe anderer Protokolle wie FTP[19] oder SMTP[20] übertragen werden. Die Flexibilität aufgrund der Umgebungsunabhängigkeit, welche Unabhängigkeit von der Plattform, vom Betriebssystem, von der Programmiersprache und vom Transportprotokoll umfasst, ist der größte Vorteil von SOAP. (Abts, 2015)

SOAP-Nachrichten enthalten neben den zu übertragenden Informationen Verarbeitungsinformationen als Metadaten, wodurch sie größer als der reine Informationsgehalt sowie aufwändig zu analysieren und zu verfassen sind. Sowohl die Datengröße als auch das erforderliche Parsen der XML-Nachrichten beeinträchtigen die Performance und erhöhen die Datenlatenz. (Abts, 2015)

Klient und Server stehen bei SOAP in einem engen Zusammenhang. Eine Änderung am SOAP-Interface des Servers erfordert eine Anpassung des Klienten. Dies beeinträchtigt die schrittweise Entwicklung und Skalierbarkeit des Systems. SOAP erfordert zudem einen vergleichsweise hohen Implementierungsaufwand. (Kloster & Martin, 2004; Zwattendorfer, 2013)

SOAP ist vor allem für die Abbildung komplexerer Prozesse geeignet, die eine formale Schnittstellenbeschreibung erfordern und deren Bandbreite nicht limitiert ist. Es unterstützt im Gegensatz zu REST (siehe Kapitel 2.5.4.2) auch prozessorientierte und asynchrone Services mit längerer Laufzeit. (Zwattendorfer, 2013)

2.5.4.2 RESTful Webservices

Einen Webservice, der eine Representational State Transfer (REST) Architektur implementiert, bezeichnet man als RESTful Webservice. REST ist ein Architekturstil für verteilte Systeme. Es stellt eine Abstraktion der Struktur und des

[19] File Transfer Protocol.

[20] Simple Mail Transfer Protocol.

Verhaltens des World Wide Webs dar und zeichnet sich durch folgende Merkmale aus (Abts, 2015; Fielding, 2000):

Ressourcenorientierung und Adressierbarkeit: Jede Anfrage eines Klienten ist ressourcenorientiert, das heißt sie bezieht sich auf eine Ressource. Ressourcen sind Informationseinheiten, welche durch einen eindeutigen URI identifizierbar und adressierbar sind. Eine URI-Webadresse repräsentiert exakt einen Seiteninhalt und antwortet daher auf mehrfache Anfragen mit demselben URI auch mit demselben Seiteninhalt.

Repräsentation von Ressourcen: Eine Ressource kann in verschiedenen Repräsentationen vorliegen. Eine Repräsentation ist die Darstellung des Zustandes einer Ressource in einem bestimmten Datenformat wie XML oder JSON. Ein Klient kann explizit eine bestimmte Repräsentation anfordern.

Statuslose Kommunikation: Die Kommunikation erfolgt statuslos, das heißt es werden keine Cookies oder Sessions für einzelne Benutzer erstellt. Stattdessen läuft jede HTTP-Anfrage isoliert ab. Alle benötigten Informationen für die Verarbeitung sind Teil der Anfrage und werden bei jeder Anfrage erneut übermittelt. Durch den Verzicht auf einen klientenspezifischen Sitzungsstatus wird die Kopplung zwischen Klient und Server verringert und die Skalierbarkeit der Anwendung erhöht.

Ressourcenzugriff: Auf die Ressourcen können folgende durch REST definierte Standardmethoden angewendet werden:

- `GET`: Abfrage einer Repräsentation einer Ressource
- `POST`: Erstellen einer neue Ressource mit einem vom Server bestimmten URI oder Auslösung eine Verarbeitung auf dem Server
- `PUT`: Erstellen oder Ändern einer Ressource mit bekanntem URI
- `DELETE`: Löschen einer Ressource
- `HEAD`: Abfrage der Metadaten einer Ressource
- `OPTIONS`: Abfrage von Informationen über eine Ressource, beispielsweise verfügbare Repräsentationsformate

REST ist wie SOAP unabhängig von der Plattform, dem Betriebssystem und der Programmiersprache. Die Zustandslosigkeit von REST resultiert in einer hohen Skalierbarkeit. Der Implementierungsaufwand ist geringer als bei SOAP, da eine Schnittstelle durch HTTP vorgegeben ist. REST sieht jedoch ausschließlich die Verwendung von HTTP vor. SOAP ist dagegen bei der Nutzung von Transportprotokollen flexibler. Die fehlende Standardisierung stellt einen Kritikpunkt an REST dar. (Richardson & Ruby, 2007; Zwattendorfer, 2013)

Die Komponenten der Präsentationsschicht und der Geschäftslogik können in unterschiedlichen Programmiersprachen entwickelt werden. Mit REST wird die Kommunikation auch zwischen heterogenen Komponenten umgesetzt.

Während SOAP das XML-Format zwingend vorschreibt, fordert REST kein spezielles Nachrichtenformat und ist somit flexibler. Die Repräsentation kann zum Beispiel im JSON-Format erfolgen. JSON ist selbst ein gültiges JavaScript und hat im Kontext dieser Arbeit den Vorteil, dass die Ressourcen direkt vom JavaScript-basierten AngularJS im Frontend in JavaScript-Objekte überführt werden können. Der Zugriff auf die einzelnen Attribute mittels AngularJS kann daher als normaler Attributzugriff erfolgen. Die Implementierung des REST-Services erfolgt mithilfe der Java-EE-Spezifikation Java API for RESTful Web Services (Vgl. Kapitel 2.5.1.1.4).

2.5.4.3 WebSocket-Protokoll

Eine HTTP-Verbindung erfordert die klientenseitige Initiierung der Kommunikation mit dem Server, welcher die Anfrage des Klienten beantwortet. Diese Kommunikation hat den Nachteil, dass neue Informationen auf dem Server erst durch eine Anfrage des Klienten abgefragt werden müssen, um an ihn weitergeleitet werden zu können. Seit der Einführung des WebSocket-Protokolls ist jedoch eine full-duplex, also eine zeitgleiche bidirektionale Kommunikation zwischen Server und Klient möglich. Das bedeutet, dass gleichzeitig sowohl Anfragen an den Server gestellt, als auch neue Informationen unaufgefordert an den Klienten geschickt werden können. Diese Art der Kommunikation ist höchst performant. (Abts, 2015)

Das WebSocket-Netzwerkprotokoll basiert auf TCP. Die erfolgreich hergestellte TCP/IP-Verbindung zwischen Klient und Server bleibt nach der ersten Klientanfrage bestehen und ermöglicht eine ständige Kommunikation über einen definierten URL. Die erste Anfrage des Klienten wird als Handshake bezeichnet und leitet einen Protokollwechsel ein. Jeder Server definiert einen Endpoint, mit welchem sich Klienten verbinden können. (Abts, 2015)

2.5.4.4 HTML5WebSocket

HTML wurde ursprünglich zur Erstellung statischer textbasierter Dokumente im Internet entwickelt. Mit HMTL5 können Webseiten moderner und interaktiver gestaltet werden, wodurch sie wie Webanwendungen wirken und die Benutzererfahrung verbessern. (Wang, Salim, & Moskovits, 2013)

Die HTML5WebSocket-Spezifikation definiert ein API, welches die Nutzung eines WebSocket-Protokolls ermöglicht. Über das WebSocket-Protokoll kön-

nen Texte, binäre Daten oder Pongs bidirektional zwischen Klient und Server ausgetauscht werden. Das WebSocket-Protokoll definiert zwei URI Schemata: ws für einen unverschlüsselten und wss für einen verschlüsselten Datenfluss. Der URL ist von der Form ws://server[:port][/resource]. (Pilgrim, 2013)

Der Constructor eines WebSockets übernimmt die Verwaltung der WebSocket-Kommunikation in HTML5. Um eine WebSocket-Verbindung herzustellen, wird der Constructor aufgerufen, der ein WebSocket-Objekt instanziiert und zurückgibt. (Wang et al., 2013)

Das WebSocket-Protokoll und das WebSocket-API sind ereignisgesteuert. WebSocket-Events werden asynchron vom Server an den Klienten weitergeleitet, ohne dass eine Anfrage vorausgehen muss. Die Anwendung überwacht das WebSocket-Objekt mithilfe eines Event Listeners, um eintreffende Daten oder den Verbindungsstatus zu verarbeiten. Es können folgende Events auftreten (Wang et al., 2013):

open: Wenn der Server auf die WebSocket-Verbindungsanfrage antwortet, wird das Event open ausgelöst und eine Verbindung hergestellt. Über die entsprechende Rückrufmethode onopen sendet der Server eine Nachricht an den Klienten. Sobald die Verbindung hergestellt ist, ist der Handshake abgeschlossen. Der WebSocket kann nun Daten senden und empfangen.

message: WebSocket Nachrichten enthalten Daten vom Server. Das message Event wird ausgelöst, wenn Nachrichten empfangen werden. Die entsprechende Rückrufmethode heißt onmessage.

error: Ein unerwarteter Fehler löst das error Event aus. Die entsprechende Rückrufmethode heißt onerror. Fehler können eine WebSocket-Verbindung beenden.

close: Das close Event wird ausgelöst, wenn eine WebSocket-Verbindung beendet wird. Die entsprechende Rückrufmethode heißt onclose.

WebSocket-Objekte besitzen zwei Methoden, send und close. Sobald eine Verbindung zwischen Klient und Server mit WebSocket hergestellt ist, kann die Methode send aufgerufen werden, um Nachrichten vom Klienten an den Server zu senden. Nach dem Senden einer oder mehrere Nachrichten kann die Verbindung bestehen bleiben oder mit der Methode close beendet werden. (Wang et al., 2013)

2.5.5 Visualisierungstools

In diesem Unterkapitel werden Tools zur Visualisierung von Daten mit Zeit- und Ortsbezug vorgestellt.

2.5.5.1 Tools zur Erstellung von interaktiven Karten zur Geovisualisierung

Hierzu gehören Werkzeuge zur Erstellung interaktiver Karten zur Geovisualisierung von ortsbezogenen Daten, wobei die bekanntesten dieser Kategorie vorgestellt werden sollen.

OpenStreetMap

OpenStreetMap (OSM) ist ein internationales opensource Projekt mit dem Ziel, eine freie Weltkarte zu erschaffen. OSM ist frei editierbar und Mitarbeit an der Kartenerstellung ist ausdrücklich erwünscht. Durch das Mitwirken von ortskundigen Freiwilligen sind die Karten oftmals genauer als die anderer Anbietern und enthalten auch Pfade oder „Querfeldeinwege". Das Konzept von OSM bringt jedoch das Problem mit sich, dass die Karte vor allem im ländlichen Bereich noch immer lückenhaft ist. Ein weiterer großer Nachteil von OSM ist, dass Zoomen und Bildverschiebungen in der Karte zum Neuladen des Kartenbereichs führen, wodurch die Darstellung nicht flüssig ist. (Ramm & Topf, 2010)

Google Maps API

Google Maps ist ein Online-Dienst von Google Inc. zur Darstellung der geografischen Lage von Objekten mit Geoinformationen auf einer Karte. Seit 2005 steht ein öffentliches API zur Integration von Google Maps Karten auf anderen Webseiten zur Verfügung (Svennerberg, 2010). Die Nutzung ist unter bestimmten Bedingungen kostenpflichtig (Google Inc., 2015c).

Die ausschlaggebenden Gründe für die Wahl des Google Maps API sind die einfache Nutzung, die Kartenabdeckung und die weite Verbreitung von Google Maps (Dincer & Uraz, 2013).

2.5.5.2 Tools zur Diagrammerstellung

Diagramme dienen der Visualisierung von zeitbezogenen Daten. Die Erstellung von Diagrammen erfolgt mithilfe von Visualisierungstools, von denen drei hier exemplarisch vorgestellt und anhand ihrer Vor- und Nachteile gegeneinander abgewogen werden sollen.

Google Charts

Google Charts ist ein JavaScript-API von Google Inc., mit welchem Daten sehr einfach in unterschiedlichen Diagrammtypen visualisiert werden können. Google-Charts-Diagramme können mittels JavaScript unter Verwendung einer mit der ID des Diagrammobjekts in eine Webseite eingebettet werden (Google Inc., 2015a). Ein großer Nachteil ist, dass die generierten Diagramme nicht responsiv sind, weshalb sie sich nicht automatisch an die Breite eines Fensters der Seite anpasst. Responsives Webdesign wird genutzt, um Inhalte flexibel und einheitlich auf einer Webseite unabhängig vom Nutzungsgerät darzustellen (Zillgens, 2013).

D3.js

D3.js[21] ist eine sehr bekannte JavaScript-Bibliothek zur Erstellung von Diagrammen. Die Diagramme werden mithilfe von HTML, CSS und SVG[22] generiert. D3.js ist ein sehr umfangreiches und flexibles Tool zur benutzerdefinierten Diagrammerstellung. Dies ist zugleich ein Nachteil, denn es sind viele Einstellungen durch den Entwickler erforderlich, was die Diagrammerstellung arbeitsintensiv macht. Der größte Nachteil ist, dass auch bei D3.js die generierten Diagramme nicht responsiv sind. (Lerner & Powell, 2014)

Chart.js

Chart.js[23] ist eine JavaScript-Bibliothek, welche die Generierung von Diagrammen auf Basis von HTML5 Canvas[24] und CSS3[25] ermöglicht. Durch eine MIT[26]-Lizensierung besteht eine fast uneingeschränkte kostenfreie Bearbeitung und Nutzung, auch für kommerzielle Zwecke. (chart.js n.d.)

Chart.js bietet zahlreiche Vorteile. Es ist leichtgewichtig, besitzt eine hervorragende Dokumentation, ist einfach zu implementieren und besticht durch eine ansprechende konsistente Optik der zur Auswahl stehenden Diagrammarten. Die animierten Diagramme wirken dynamisch und sind zudem interaktiv. Es bietet außerdem zahlreiche benutzerdefinierte Optionen. Im Gegensatz zu den

[21] http://d3js.org.

[22] Scalable Vector Graphics.

[23] http://www.chartjs.org.

[24] HTML5 Canvas ist ein Element zur Erstellung von Grafiken mittels JavaScript-API (Wang et al., 2013).

[25] Cascading Style Sheets Level 3.

[26] Massachusetts Institute of Technology.

beiden anderen vorgestellten Tools ist es responsiv. (chart.js, 2015; Shokeen, 2015)

All diese Vorteile, insbesondere jedoch die Responsivität, die Animationen und die Interaktivität, begründen die Wahl von Chart.js zur Diagrammerstellung im Rahmen dieser Arbeit.

3 Anforderungsanalyse

Eine Anforderungsanalyse ist die Grundlage für die Entwicklung eines Implementierungskonzepts. Dabei werden sowohl die funktionalen, als auch die nicht-funktionalen Anforderungen in Hinblick auf die Fragestellung und technischen Randbedingungen ermittelt, auf deren Basis ein konzeptueller Entwurf entwickelt werden kann.

„Smart City“ bezeichnet das Konzept zur technologiebasierten Veränderung und Innovation in urbanen Räumen zur Gestaltung der Städte von morgen, um den zukünftigen Anforderungen moderner Städte entsprechend dem Prinzip der Nachhaltigkeit gerecht zu werden. Dazu ist es notwendig, die Stadt- und Verkehrsplanung effektiv zu gestalten. Effektive Maßnahmen erfordern informationsbasiertes Wissen, auf dessen Grundlage Entscheidungen getroffen werden. Das Streetlife-Projekt entwickelt in diesem Kontext urbane Mobilitätsinformationssysteme zur integrierten Mobilitätsplanung für drei Pilot-Städte. Das Streetlife-Projekt entwickelt individuelle Lösungen für jede Stadt. (Frigeri et al., 2014)

Das im Rahmen dieser Arbeit zu entwickelnde Visualisierungssystem soll als integrierter Lösungsansatz dienen und beliebige im verkehrsplanerinerischen Kontext relevante Daten zur Analyse in angemessener Weise in einer Verkehrsplanungsplattform visualisieren. Durch die Analyse dieser Daten können informationsbasierte Erkenntnisse gewonnen werden, welche zur Entscheidungsfindung in der Verkehrs- und Stadtplanung verwendet werden können.

3.1 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben die geforderten Funktionalitäten des Systems. Die Anforderungen an das Visualisierungssystem leiten sich aus den Anforderungen des Streetlife-Projekts ab (Frigeri et al., 2014).

3.1.1 Use Case für einen Systemkonfigurator

Im Rahmen des Streetlife-Projekts werden Verkehrsdaten in drei Pilotstädten erhoben, welche in separaten Systeminstanzen visualisiert werden (Frigeri et al., 2014). Das zu entwickelnde System soll die Visualisierung aller verfügbaren Daten durch eine einzige Systeminstanz ermöglichen. Aus dieser Anforderung ergibt sich die Rolle des Systemskonfigurators. Der entsprechende Use Case ist in Abbildung 8 dargestellt. Ein Systemkonfigurator kann neue Daten

in das Visualisierungssystem importieren und Systemkonfigurationen vornehmen.

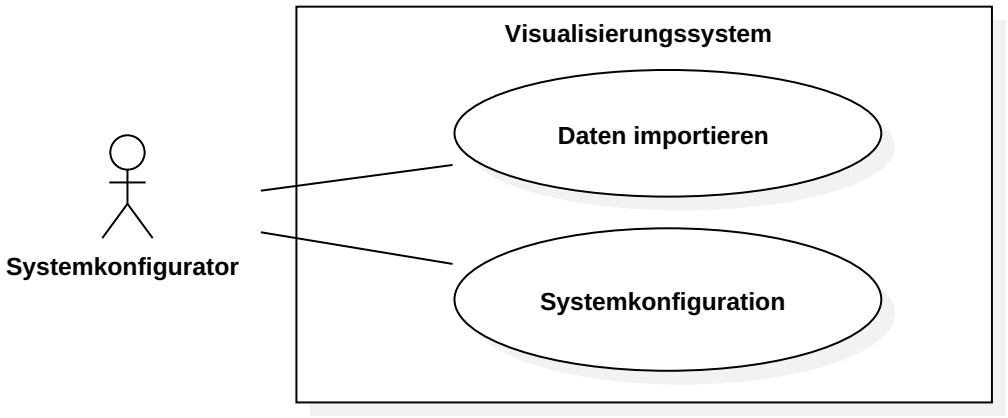

Abbildung 8: Use Case für einen Systemkonfigurator

Die Systemkonfiguration in Abbildung 9 umfasst die Entwicklung einer Komponente zur Datentransformation und spezifischer Auswertungsfunktionen, die an das Visualisierungssystem angebunden und konfiguriert werden.

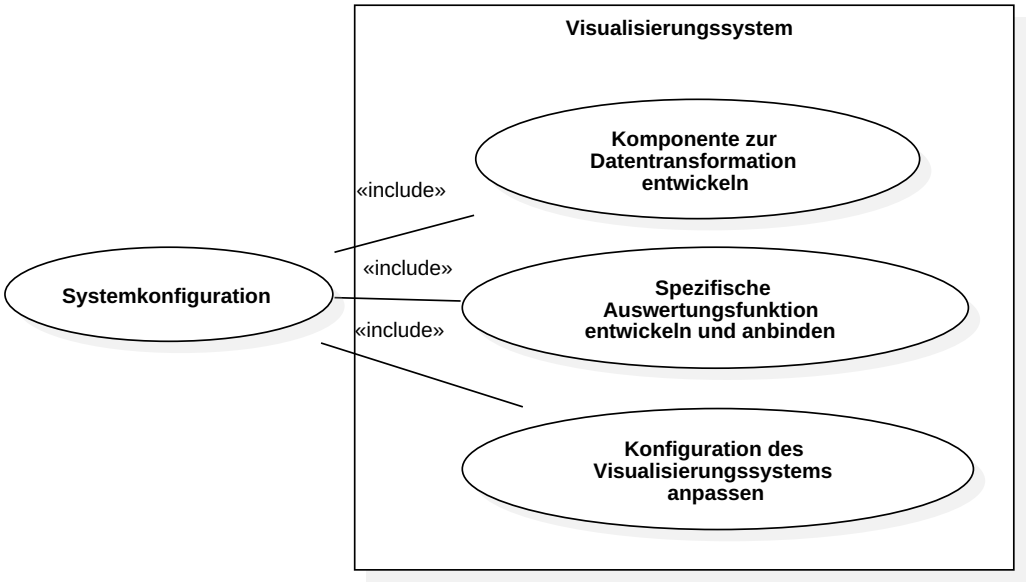

Abbildung 9: Use Case für die Systemkonfiguration

Der Datenimport in Abbildung 10 beinhaltet das Importieren, Validieren und Persistieren der Daten aus einer Datenquelle.

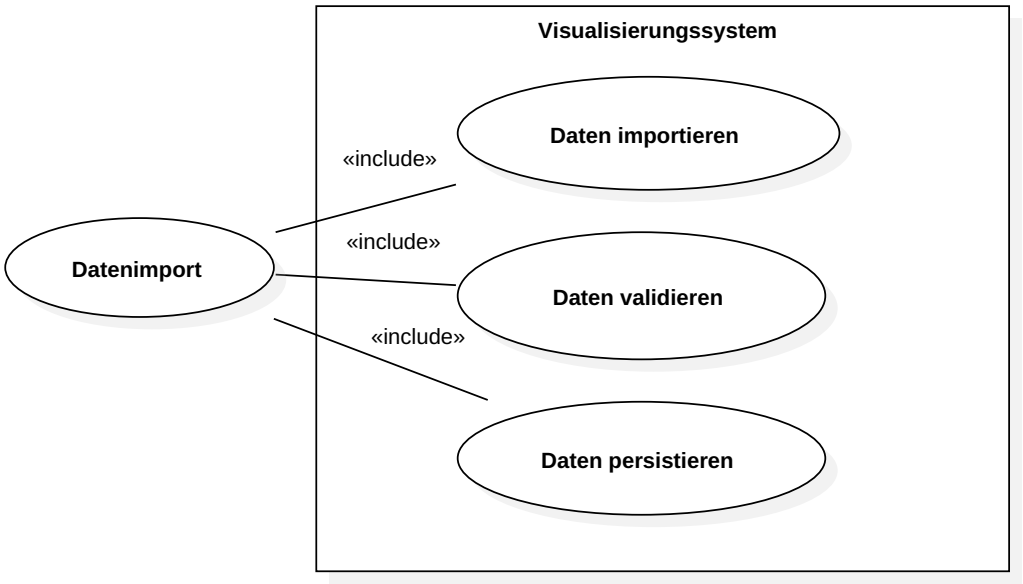

Abbildung 10: Use Case für den Datenimport

3.1.2 Use Case für einen Systemanwender

Ein Systemanwender nutzt die Funktionalitäten des Visualisierungssystems zur Visualisierung und Auswertung von Daten im Rahmen einer Datenanalyse. Diese umfasst die im Use Case in Abbildung 11 dargestellt Schritte, die anschließend detailliert beschrieben werden.

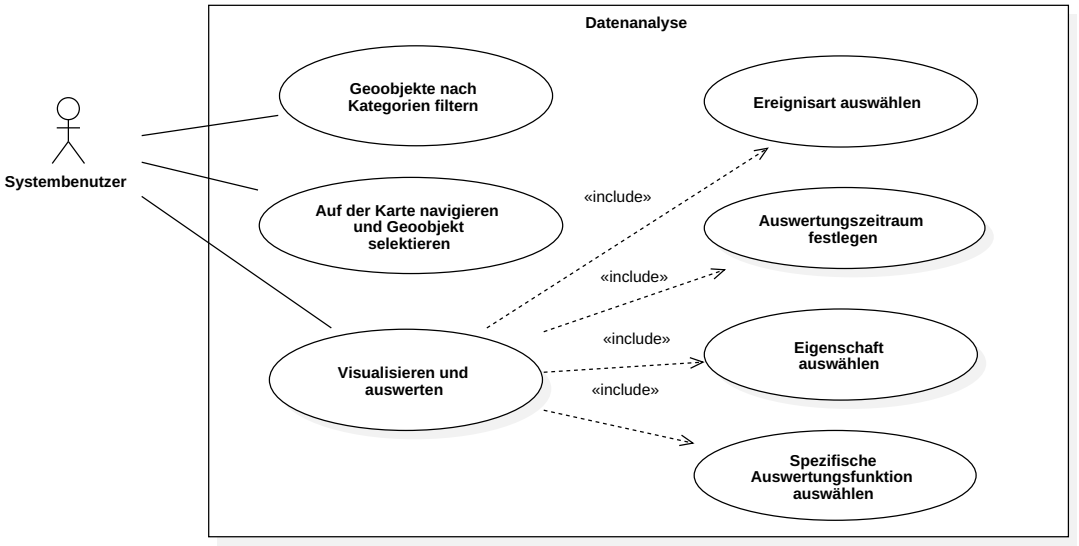

Abbildung 11: Use Case für einen Systembenutzer

Geoobjekte nach Kategorien filtern: Filter ermöglichen die Betrachtung einer Gruppe von Objekten mit einem gemeinsamen Merkmal. Zur Auswahl stehen beispielsweise Schulen und Parkplätze.

Auf der Karte navigieren und ein Geoobjekt selektieren: Die Karte wird mit dem Start des Visualisierungssystems automatisch initialisiert und auf den Standort des Systemanwenders zentriert. Der Systembenutzer kann in der Karte zoomen und navigieren. Er kann ein Geoobjekt selektieren, um dessen statische Informationen anzuzeigen und dessen dynamischen Daten im weiteren Verlauf zu analysieren.

Visualisieren und auswerten: Der Systemanwender wählt die Art des zu betrachtenden Ereignisses, zum Beispiel den Belegungsstatus eines Parkplatzes sowie eine zu analysierende Eigenschaft des Ereignisses, beispielsweise die Anzahl der freien Stellplätze, aus. Durch die Angabe eines Start- und eines Endzeitpunkts wird ein Auswertungszeitraum vom Nutzer festgelegt, um nur Ereignisse innerhalb des gewählten Zeitintervalls zu betrachten. Ein Diagramm, welches die ausgewählte dynamische Eigenschaft im angegebenen Zeitraum visualisiert, wird automatisch erzeugt und angezeigt. Der Nutzer kann die Darstellung des Diagramms anpassen. Des Weiteren werden Auswertungsfunktionen zur Auswahl gestellt. Die Ergebnisse elementarer statistischer Auswertungsfunktionen zur Bestimmung des Maximums, des Minimums und des Durchschnittswertes, welche automatisch auf die ausgewählten Daten angewendet werden, und das Ergebnis einer aus gewählten spezifischen Auswertungsfunktion werden angezeigt.

3.2 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen beschreiben die Qualitäts- und Leistungsanforderungen an ein System und legen Rahmenbedingungen fest (Hartmann, Teusch, & Wolf, 2013). Die nicht-funktionalen Anforderungen an das zu entwickelnde Visualisierungssystem ergeben sich aus dem Streetlife-Projekt und dem festegelegten Umfang der Arbeit.

Usability: Die Benutzung des Visualisierungssystems durch den Systemanwender soll über eine intuitiv zu bedienende, übersichtliche und benutzerfreundliche Oberfläche erfolgen. Das Design soll zunächst funktional ausgerichtet sein. Die gestalterischen Aspekte stehen nicht im Vordergrund.

Dokumentation: Das Visualisierungssystem soll verständlich und in einem angemessenen Umfang dokumentiert werden.

Wartbarkeit: Das Ssystem soll über eine hohe Wartbarkeit verfügen und deshalb modular aufgebaut werden.

Browserunterstützung: Die Desktopversionen der Browser Firefox 40.0, Safari 9.0 und Google Chrome 45.0 sollen unterstützt werden.

Systemreife: Das Visualisierungssystem wird als Prototyp entwickelt. Es soll lauffähig auf einem Server bereitgestellt werden, um es für Partner aus dem Streetlife-Projekt zugänglich und testbar zu machen.

Antwortzeiten und Datenlatenz: Verkehrsdaten sind hochdynamisch, weshalb eine schnelle Antwortzeit und eine minimale Datenlatenz erwünscht sind. Der Fokus der Arbeit liegt zwar nicht auf der Optimierung der dieser Aspekte, jedoch wird ein möglichst performanter Lösungsansatz gewählt, da dieser die Systemarchitektur beeinflusst. Dies erleichtert die zukünftige Optimierung.

Integration in die Streetlife-Plattform: Das Visualisierungssystem soll einfach in die Streetlife-Plattform integrierbar sein.

3.3 Analyse der Datenstruktur

Nun gilt es, die zur Verfügung gestellten Daten hinsichtlich ihrer Struktur und den enthaltenen Informationen zu analysieren. Dies ist notwendig, um die Regeln für die Transformation in eine definierte Datenstruktur festzulegen. In Anhang B wird die Transformation anhand von Beispielen gezeigt.

3.3.1 Fallbeispiel Rovereto

Der erste für diese Arbeit zur Verfügung gestellte Datensatz stammt aus dem Streetlife-Projekt. Er enthält Daten zu Parkplätzen in der italienischen Stadt Rovereto, die von der Comune di Rovereto erhoben wurden.

Die Daten liegen im JSON-Format als Dateien vor. Statische und dynamische Daten liegen dabei in getrennten Dateien vor. Nachfolgend sind die Schemata der statischen (siehe Tabelle 3) und dynamischen (siehe Tabelle 4) Daten dargestellt und beschrieben.

Tabelle 3: Schema der statischen Daten von Rovereto

Feldbezeichnung	Beschreibung
type	Typ des Objekts
objectID	Eindeutiger Index des Objekts
objectType	Typ des Objekts
objectSubtype	Subtyp des Objekts
description	Beschreibung des Objekts
location	Stützpunkt des Objekts
type	Typ des Stützpunkts
coordinates	Koordinaten des Stützpunkts
elements	Elemente des Objekts (Beschreibungen)
attribute	Eigenschaft des Elements
label	Name der Eigenschaft
value	Wert der Eigenschaft
maparea	Fläche auf der Karte
area	Fläche
type	Typ der Fläche
coordinates	Koordinaten der Fläche
color	RGB-Code der Farbe der Fläche
red	Rot-Wert der Farbe der Fläche
green	Grün-Wert der Farbe der Fläche
blue	Blau-Wert der Farbe der Fläche
alpha	Transparenz der Farbe der Fläche

In den statischen Daten werden die statischen Informationen eines Objekts aufgelistet. Das Objekt wird eindeutig indexiert und typisiert. Es handelt sich hierbei um Parkplätze. Der Subtyp spezifiziert den Objekttyp und gibt beispielsweise an, ob der Parkplatz kostenpflichtig ist. Es werden die Koordinaten des Stützpunkts angegeben. In den Elementen werden Eigenschaften definiert, die das Objekt beschreiben. Dort wird unter anderem beschrieben, wie viele Parkplätze insgesamt vorhanden sind und wie viele davon kostenpflichtig sind. Anschließend wird die Fläche definiert, welche das Geoobjekt auf der Karte abbildet. Die Fläche ist ein Polygon, dessen Form, Größe und Lage durch die Anzahl und die geografische Lage der angegebenen Koordinaten bestimmt wird. Die Datei schließt mit der Angabe eines RGB-Codes, welcher definiert, in welcher Farbe und mit welchem Transparenzgrad das Objekt auf der Karte dargestellt werden soll.

Für die Visualisierung auf der Karte ist der Abschnitt maparea relevant. Er enthält die Koordinaten, welche die Lage und Geometrie des Objekts auf der Kar-

te definieren sowie die Farbinformation für die Darstellung. Die Beschreibungen im Abschnitt elements werden als statische Information angezeigt, wenn das Objekt in der Karte angewählt wird.

Tabelle 4: Schema der dynamischen Daten von Rovereto

Feldbezeichnung	Beschreibung
_id: {$oid}	Eindeutiger Index
_class	Name der Klasse
author	Name des Autors
time	Zeitstempel des Erstellungszeitpunkts
value	Eigenschaften der dynamischen Informationen
_id	Inhaltsindex
slotsFree	Anzahl der kostenlosen Parkplätze
slotsOccupiedOnFree	Anzahl der kostenlosen freien Parkplätze
slotsUnavailable	Anzahl der belegten Parkplätze
slotsPaying	Anzahl der kostenpflichtigen Parkplätze
slotsOccupiedOnPaying	Anzahl der belegten kostenpflichtigen Parkplätze
slotsTimed	Anzahl der zeitlich begrenzten Parkplätze
slotsOccupiedOnTimed	Anzahl der belegten zeitlich begrenzten Parkplätze
polyline	Stützpunkt der Geoinformation
areaId	Index des Gebiets
agency	Name der Agentur
position	Geografische Koordinaten des Objekts
name	Refenz auf das zugehörige statische Objekt
updateTime: {$numberLong}	Zeitpunkt der Aktualisierung
version: {$numberLong}	Nummer der Version

Die dynamischen Daten werden zunächst eindeutig indexiert und klassifiziert. Der Autor der Daten und der Erstellungszeitpunkt werden vermerkt. Der Erstellungszeitpunkt enthält den Zeitbezug der Daten. Es folgt die Angabe von Eigenschaften der dynamischen Informationen, zum Beispiel die Anzahl der freien Stellplätze auf einem Parkplatz.

Dynamischen Daten bilden den Zustand von dynamischen Eigenschaften eines Geoobjekts zu einem Beobachtungszeitpunkt ab. Für die Visualisierung und Auswertung sind der Erstellungszeitpunkt, die Eigenschaften der dynamischen Informationen, welche die Anzahl freier und belegter Stellplätze angeben, und die Referenz auf das zugehörige statische Objekt relevant.

3.3.2 Fallbeispiel goBerlin

Der zweite zur Verfügung gestellte Datensatz stammt aus der Webanwendung Schullotse im Rahmen des Projekts goBerlin. Sie stellt vergleichende Informationen über Berliner Schulen auf Basis statistischer Auswertungen zur Verfügung.

Die statischen und dynamischen Daten des goBerlin-Projekts liegen getrennt voneinander im XML-Format als Dateien vor. Tabelle 5 beschreibt das Schema der statistischen, Tabelle 6 das der dynamischen Daten.

Tabelle 5: Schema der statischen Daten von goBerlin

Feldbezeichnung	Beschreibung
schulId	Eindeutiger Index des Objekts
name	Name der Schule
schulart	Schulart
adresse	Adresse der Schule
homepage	Homepage der Schule
telefon	Telefonnummer der Schule
ortsteil	Ortsteil der Schule
plz	Postleitzahl der Schule

Die statischen Daten enthalten die statischen Informationen der Schule, wie deren Namen und die Schulart. Die Daten enthalten jedoch keine Koordinaten, sondern nur die Adresse der Schule. Die dazugehörigen Koordinaten müssen vor der Darstellung auf der Karte bestimmt werden.

Tabelle 6: Schema der dynamischen Daten von goBerlin

Feldbezeichnung	Beschreibung
schulId	Eindeutiger Index und Referenz auf das zugehörige statische Objekt
schuljahr	Schuljahr
jahrgangsstufe	Jahrgangsstufe
anzahlSchueler	Anzahl der Schüler
anzahlSchuelerinnen	Anzahl der Schülerinnen
gesamtAnzahl	Gesamtzahl der Schüler/-innen

Die dynamischen Daten enthalten die zu einem statischen Objekt gehörigen dynamischen Eigenschaften. Die Zuordnung erfolgt über den eindeutigen Index. Die Daten bilden die Anzahl der Schüler und Schülerinnen einer Jahrgangsstufe in einem bestimmten Schuljahr ab. Sie enthalten keinen definierten Zeitpunkt. Es wird daher ein Zeitpunkt im angegebenen Schuljahr gewählt, um einen Zeitstempel zu erstellen und die Daten zeitlich filtern zu können. Die Anzahl der Schüler/-innen sind die relevanten dynamischen Informationen, die visualisiert und ausgewertet werden sollen.

4 Konzeption des Visualisierungssystems

Dieses Kapitel beschreibt die Konzeption des Visualisierungssystems und diskutiert das entwickelte Konzept eines generischen Datenmodells für orts- und zeitbezogene Daten, welches zentraler Bestandteil des Visualisierungssystems ist.

4.1 Funktionalitäten des Visualisierungssystems

Im Folgenden werden die Funktionalitäten des Visualisierungssystems konzipiert.

4.1.1 Datentransformation

Das Visualisierungssystem soll datensatzübergreifende Analysen ermöglichen. Das bedeutet, dass alle Datensätze unabhängig von ihrer Struktur, ihrer Zugriffstechnologie oder ihres Formats auswertbar sein sollen. Für die zur Verfügung gestellten Daten bedeutet dies, dass sowohl die Schulen aus dem go-Berlin-Projekt als auch die Parkplätze des Rovereto-Projekts ohne Anpassungen am Analyse- und Darstellungsteil und ohne eine vorherige Auswahl eines zu betrachtenden Datensatzes auswertbar sein sollen. Für die Erfüllung dieser Anforderung ist ein generisches Datenmodell nötig, in welches die Daten überführt werden. Dieser Vorgang wird als Transformation bezeichnet und ermöglicht einen einheitlichen Umgang mit den Daten.

Ein Nachteil von Datentransformationen ist im Allgemeinen der Verlust von Informationen, welche nicht im neuen Schema abgebildet werden können, was wiederum eine verminderte die Datenqualität zur Folge hat. Es soll daher ein möglichst verlustfreier Transformationsmechanismus entwickelt werden. Auf diesen Punkt wird in der Beschreibung des konzipierten Datenmodells in Kapitel 4.2.1 genauer eingegangen.

Für jeden Datensatz mit einer eigenen Datenstruktur muss ein eigener Transformationsmechanismus implementiert werden. Die Festlegung der Transformationsregeln erfordert eine manuelle Analyse der Daten, weshalb kein universeller Transformationsmechanismus implementierbar ist.

Um die Abhängigkeit des Visualisierungssystems zur Implementierung neuer Transformationsmechanismen vom Entwickler zu reduzieren, soll die Transformation dezentralisiert werden. Dies wird durch den Einsatz spezifischer Transformer ermöglicht. Jeder Transformer ist für die Extraktion der Daten, für die er konzipiert wurde, aus der entsprechenden Datenquelle und deren

Transformation in das einheitliche Datenformat verantwortlich. Die Transformer sind voneinander unabhängig und können entweder fest in das Visualisierungssystem integriert werden oder dezentral in externen Systemen integriert sein. Ein Datenbereitsteller, der seine Datensätze für das Visualisierungssystem lokal in seiner Umgebung zur Verfügung stellen möchte, kann seine Daten mit einem spezifischen externen Transformer in seinem eigenen System in das einheitliche Datenformat transformieren und anschließend über ein vordefiniertes, generisches API zur Verfügung stellen. Liegen die Daten bereits in der Umgebung des Visualisierungssystems vor, so kann der Transformer auch als interne Teilkomponente mit einem ebenfalls vorgegebenen, generischen API in das Visualisierungssystem integriert werden. Dadurch werden die spezifischen Probleme eines verteilten Lösungsumsatzes umgangen.

Innerhalb dieser Arbeit werden exemplarisch sowohl ein interner Transformer für die goBerlin-Daten, als auch ein externer Transformer für die Rovereto-Daten entwickelt.

4.1.2 Laden und Speichern der Daten

Um die Daten visualisieren und auswerten zu können, müssen sie zunächst in das Visualisierungssystem geladen werden. Daten können im Allgemeinen entweder zur Laufzeit eines Systems bereitgestellt oder dauerhaft persistiert werden. Die Bereitstellung zur Laufzeit würde bei jedem Start des Visualisierungssystems die erneute Extraktion und Transformation der Daten erfordern. Durch die Persistierung stehen die transformierten Daten jederzeit zur Visualisierung bereit, weshalb eine persistente Lösung gewählt wird.

Die Persistierung der Daten erfordert die Implementierung eines Validierungsmechanismus. Bei der Validierung werden neue transformierte Daten mit bereits persistierten Daten in der Datenbank abgeglichen, um Duplikate zu vermeiden und Aktualisierungen zu übertragen.

Als Grundlage zur Realisierung des Visualisierungssystems soll das Modell des Data Warehouse System dienen. Der Import und die Persistierung heterogener Daten aus unterschiedlichen Quellen sind zentrale Funktionen eines DWS, welches daher ein sinnvoller Startpunkt für den Entwurf des Visualisierungssystems ist.

4.1.3 Visualisierung und Auswertung der Daten

Das Visualisierungssystem soll die Visualisierung und Auswertung der persistierten Daten ermöglichen. Mithilfe eines Visualisierungsmechanismus werden die persistierten Daten generisch auf einer Benutzeroberfläche visualisiert. Konzeptionell sollen die funktionalen Anforderungen wie folgt erfüllt werden:

(1) Die statischen Daten enthalten Informationen mit geografischem Bezug, anhand derer sie als Geoobjekte durch eine Geometrie auf einer Karte verortet dargestellt werden. Sie werden nach Merkmalen gruppiert, um die Anwendung von Filtern zu ermöglichen. Die statischen Informationen zur genaueren Beschreibung eines Geoobjekts können angezeigt werden, sind jedoch nicht weiter auswertbar.

(2) Dynamische Daten referenzieren auf statische Objekte. Sie enthalten sich im zeitlichen Verlauf verändernde Attribute, welche für die Verkehrsplanung wichtige Aspekte beschreiben und daher für die Analysen eines Verkehrsplaners relevant sind. Die dynamischen Daten eines ausgewählten statischen Objekts sollen in einem Diagramm visualisiert werden, damit die Informationen über die zeitliche Veränderung des betrachteten Attributs schnell visuell erfasst werden können. Bei der Visualisierung ist das Skalenniveau zu beachten, um einen geeigneten Diagrammtyp für die Darstellung zu verwenden. Die dynamischen Daten besitzen einen Zeitbezug und sollen nach Zeiträumen gefiltert werden können.

(3) Zur Datenanalyse sollen elementare und spezifische Auswertungsfunktionen angeboten werden. Im Rahmen dieser Arbeit werden elementare Auswertungsfunktionen zur Bestimmung des Maximums, des Minimums und des Durchschnittswerts fest implementiert. Die Analysemöglichkeiten sollen mittels eines Plug-In-Mechanismus um spezifische Auswertungsfunktionen erweiterbar sein, welcher exemplarisch umgesetzt wird. Das Skalenniveau beschränkt die Anwendbarkeit von Funktionen, weshalb nur anwendbare spezifische Auswertungsfunktionen zur Auswahl gestellt werden sollen. Die Ergebnisse der Auswertungsfunktionen werden dem Anwender auf einer übersichtlichen und benutzerfreundlichen Oberfläche angezeigt.

4.2 Datenmodell

Das Datenmodell spielt eine Schlüsselrolle in dieser Arbeit. Es ist das zentrale Konzept, das die übergreifende Nutzung heterogener Datensätze ermöglicht, da es die Grundlage der Datentransformation bildet. In diesem Abschnitt wird der Entwicklungsprozess des Datenmodells dokumentiert und der Aufbau des Modells erläutert. Dabei wird besonderer Wert darauf gelegt, den Geo- und Zeitbezug der betrachteten Daten angemessen ausdrücken zu können und die Werte der veränderbaren Datenattribute möglichst verlustfrei repräsentieren zu können.

4.2.1 Konzept der verlustfreien Datentransformation

Das generische Datenmodell des Visualisierungssystems soll eine möglichst verlustfreie Transformation ermöglichen. Die Transformation der für die Analyse und Visualisierung irrelevanten Attribute kann bewusst entfallen.

Daten enthalten Informationen in Form von Attributen. Ein einfaches Attribut besteht aus einer Attributsbeschreibung (Key) und einem Attributswert (Value), welche ein Key-Value-Pair (KVP) bilden. Komplexe Attribute können aus einfachen Attributen oder wiederum aus komplexen Attributen bestehen. Das Konzept des KVP dient als Basis des verlustfreien Transformationmechanismus. Dabei wird ein einfaches Attribut in ein komplexes Attribut transformiert. Ein KVP bildet den Key des zu transformierenden Attributs ab, ein weiteres KVP dessen Value. Das komplexe Attribut kann mit weiteren Attributen ergänzt werden, welche Metainformationen abbilden. Dieser Transformationsmechanismus ermöglicht eine beliebig lange und tiefe Datenstruktur, in welcher alle Attribute der zu transformierenden Daten verlustfrei abgebildet werden können.

Um die Homogenität der einzelnen Attribute zu gewährleisten wird diesen einheitlich der Typ „String" zugewiesen. Der Datentyp „String" bietet im Gegensatz zu anderen primitiven Datentypen die Möglichkeit, aus jedem beliebigen anderen primitiven Datentyp ohne Informationsverlust eine Zeichenkette zu erzeugen. Im KVP werden daher sowohl der Key als auch der Value als Zeichenkette repräsentiert (serialisiert). Dies ermöglicht die Transformation aller Attribute in ein fest definiertes generisches Datenmodell. Durch geeignete Serialisierung und Deserialisierung der Attribute wird die verlustfreie Transformation aller Attribute ermöglicht.

Abbildung 12 zeigt an einem Beispiel die Transformation einer statischen Information in Form eines Attributs. Das Attribut der statischen Information

„parkplatzName: ‘Osloer A1’“ wird als Objekt transformiert. Das Objekt enthält das KVP und eine Metainformation zur Beschreibung des Datentyps des Attributwerts (dataType). Metainformationen sind als Konstanten in Enumerationen definiert. Der Key ist „parkplatzName”, der Value ist „Osloer A1” und vom Datentyp String, da es sich um eine Zeichenkette handelt.

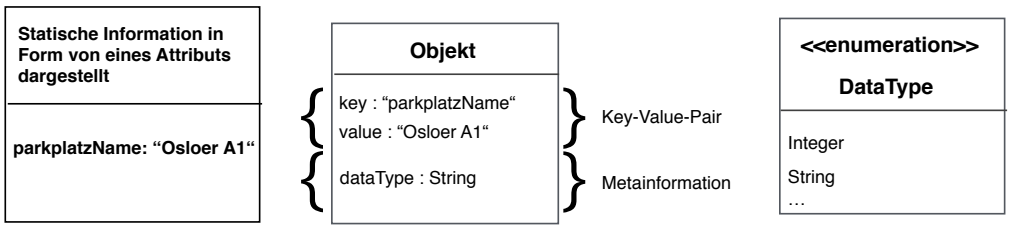

Abbildung 12: Beispielhafte Transformation einer statischen Information in Form eines Attributs

Abbildung 13 stellt die analoge Transformation von dynamischen Informationen in Form von Attributen in Objekte. Das erste Attribut der einer dynamischen Information ist „status: ‘out of order’“. Der Key ist „status”, der Value ist „out of order”. Das KVP wird zusätzlich zum Datentyp um die Angabe des Skalenniveaus (scaleType) ergänzt. Der Zeichenkette „out of order” werden der Datentyp „String“ und das Skalenniveau „nominal“ zugewiesen. Das zweite Attribut der dynamischen Information ist „free: 12”. Der Key ist „free” und vom Typ „String“. Der zugehörige Value ist „12” und wird entsprechend des Transformationsmechanismus ebenfalls in eine Zeichenkette transformiert. Die metriche Zahl „12” ist vom Datentyp „Integer“ und ratioskaliert.

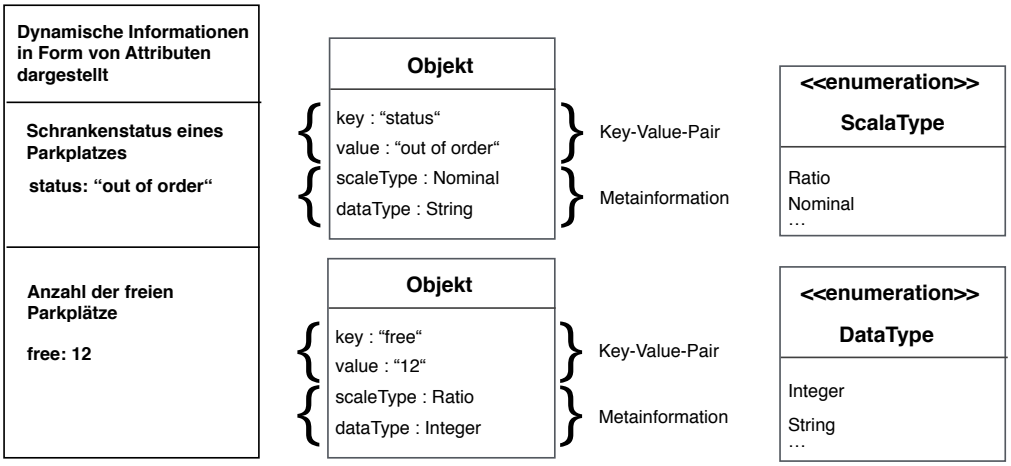

Abbildung 13: Beispielhafte Transformation dynamischer Informationen in Form von Attributen

4.2.2 Logische Bestandteile des Datenmodells

In diesem Unterkapitel werden Vorüberlegungen zu der Beziehung zwischen statischen und dynamischen Informationen angestellt, anhand derer unter Berücksichtigung des Konzepts zur verlustfreien Transformation ein Datenmodell abgeleitet wird.

Es werden statische Objekte in der realen Welt zu unterschiedlichen Zeitpunkten betrachtet. Die statischen Objekte enthalten einen Geobezug und sind mit dynamischen Informationen verknüpft, die Zustandsinformationen zu bestimmten Zeitpunkten repräsentieren. Ein statisches Objekt kann mit einer oder mehreren dynamischen Informationen verknüpft sein oder keine dynamischen Informationen aufweisen, falls noch kein Ereignis beobachtet wurde und folglich keine zeitbezogenen Zustandsinformationen vorliegen. Eine dynamische Information existiert nur in Verbindung mit einem statischen Objekt als Bezugspunkt.

Basierend auf diesen Vorüberlegungen lassen sich die wesentlichen Aspekte ableiten, die das Datenmodell beinhalten muss. Aus Gründen der Übersichtlichkeit werden sie durch Teilpakete repräsentiert:

(1) Das Teilpaket `MapObject root` beinhaltet die statischen Informationen eines Objekts welche sich aus der statischen Objektbeschreibung und der Art des Objekts zusammensetzen.
(2) Das Teilpaket `Simple Feature Geometry` enthält das Geometrieschema zur Darstellung des Geobezugs und der Geometrie des statischen Objekts.
(3) Das dritte Teilpaket `Event root` umfasst die dynamischen Daten, die auf ein statisches Objekt referenzieren und Beschreibungen von Ereignissen sowie deren Art beinhalten.
(4) Ein weiteres Teilpaket, `Description root`, dient der Abbildungen der Attribute eines Objets.
(5) Die Enumerationen der Teilpakete sind im Teilpaket `Enumeration root` zusammengefasst.

In Abbildung 14 sind die Teilpakete des Datenmodells und ihre Beziehungen in Form eines Klassendiagramms dargestellt. Die beiden Teilpakete `MapObject root` und `Simple Feature Geometry` stehen in einer Eins-zu-eins-Beziehung. Zwischen den Teilpaketen `Event root` und `MapObject root` besteht eine Eins-zu-n-Beziehung. `Description root` ist mit den Teilpaketen `MapObject root` und `Event root` assoziiert. `Enumeration root` steht in keiner direkten Beziehung zu den anderen Paketen. Die einzelnen Teilpakete werden im Entwurf konkretisiert.

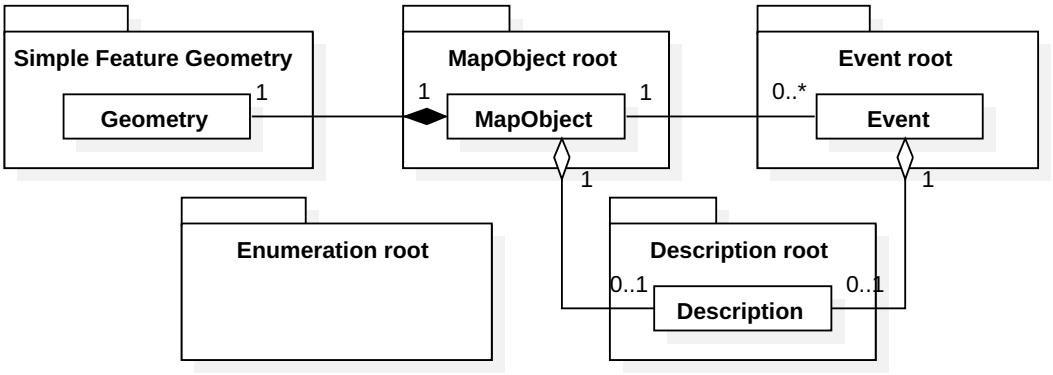

Abbildung 14: Klassendiagramm der Teilpakete des Datenmodells

5 Entwurf des Visualisierungssystems

Das Ziel des Entwurfs ist die Konkretisierung der Konzeption unter Berücksichtigung der Anforderungen. Der Entwurf bildet im weiteren Verlauf der Arbeit die Grundlage für die Entwicklung des Visualisierungssystems.

Zunächst wird das im Kapitel 4 eingeführte Datenmodell konkretisiert. Anschließend wird die Grobarchikektur des Visualisierungssystems beschrieben. Ferner werden die getroffenen Designentscheidungen beschrieben. Abschließend werden die Komponenten des Visualisierungssystems und deren Subkomponenten spezifiziert.

5.1 Konkretisierung des Datenmodells

5.1.1 Statische Objekte

Die statischen Objekte werden durch die Klasse `MapObject` im Teilpaket `MapObject root` repräsentiert. Ein `MapObject` besitzt die Attribute ID, Type und Color. Die ID ist ein eindeutiger Index. Das Attribut Type ist ein vordefinierter Aufzählungstyp der Enumeration `MapObjectType`. Das Attribut Color vom Typ „String“ wird als hexadezimaler Farbencode angegeben und definiert die Farbe, mit welcher es auf der Karte dargestellt wird.

Die Klasse `MapObject` hat eine Eins-zu-n-Beziehung zur Klasse `Description` des Teilpakets `Description root`, welches die Transformation der Attribute eines statischen Objekts ermöglicht. Sie steht in einer kompositorischen Eins-zu-eins-Assoziation zur Oberklasse `Geometry` des Teilpakets `Simple Feature Geometry`, da die Geometrie ein Teil des statischen `MapObjects` ist. Ferner steht sie in einer bidirektionalen Beziehung zur Klasse `Event` des Teilpakets `Event root`. Zur Veranschaulichung der Beziehungen der Klasse `MapObject` ist ein vereinfachtes Klassendiagramm in Abbildung 15 dargestellt. Ein ausführliches UML[27]-Klassendiagramm des Datenmodells ist in Abbildung 23 in Anhang A.

[27] Unified Modeling Language.

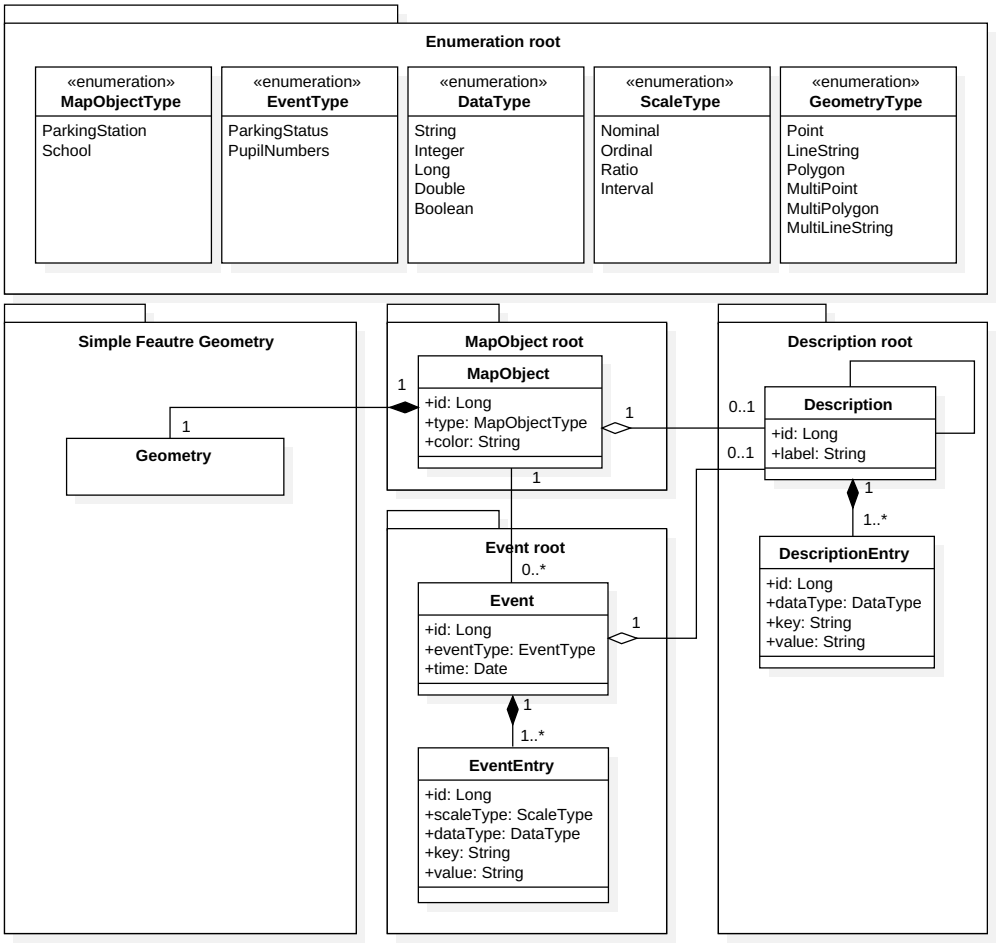

Abbildung 15: Klassendiagramm der Beziehungen der Klassen der Teilpakete

Objekte werden in Descriptions transformiert. Sie können Attribute oder Unterobjekte enthalten. Unterobjekte werden in Descriptions innerhalb der Description des Objekts transformiert. Attribute, die statische Informationen repräsentieren, werden in DescriptionEntries innerhalb der Descriptions transformiert und durch Metainformationen ergänzt. Durch dieses Vorgehen entsteht eine unbegrenzt verschachtelte Struktur, welche die Hierarchie der Objekte, Unterobjekte und Attribute der ursprünglichen Daten abbildet.

5.1.2 Dynamische Objekte

Das Teilpaket Event root ist für die Transformation von dynamischen Informationen unter Zuhilfenahme des Teilpakets Description root zuständig. Dynamische Informationen enthalten Events. Sie geben den Zustand von dynamischen Eigenschaften eines statischen Objekts zu einem Beobachtungszeitpunkt an. Die Beobachtung erfolgt in beliebigen Zeitintervallen.

Ein Event besitzt eine eindeutige ID, einen EventType, welcher in der entsprechenden Enumeration definiert sein muss, und einen Zeitstempel. Es enthält EventEntries, die Attribute abbilden, welche eine dynamische Information repräsentieren.

Die Beziehungen des Teilpakets sind Abbildung 15 zu entnehmen. Durch die Beziehung zwischen der Klasse Event und der Klasse Description können auch Attribute transformiert werden, welche statische Informationen enthalten, wie den Namen des Autors der Daten. Dieses Attribut wird in einen DescriptionEntry transformiert. Die Klasse Event steht ferner in einer bidirektionalen Beziehung zur Klasse MapObject. Diese Assoziation erlaubt den Aufruf der existierenden Events für ein MapObject oder die Identifikation eines zu einem Event gehörenden MapObjects. Ein Event referenziert auf genau ein MapObject, während ein MapObject n Events besitzen kann.

5.1.3 Geometrieschema

Die Geometrie eines MapObjects wird mithilfe des Teilpakets Simple Feature Geometry abgebildet (siehe Abbildung 23). Das Teilpaket basiert auf dem Simple Feature Modell, welches in Kapitel 2.3.3.2 vorgestellt wurde. Das Geometrieschema des Simple Feature Modells wird in dieser Arbeit als Grundlage für die Transformation und Visualisierung von Geoinformationen verwendet.

Die Klasse Geometry ist eine abstrakte Oberklasse für verschiedenartige Geometrien, welche durch Unterklassen repräsentiert sind. Die gemeinsamen Eigenschaften und Methoden der Geometrieobjekte werden in der Oberklasse Geometry zusammengefasst.

Eine Geometry besitzt einen automatisch generierten eindeutigen Index und Eigenschaften, welche in der Klasse Properties festgelegt sind. Die Eigenschaften beinhalten den GeometryType, welcher in der entsprechenden Enumeration definiert sein muss. Der GeometryType ermöglicht die Konkretisierung des Geometrieobjekts. Die Klasse Properties steht in einer Eins-zu-n-Beziehung mit der Klasse CoordinatesEntry. Sie enthält die Koordinaten eines Punktes auf der Erdoberfläche, bestehend aus Längen- und Breitengrad vom Datentyp double. Mithilfe der Methode addCoordinates(lat: double, long: double) können neue Koordinateneinträge mit den als Parameter angegebenen Längen- und Breitengraden erstellt werden. Der Aufruf der Methode erzeugt eine neue Eigenschaft mit einem CoordinatesEntry, welcher den angegebenen Längen- und Breitengrad enthält.

Die Klasse Point ist eine Unterklasse der Oberklasse Geometry. Sie beschreibt eine Punktgeometrie und deren Lage in einem Koordinatensystem. Die Klasse Polygon ist eine Unterklasse der Klasse Surface, welche wiederum eine Unterklasse der Klasse Geometry ist. Sie beschreibt die Lage und Form eines polygonen Flächenobjekts in einem Koordinatensystem. Die Klasse Polygon hat eine indirekte Beziehung zur Klasse Point und kann dadurch die Koordinateneinträge der Klasse Point nutzen. Ein Polygon wird durch Menge von Punkten beziehungsweise Koordinaten beschrieben.

Die objektorientierte Methodik des erweiterten Geometriemodells ermöglicht die einfache Verwendung der Geoinformation bei der Visualisierung.

5.1.4 Enumerationen

Im Teilpaket Enumeration root sind alle Enumerationen des Datenmodells zusammengefasst. In den Enumerationen müssen alle Aufzählungswerte, welche bei der Transformation verwendet werden sollen, definiert sein. Die Enumeration GeometryType ist durch die ISO-Normen 19125-1 und 19125-2 vollständig definiert (Vgl. Kapitel 2.3.3.2). Die Enumeration ScaleType ist durch die *per definitionem* existierenden Skalenniveaus ebenfalls vollständig definiert (Vgl. Kapitel 2.1.3). Da eine endliche Anzahl verschiedener Datentypen existiert, kann die Enumeration DataType umfassend definiert werden. Die benötigten Konstanten der Enumerationen MapObjectType und EventType sind von den verwendeten Daten abhängig. Diese Enumerationen können nicht vollständig definiert werden. Es wird daher nötig sein, sie in Zukunft um neue Einträge zu ergänzen, um neue Datensätze transformieren und im Visualisierungssystem nutzen zu können. Der Aufwand ist jedoch gering.

5.2 Grobarchitektur

Die Architektur des Visualisierungssystems soll in drei Schichten realisiert werden. Die Drei-Schichten-Architektur ist durch das in dieser Arbeit verwendete Java-EE-Framework vorgegeben. Sie besteht aus einer Datenhaltungs-, einer Fachkonzept- und einer Präsentationsschicht. Die Schichten greifen jeweils über ein API auf die hierarchisch niedrigere Schicht zu. (Weber, 2012)

Die Funktionen und Logiken des Systems werden in Komponenten gekapselt und durch die Schichtenarchitektur voneinander getrennt. Dies verbessert das Verständnis der komplexen Systemstruktur, erhöht die Wartbarkeit des Systems und ermöglicht den Austausch sowie die Wiederverwendung der einzelnen Komponenten.

Das allgemeine Komponentendiagramm der Drei-Schichten-Architektur des Visualisierungssystems ist in Abbildung 16 dargestellt. Das Frontend entspricht der Präsentationsschicht. Die Geschäftslogik wird dem Backend zugeordnet. *Specific Transformer* sind Bestandteil der Geschäftslogik, können jedoch in externe Systeme ausgelagert werden. Die Datenhaltungsschicht enthält die MySQL-Datenbank.

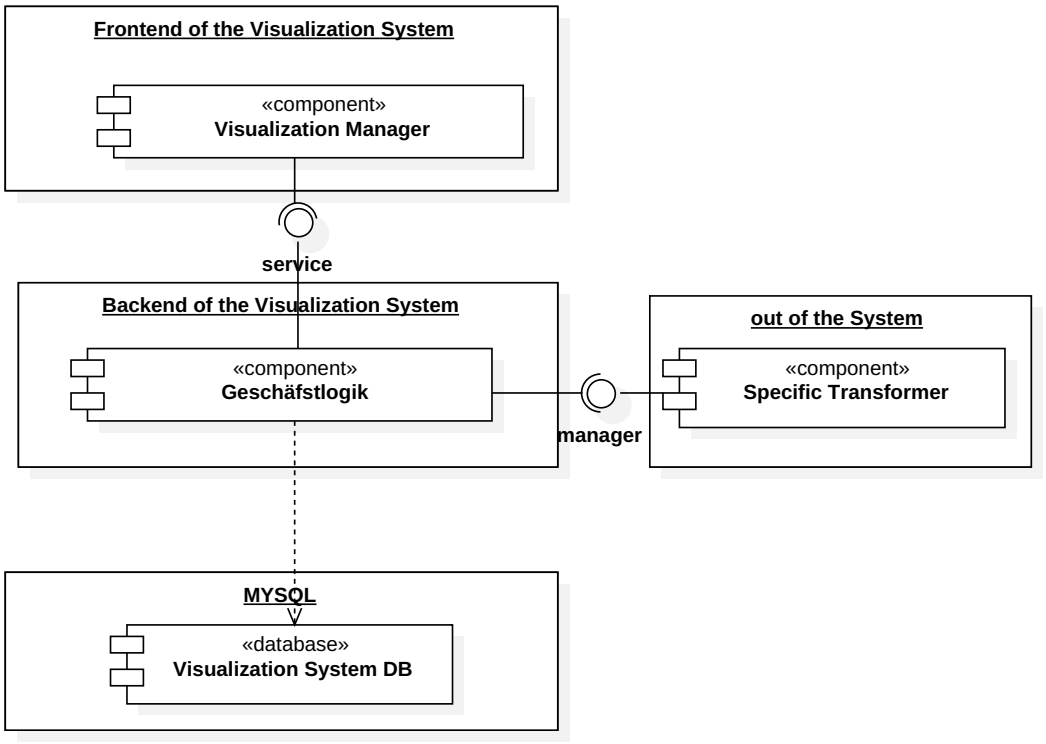

Abbildung 16: Allgemeines Komponentendiagramm der Drei-Schichten-Architektur des Visualisierungssystems

Die Persistenzschicht, die unterste Ebene des Visualisierungssystems, enthält die Datenbank, in welcher Objekte persistiert werden. Die Persistierung wird von der Geschäftslogik angestoßen. Sie enthält eine Datenzugriffskomponente, mit deren Hilfe sie mit der Persistenzschicht kommuniziert. Die Datenzugriffskomponente enthält alle benötigten Funktionalitäten für die Persistenz der Objekte. Mithilfe der sogenannten CRUD-Operationen `Create`, `Read`, `Update` und `Delete` ermöglicht sie die Speicherung und Verwaltung der fachlichen Objekte in der relationalen Datenbank. Durch Object-Relational Mapping (ORM) bildet sie die Objekte auf das relationale Datenbankschema ab. Alle Klassen des in der Konzeption entworfenen Klassendiagramms werden persistiert und als Entitätsklassen abgebildet. Das Datenmodell definiert Beziehungs- und Vererbungshierarchien in einer Tabellenstruktur und nutzt die Datentypen des relationalen Datenbankmanagementsystems.

Das Backend enthält die Komponenten der Geschäftslogik, in welchen die Mechanismen des Visualisierungssystems zum Importieren, Transformieren, Validieren, Aufbereiten und Auswerten der Daten umgesetzt werden. Die in ihnen enthaltenen Geschäftsprozessklassen besitzen Methoden, mit deren Hilfe die Geschäftslogik implementiert wird.

Die Präsentationsschicht enthält die Komponente *Visualization Manager*. Über eine Fachkonzeptzugriffskomponente greift sie auf die Geschäftslogik zu. Der *Visualization Manager* dient der Visualisierung der transformierten Daten, welche von der Geschäftslogik aus der Persistenzschicht zur Verfügung gestellt werden. Eine weitere Aufgabe des *Visualization Managers* ist die Dialogkontrolle. Der Dialogkontroller reagiert auf Eingaben des Benutzers, indem er die entsprechenden Prozesse in der Geschäftslogik anstößt.

5.3 Designentscheidungen der funktionalen Komponenten

Anhand der Anforderungen an das Visualisierungssystem und der Konzeption werden Designentscheidungen bezüglich des Systementwurfs getroffen. Diese betreffen die Anbindung neuer Datenquellen an das System und deren Konfiguration sowie die Implementierung spezifischer Auswertungsfunktionen.

5.3.1 Anbindung neuer Datenquellen (Specific Transformer)

Um neue Datenquellen, welche Daten in einer neuen Struktur zur Verfügung stellen, an das Visualisierungssystem anzubinden, ist ein spezifischer Transformer (*Specific Transformer*) nötig, wie in Kapitel 4 begründet wurde. Er übernimmt den Import der Daten aus der Quelle und transformiert Daten einer bestimmten Datenstruktur in das generische Datenmodell. *Specific Transformer* werden als interne oder externe Komponenten realisiert. Zur Realisierung eines *Specific Transformers* wird eine Spezifikation bereitgestellt. Die transformierten Daten werden dem *Transformer Manager* des Visualisierungssystems zur Verfügung gestellt.

Die Daten können konzeptionell entweder über einen Pull-Mechanismus mittels einer Anfrage vom *Transformer Manager* angefordert werden oder ohne vorhergehende Anfrage über einen Push-Mechanismus vom *Specific Transformer* an den *Transformer Manager* gesendet werden:

Pull-Mechanismus: Der *Transformer Manager* kann die Schnittstellen der *Specific Transformer* einlesen und über einen Pull-Mechanismus neue transformierte Daten in bestimmten Zeitintervallen anfordern. Jeder neue *Specific Transformer* muss in das System eingetragen werden, damit er erkannt wird und vom System genutzt werden kann.

Push-Mechanismus: Die *Specific Transformer* können neue Daten mithilfe eines Push-Mechanismus unmittelbar nach deren Transformation über eine festgelegte Schnittstelle an den *Transformer Manager* senden, ohne dass der *Transformer Manager* eine Anfrage initiieren muss. Die Schnittstelle zum *Transformer Manager* muss von jedem *Specific Transformer* implementiert werden.

In dieser Arbeit soll die Kommunikation zwischen den *Specific Transformer* und dem *Transformer Manager* mithilfe eines Push-Mechanismus implementiert werden. Eine Push-basierte Kommunikation ist performanter, da redundante Abfragen entfallen und die Datenlatenz minimiert wird. Zudem reduziert die N-zu-eins-Beziehung zwischen den *Specific Transformer* und dem *Transformer Manager* den Konfigurationsaufwand bei der Anbindung neuer *Specific Transformer*.

5.3.2 Systemkonfiguration für neue Datenquellen

Die neuen *MapObjects* und *Events* werden getrennt voneinander an den *Transformer Manager* gesendet, um Konflikte bei der Persistierung zu vermeiden. Dieses Problem wird in Kapitel 5.4.1.6 genauer beschrieben. Um transformierte Daten im Visualisierungssystem nutzen zu können, müssen die Beziehungen zwischen den *MapObjects* und ihren zugehörigen *Events* rekonstruierbar sein. In einer Konfigurationsdatei wird daher für jede Datenquelle die Beziehung der statischen und dynamischen Objekte des transformierten Datensatzes individuell definiert. Statische und dynamische Objekte enthalten einen Primärschlüssel. Dynamische Objekte enthalten zusätzlich noch einen Fremdschlüssel, welcher auf das zugehörige statische Objekt referenziert. In der Konfigurationsdatei wird die Beziehung zwischen zusammengehörenden statischen und dynamischen Objekten durch den Primärschlüssel des statischen Objekts und den Fremdschlüssel des dynamischen Objekts definiert.

5.3.3 Bereitstellung spezifischer Auswertungsfunktionen

Spezifische Auswertungsfunktionen werden über eine Geschäftslogikkomponente bereitgestellt. Eine Auswertungsfunktion kann in Form einer Java-Komponente, beispielsweise als POJO, Java Bean oder EJB, implementiert und

als Java Archive (JAR) zur Verfügung gestellt werden. Die Auswertungsfunktionen werden als Dependencies in das Visualisierungssystem eingebunden. Spezifische Auswertungsfunktionen werden einem `EventType` spezifisch zugeordnet, da sich die Skalenniveaus, Beobachtungsintervalle und Kontexte der dynamischen Daten der Ereignisarten unterscheiden und die Anwendbarkeit beziehungsweise die Sinnhaftigkeit der Funktionsanwendung beeinflussen. Der Klassenpfad der auszuführenden Methode und ihre Beschreibung werden in einer Konfigurationsdatei hinterlegt, aus welcher sie mithilfe von Reflection[28] aufgerufen wird.

Die Entwicklung einer spezifischen Auswertungsfunktion durch einen Systemkonfigurator erfordert die Kenntnis des ursprünglichen Datentyps der auszuwertenden *EventEntries*. Das Ergebnis der Funktion ist als Zeichenkette zurückzugeben, um die Implementierung eines einheitlichen Mechanismus für den Aufruf und die Rückgabe des Ergebnisses zu ermöglichen. Das Ergebnis ist nur für den Anwender relevant und wird nicht weiter vom System verwendet. Es wird auf der Benutzeroberfläche ausgegeben, wobei das Format die Darstellung nicht beeinflusst.

5.4 Spezifikationen der Systemkomponenten

Nun gilt es, die Spezifikationen der Systemkomponenten genauer zu betrachten. Die Systemkomponenten sind dabei ihrer jeweiligen Architekturschicht zugeordnet. Ein detailliertes Komponentendiagramm befindet sich in Abbildung 24 in Anhang A.

5.4.1 Komponenten der Geschäftslogik

5.4.1.1 Persistence Engine

Die *Persistence Engine* ist die Datenzugriffskomponente des Visualisierungssystems. Die Geschäftslogikkomponenten *Transformer Manager*, *Validator*, *Data Arranger* und *Resource Manager* greifen über die zugehörige Schnittstelle auf die *Persistence Engine* zu. Die Datenbank wird über eine Schnittstelle an das Visualisierungssystems angebunden.

[28] Das Reflection API dient der dynamischen Analyse und Benutzug von Klassenbestandteilen zur Laufzeit der Anwendung. Es wird verwendet, um Klassen, deren Namen zum Zeitpunkt der Kompilierung noch nicht bekannt sind, zur Laufzeit einzubinden (Grude, 2005).

5.4.1.2 Configurator

Es liegen zwei Konfigurationsdateien vor, welche die Referenzen der Datenquellen beziehungsweise der spezifischen Auswertungsfunktionen enthalten. In den Datenquellenreferenzen sind die Beziehungen zwischen den statischen und dynamischen Objekten definiert. Der *Configurator* bildet die Referenzen der Konfigurationsdateien intern ab.

5.4.1.3 Specific Transformer

Der Import und die spezifische Transformation der heterogenen Daten erfolgt durch *Specific Transformer*. Für jede Datenstruktur muss ein eigener *Specific Transformer* entwickelt werden, für dessen Realisierung eine Spezifikation der zu implementierenden Schnittstelle zur Verfügung gestellt wird.

5.4.1.4 Transformer Manager

Der *Transformer Manager* ist eine zentrale Komponente der Geschäftslogik. Er ist für den Empfang der von den *Specific Transformer* transformierten Daten und deren Persistierung mithilfe des *Validators* zuständig.

5.4.1.5 Validator

Der *Validator* validiert die neuen transformierten Daten mithilfe des Namens des eindeutigen Indexes der zu transformierenden Rohdaten, welcher im Folgenden als Rohdatenschlüssel bezeichnet wird. Über die Schnittstelle der *Persistence Engine* gleicht er die neuen transformierten Daten mit den bereits persistierten Daten in der Datenbank ab. Abhängig davon, ob die transformierten Objekte oder eine frühere Version bereits in der Datenbank gespeichert sind, wird eine Persistierung oder Aktualisierung von der *Persistence Engine* ausgeführt.

5.4.1.6 Data Arranger

Bei der Persistierung können Konflikte auftreten, wenn statische und dynamische Daten, welche in Beziehung zueinander stehen, aus unterschiedlichen Datenquellen stammen. Die Daten würden von unterschiedlichen *Specific Transformer* transformiert werden. Dadurch können die *Events* zuerst im Visualisierungssystem vorliegen. Während ihrer Laufzeit existiert dann noch kein zugehöriges *MapObject* im System. Dies stört den Arbeitsfluss des Visualisierungssystems. Um diese Konflikte bei der Persistierung zu vermeiden, werden *MapObjects* und *Events* daher zunächst ohne Beziehung zueinander in

der Datenbank abgelegt. Die Beziehungen werden mithilfe des *Data Arrangers* rekonstruiert.

5.4.1.7 Function Invoker

Der *Function Invoker* hat die Aufgabe, eine spezifische Auswertungsfunktion mithilfe des *Configurators* auszuführen. Ihm werden die Identifikation der Funktion und die Daten zur Auswertung in Form eines Dictionary, welches die Zeitstempel und Werte der zu analysierenden *EventEntries* als KVP enthält, zur Verfügung gestellt. Mithilfe der Identifikation kann er die Referenz der Funktion über eine Schnittstelle auflösen und auf ihre Methoden zugreifen.

5.4.1.8 Resource Manager

Der *Resource Manager* ist die Datenzugriffskomponente des Systems und hat die Aufgabe, die persistierten transformierten Daten ressourcenorientiert für die Präsentationsschicht zur Verfügung zu stellen. Er soll Operationen anbieten, mit welchen *MapObjects*, *Events*, die Konstanten der Enumerationen und die spezifischen Auswertungsfunktionen angefordert werden können, wobei *MapObjects* und *Events* getrennt voneinander aufgerufen werden sollen.

5.4.2 Komponente der Präsentationsschicht (Visualization Manager)

Die Präsentationsschicht stellt die Benutzeroberfläche des Visualisierungssystems dar. Sie beinhaltet nur eine Komponente, den *Visualization Manager*, welche auf die Services des *Resource Managers* zugreift. Sie stellt dem Anwender die Funktionalitäten des Visualisierungssystems auf einer Benutzeroberfläche zur Verfügung. Abbildung 25 in Anhang A zeigt ein Mockup der Oberfläche.

Beim Aufruf des Visualisierungssystems wird eine Karte initialisiert und auf den Standort des Nutzers zentriert. Der Anwender wählt eine ihn interessierende Kategorie von *MapObjects* aus. Die *MapObjects* werden durch ihre definierte Geometrie und Farbe auf der Karte visualisiert. Punktgeometrien werden durch einen Pointer gekennzeichnet, während Gebiete als Polygone abgebildet werden. Durch Anklicken eines *MapObjects* werden dessen statische Informationen angezeigt.

Der *Visualization Manager* betrachtet für die Visualisierung und Auswertung nur die dynamischen Daten eines selektierten *MapObjects*. Alle *Events* des selektierten *MapObjects* werden über die entsprechende Operation aufgerufen. Über eine Auswahlliste werden die zur Verfügung stehenden Ereignisarten für das selektierte *MapObject* angezeigt. Nach der Selektion einer Ereignisart

werden die statischen Informationen der dynamischen Daten angezeigt. Diese enthalten beispielsweise die Gesamtanzahl verfügbarer Stellplätze des Parkplatzes. Es kann nun eine interessierende Eigenschaft ausgewählt werden, wie die Anzahl freier Stellplätze. Die Werte der Keys der *EventEntries* werden dazu gefiltert und in einer Auswahlliste angeboten. Der Start- und Endpunkt des Auswertungszeitraums werden über zwei Felder zur Zeitstempelauswahl festgelegt.

Durch die Auswahl von Ereignisart, Eigenschaft und Auswertungszeitraum wird eine Filteroperation definiert, mit deren Hilfe die relevanten *EventEntries* zeitlich sortiert in ein Dictionary überführt werden. Das Dictionary enthält KVP, wobei die Zeitstempel der *Events* als Keys im Datentyp „String" und die Values der *EventEntries* als Values vorliegen. Die Values der *EventEntries* wurden in den Datentyp „String" transformiert, der ursprüngliche Datentyp wurde jedoch in den Metadaten aufbewahrt (Vgl. Kapitel 4.2.1). Bei der Erstellung des Dictionary werden die Values zurück in ihren ursprünglichen Datentyp überführt. Basierend auf dem neu erzeugten Dictionary wird ein Diagramm erstellt. Durch eine Aggregationsfunktion wird die Darstellung des Diagramms anpassbar gestaltet, um die Auswertung nach Minuten, Stunden, Tagen und Monaten zu ermöglichen.

Nach der Selektion einer Ereignisart werden alle definierten Referenzen der spezifischen Auswertungsfunktionen, die dem ausgewählten `EventType` zugeordnet sind, aufgerufen. Ihre Beschreibungen werden in einer Auswahlliste dargestellt. Das Ergebnis der ausgewählten Funktion wird angefordert, als Zeichenkette zurückgegeben und auf der Benutzeroberfläche angezeigt. Die elementaren Auswertungsfunktionen werden automatisch auf das Dictionary angewendet und ihre Ergebnisse auf der Benutzeroberfläche ausgegeben.

6 Implementierung des Visualisierungssystems

In diesem Kapitel werden relevante Punkte der Implementierung des Visualisierungssystems beschrieben. Der Arbeit wird eine JavaScript Dokumentation (JsDOC) und eine Java Dokumentation (JavaDOC) beigelegt.

6.1 Konfiguration des Projekts

Die Beispieldatensätze von Rovereto und goBerlin liegen als Dateien vor. Um die Funktionalität des Visualisierungssystems zu testen, werden die Datensätze manipuliert. Die Auswirkungen der Änderungen innerhalb der Daten lassen Rückschlüsse darauf zu, ob das System die funktionalen Anforderungen erfüllt und fehlerfrei funktioniert. Die Daten der Dateien werden unter Verwendung von Express.js[29] über einen RESTful Webservice zur Verfügung gestellt. Sie werden in ihrem ursprünglichen Datenformat, JSON beziehungsweise XML, dargestellt.

Es wird das Projekt „visualizationsystem-datasource" angelegt. Darin wird mithilfe von Node.js und Express.js ein app.js Script implementiert, welches die Daten mittels der HTTP-Methode `GET` über verschiedene URI zur Verfügung stellt (siehe Tabelle 7).

Tabelle 7: Webservice-Adressen der Daten

URI	Projekt	Beschreibung
http://server[30]/visualizationsystem-datasource/parkingStationsClock	Streetlife	Statische Daten von Parkplätzen mit Parkuhr
http://server/visualizationsystem-datasource/parkingStationsFee	Streetlife	Statische Daten von Parkplätzen mit Parkgebühr
http://server/visualizationsystem-datasource/streetlogrovereto	Streetlife	Dynamische Daten der Aufzeichnungen des Parkstatus
http://server/visualizationsystem-datasource/parkingStationsFree	Streetlife	Statische Daten der freien Parkstationen
http://server/visualizationsystem-datasource/stammdaten	goBerlin	Statische Daten von Schulen
http://server/visualizationsystem-datasource/schuelerZahlen	goBerlin	Dynamische Daten der Schülerzahl der Schulen

[29] Express.js ist ein minimalistisches Node.js-Webanwendungsframework.

[30] Die Daten werden über den Port 4730 zur Verfügung gestellt.

Die Komponenten des Visualisierungssystems werden entsprechend ihrer Spezifikation als Java-EE- oder JavaScript-Komponenten realisiert (Vgl. Kapitel 5.4). Zur Bereitstellung der Komponenten werden daher sowohl ein Java-EE-, als auch ein JavaScript-Anwendungsserver benötigt. Es werden konkret ein WildFly-Anwendungsserver und ein Node.js-Webserver verwendet (Vgl. 2.5.2). Die Installation der Server wird in der beigefügten Installationsanleitung beschrieben.

Die Persistierung der Daten erfordert die Anbindung einer Datenbank an das Visualisierungssystem. Es wird eine relationale MySQL-Datenbank verwendet, die mit dem WildFly-Anwendungsserver verbunden wird.

Insgesamt werden drei Projekte für die Umsetzung des Visualisierungssystems anlegt: eines für die Java-EE-Komponenten, eines für die JavaScript-Komponente und eines für den *Specific Transformer* für die Rovereto-Daten.

Die Quelltexte der Projekte werden während der Implementierung mithilfe des Versionsverwaltungsframeworks Git[31] auf einem GitLab Server verwaltet (siehe Tabelle 8).

Tabelle 8: URI der Repositories zur Verwaltung der Quelltexte der Projekte

URI	Repository zur Verwaltung des Quelltexts...
http://server[32]/visualizationsystem	...der Java EE Komponenten
http://server/visualizationsystem-ui	...der JavaScript Komponente
http://server/rovereto-transformer	...des externen *Specific Transformers*

Für die Implementierung des Visualisierungssystems werden ein lokaler Entwicklungsserver und ein Linux-Live-Server genutzt. Der Entwicklungsserver dient zu Testzwecken. Auf dem Live-Server wird der aktuellste Entwicklungsstand veröffentlicht. Dazu wird auf dem Live-Server das Continuous Integration System Jenkins[33] verwendet, welches die aktuellste Quelltextversion von GitLab bereitstellt und baut.

[31] https://git-scm.com.

[32] Es wird der Server-URL vps94307.ovh.net verwendet.

[33] https://jenkins-ci.org.

6.1.1 Konfiguration des Java-EE-Projekts

Die Java-EE-Anwendung kapselt alle Java-EE-Komponenten. Sie wird mithilfe des Build-Management-Tools Maven[34] als Projekt visualizationsystem umgesetzt. In Maven werden für das Projektmanagement Module für die einzelnen Komponenten angelegt sowie ein weiteres Modul für ein Enterprise Archive (EAR). Die Abhängigkeiten der Module werden durch CDI aufgelöst (Vgl. Kapitel 2.5.1.1.3). Die Modulstruktur des Projekts visualizationsystem ist in Abbildung 17 dargestellt. Jedes Modul besitzt ein eigenes POM[35]. Zusätzlich wird ein Parent POM angelegt. Im Parent POM werden die Submodule sowie ihre Abhängigkeiten und deren Versionen werden als Konfiguration definiert und verwaltet.

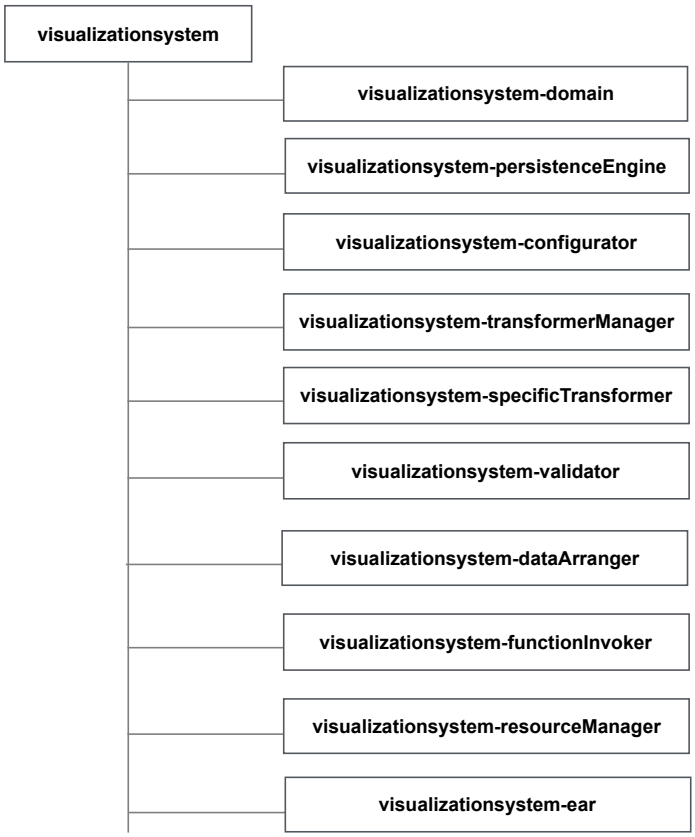

Abbildung 17: Modulstruktur des Projekts visualizationsystem in Maven

Aus den Spezifikationen der Komponenten ergeben sich die Abhängigkeiten und das Packaging der Module (siehe Tabelle 9). Java EE API, Jackson API und Hibernate API dienen der Annotation und Konfiguration der Klasssen zur

[34] https://maven.apache.org.

[35] Project Object Model.

Persistierung. SLF4J API[36] ist eine Logging-Fassade[37] für Java-EE-Anwendungen. JSON API und Jackson API dienen der Bearbeitung des JSON-Formats, insbesondere der Konvertierung von JSON in Java-Objekte und umgekehrt. Tyrus API[38] unterstützt die Implementierung eines WebSocket Clients. Die Spezifikation enthält Encoder und Decoder für *MapObjects* und *Events*.

Tabelle 9: Abhängigkeiten der Module

Modul	Packaging	Abhängigkeiten
domain	JAR	Java EE API, Jackson API, Hibernate API
persistenceEngine	EJB[39]	Java EE API, SLF4J API, domain
configurator	EJB	Java EE API
transformerManager	WAR[40]	Java EE API, SLF4J API, domain, validator, dataArranger, Spezifikation decoder
specificTransformer	EJB	JSON API/Jackson API, Tyrus API, SLF4J API, domain, Spezifikation encoder
validator	EJB	Java EE API, SLF4J API, domain, persistenceEngine, configurator
dataArranger	EJB	Java EE API, SLF4J API, domain, persistenceEngine, configurator
functionInvoker	EJB	Java EE API, SLF4J API, configurator, ressourceManager, specificFunction
resourceManager	WAR	Java EE API, domain, persistenceEngine, configurator
ear	EAR	domain, transformerManager, specificTransformer, resourceManager

Die Anwendung wird als EAR gekapselt, welches jedoch nur ausgewählte Module enthält (siehe Abbildung 18). Die weiteren Module werden durch die Auflösung der Abhängigkeiten geladen.

[36] Simple Logging Facade for Java (SLF4J).

[37] Die Logging-Fassade ermöglicht die Speicherung von Ereignissen, beispielsweise die erfolgreiche Transformation eines Objekts, in einer Log-Datei und zeigt die Log-Einträge in der Konsole an.

[38] https://tyrus.java.net.

[39] Ein EJB-Modul enthält EJB und wird als JAR bereitgestellt.

[40] Web Application Archive.

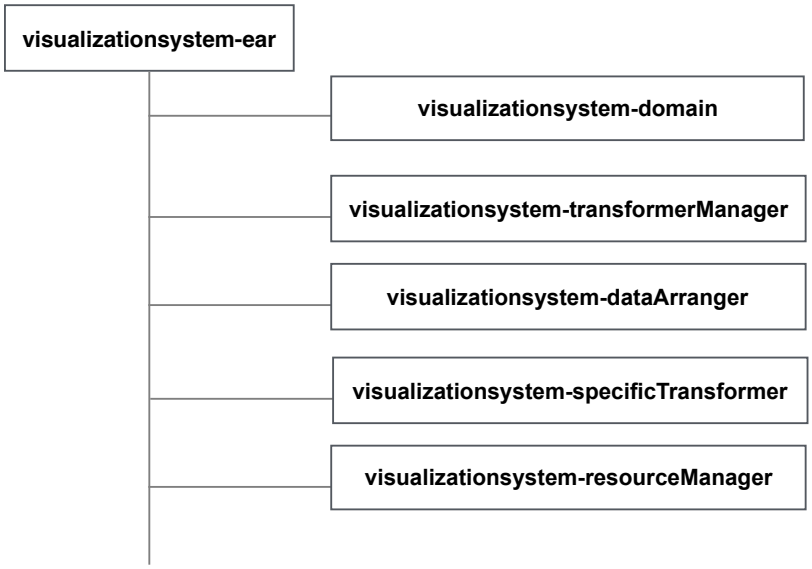

Abbildung 18: Module des EAR

6.1.2 Konfiguration des JavaScript-Projekts

Das JavaScript-Projekt besteht aus der JavaScript-Komponente *Visualization Manager*. Sie wird als AngularJS-Anwendung in einem Node.js-Webserver realisiert. Das Projekt wird vom JavaScript Task Runner Grunt[41] und vom Paketverwaltungstool Bower[42] unterstützt.

Der *Visualization Manager* besitzt Abhängigkeiten, die aus der Spezifikation der Komponente abgeleitet werden. Die wichtigsten Abhängigkeiten sind in Tabelle 10 aufgelistet.

Die Bibliotheken sind in einem Bower Respository verfügbar. Sie werden in der Datei Bower.JSON als Dependencies eingetragen. Beim Kompilieren des Projekts werden die Bibliotheken vom Repository zur Verfügung gestellt. Sie enthalten JavaScript- und CSS-Dateien. Mithilfe von Grunt werden die Bibliotheken als Abhängigkeiten injiziert. Die Injektion ist durch ein Script in einer Gruntfile.js Datei automatisiert.

[41] Grunt ist ein Node.js-basierter JavaScript Task Runner zur Aufgabenautomatisierung (Grunt, 2015).

[42] Bower ist ein JSON-basierter Frontend-Paketmanager, der alle Abhängigkeiten des Frontends zu externen JavaScript Bibliotheken lädt und für die Einbindung im Projekt zur Verfügung stellt (Bower, 2015).

Tabelle 10: Übersicht der wichtigsten Abhängigkeiten des *Visualization Manangers*

Abhängigkeit	Verwendete JavaScript Bibliothek
AngularJS	angular um eine AngularJS-Anwendung zu realisieren
Google Maps	Direktive angular-google-map[43] zur Verwendung des Google Maps API zur Erstellung und Verwaltung einer Karte
Chart.js	Direktive chart.js zur Verwendung des Chart.js API zur Erstellung von Diagrammen
D3.js	Direktive angularjs-nvd3-directives-master zur Verwendung des D3.js API zur Erstellung von Diagrammen
Foundation	CSS-Framework foundation[44] zur Gestaltung der Benutzeroberfläche

6.2 Implementierung der Module

Hier wird die Implementierung der einzelnen Module anhand ihrer Spezifikationen und Konfigurationen beschrieben.

6.2.1 Domain

Die Klassen des Datenmodells werden mithilfe des JPA in der relationalen Datenbank persistiert. Die zu persistierenden Klassen werden mit der JPA Annotation `@Entity` versehen. Die Beziehungen zwischen den Entitäten werden über die `cascade`-Annotation kaskadierend gestaltet.

In der persistence.xml werden Datenbankkonfigurationen für den JPA Provider festgelegt. Sie enthält den gewählten Namen der Datenbankkonfiguration und die Konfiguration der Datenbankverbindung.

6.2.2 Komponenten der Geschäftslogik

6.2.2.1 Persistence Engine

Die *Persistence Engine* besitzt spezielle, mithilfe von JPQL implementierte Query-Operation für SQL-Anfragen an die Datenbank. Entsprechend des Transformationskonzepts werden die Rohdatenschlüssel als `DescriptionEntry` abgebildet. Mithilfe dieser Information werden *MapObjects* und *Events* aus der Datenbank abgefragt.

[43] http://angular-ui.github.io/angular-google-maps/#!/.

[44] http://foundation.zurb.com/docs/.

Mittels der Query-Operation getMapObject kann ein bestimmtes *MapObject* aus der Datenbank angefordert werden. Die Operation enthält als Parameter den Rohdatenschlüssel und dessen Wert. Die Anfrage mit der Query-Operation getEvent läuft analog ab.

Mit der Operation findEventWithoutRelationship werden *Events* zurückgegeben, die ohne Beziehung in der Datenbank persistiert sind. Durch die Operation getMapObject wird das zugehörige *MapObject* eines *Events* gefunden. Über die Operation checkIfMapObjectExists wird abgefragt, ob ein bestimmtes *MapObject* in der Datenbank persistiert ist. Die Ausgabe ist „true" oder „false".

Des Weiteren verfügt die *Persistence Engine* über die CRUD-Funktionen, welche durch eine Fassade[45] realisiert werden. Die abstrakte Fassadenklasse enthält die CRUD-Methoden und delegiert die Funktionalität an andere Klassen. Es werden nur zwei Fassaden benötigt, MapObjectFacade und EventFacade, da die Beziehungen zwischen den Klassen kaskadierend gestaltet sind. Die Fassade stellt den anderen Komponenten des Visualisierungssystems eine Schnittstelle zum Ausführen der CRUD-Operationen zur Verfügung. Die CRUD-Operationen nutzen die Dienste des Entity Managers des Visualisierungssystems.

6.2.2.2 Configurator

Der *Configurator* wird als Stateless EJB implementiert, da gegebenenfalls mehrere Instanzen der Komponente vom *Validator* und *Data Arranger* benötigt werden.

Die zwei Konfigurationsdateien werden im JSON-Format manuell erstellt. Für sie werden zwei Subkomponenten implementiert, *DatasourceReference* und *FunctionReference*. Dadurch können die Daten der Dateien innerhalb des Visualisierungssystems objektorientiert genutzt werden. Sie enthalten die Attribute der entsprechenden Konfigurationsdatei.

Der *Configurator* übergibt die Daten aus den Konfigurationsdateien mithilfe von Java-Stream und des JSON API als Objekte beziehungsweise als Datenquellen- und Funktionsreferenzliste. Bei der Instanziierung eines Konfigurationsobjekts werden die beiden Listen zur Verfügung gestellt.

[45] Eine Fassade ist ein Strukturmuster zur Bereitstellung einer einheitlichen vereinfachten Schnittstelle zu einer Menge von Schnittstellen (Dunkel & Holitschke, 2003).

6.2.2.3 Specific Transformer

Ein *Specific Transformer* wird als Singleton EJB implementiert. Für die Weiterleitung der transformierten Daten werden zwei Client WebSocket Endpoints mithilfe der Konfiguration `@ClientEndpoint` implementiert, *WebSocketClient ForEvent* und *WebSocketClientForMapObject*. Zur Umsetzung wird eine Referenzimplementierung des Java API for WebSocket (JSR 356) in Form des Tyrus API genutzt.

Die Client WebSocket Endpoints werden vom *Specific Transformer* instanziiert. Bei der Instanziierung wird der URI des Server WebSocket Endpoints als Parameter und dem Rohdatenschlüssel als Pfadparameter an den Konstruktur übergeben. Die Client WebSocket Endpoints initiieren mithilfe der `@OnOpen`-Methode über den jeweiligen URI die Verbindung zum Server WebSocket Endpoint.

Neben den üblichen WebSocket-Methoden verfügen die Client WebSocket Endpoints zusätzlich über die Methode `sendObject`, welche vom *Specific Transformer* aufgerufen wird, um ein transformiertes Objekt als Java-Objekt zu senden. *MapObjects* und *Events* werden einzeln an den *Transformer Manager* gesendet. Über eine WebSocket-Verbindung können keine komplexen Objekte gesendet werden. Daten können nur in einem binären oder textuellen Format transferiert werden (Pilgrim, 2013), weshalb die transformierten Daten vor der Übermittlung an den *Transformer Manager* vom *Specific Transformer* umgewandelt werden müssen. Mithilfe des Jackson API implementieren die Client WebSocket Endpoints mit der Schnittstelle `Encoder.Text<T>` jeweils einen Encoder zur Konvertierung von Java-Objekten in JSON-Objekte und deren Kodierung in String. Durch die Nutzung der Client WebSocket Endpoints zur Kodierung wird eine Trennung der Zuständigkeiten erreicht. Die transformierten und kodierten Daten werden als Zeichenketten über die WebSocket-Verbindung an den *Transformer Manager*, welcher Server WebSocket Endpoints zur Verfügung stellt, gesendet. Der Rohdatenschlüssel wird ebenfalls als Zeichenkette mitgesendet, um die *MapObjects* und *Events* validieren zu können (Vgl. Kapitel 5.4.1.5).

Für die Entwicklung eines neuen *Specific Transformers* zur Transformation von Daten einer neuen Datenstruktur werden die Encoder zur Kodierung von *MapObjects* beziehungsweise *Events* sowie die Domain des Visualisierungssystems als Spezifikation in Form eines JAR zur Verfügung gestellt. Die Nutzung der Domain und der Encoder ist erforderlich. Die Implementierung der Client WebSocket Endpoints kann beliebig gestaltet werden und unterliegt

lediglich der Bedingung, dass valide *MapObjects* und *Events* an den *Transformer Manager* gesendet werden.

6.2.2.3.1 Specific Transformer für die goBerlin-Daten

Der *Specific Transformer* für die goBerlin-Daten soll intern implementiert werden. Er wird daher als Maven-Modul des Maven-Projekts visualizationsystem erstellt.

Die dynamischen und statischen Daten werden jeweils über einen URI zur Verfügung gestellt (siehe Tabelle 7), weshalb zwei Subkomponenten, EventTransformerImpl und MapObjectTransformer, erstellt werden. Eine Subkomponente ist für die Transformation von dynamischen Objekten in *Events* zuständig, die andere Subkomponente transformiert statische Objekte in *MapObjects*. Sie werden als Singleton EJB implementiert und mit der Annotation @StartUp versehen, wodurch die EJB vor dem ersten Zugriff auf die Anwendung vom EJB-Container instanziiert wird. Die Subkomponenten verfügen über eine mit @PostConstruct annotierte Funktion, welche mithilfe des Scheduled Periodic Task Prinzips in definierten Zeitintervallen über den URI neue Daten abfragt. Die Daten werden mithilfe des Java-Stream-Konzepts extrahiert und in ein binäres InputStream-Objekt umgewandelt. Von diesem wird mithilfe des Algorithmus SHA-256 ein Hash-Wert erzeugt, welcher in einer Variablen hinterlegt wird. Bei jeder Abfrage wird der Hash-Wert mit dem der vorherigen Abfrage vergleichen.

Sind neue Daten vorhanden, so wird die Funktion transformAndSend aufgerufen. Diese Funktion importiert zunächst die Daten mithilfe von Java-Stream. Anschließend werden die Daten, die im XML-Format vorliegen, mithilfe des JSON API in das JSON-Format geparst. Das JSON API bietet einen XML-Parser an, sodass keine Bibliotheken für die Transformation von XML-Daten in Java-Objekte eingebunden werden müssen. Es wird ein Loopmechanismus verwendet, um die Objekte iterativ mithilfe des JSON API in Java-Objekte beziehungsweise *MapObjects* oder *Events* zu transformieren.

Die statischen Objekte aus dem goBerlin-Projekt besitzen keine Koordinaten, sondern Adressen. Die Koordinaten der Adressen werden während der Transformation mithilfe der Geocoding-Funktion[46] des Google Maps API bestimmt und zum *MapObject* hinzugefügt.

[46] Die Geocoding-Funktion ermöglicht die Konvertierung von Adressen in geografische Koordinaten (Google Inc., 2015b).

Die transformierten Daten werden über den Client WebSocket Endpoint an den *Transformer Manager* gesendet. *Events* und *MapObjects* werden an verschiedene Server WebSocket Endpoint URI gesendet (siehe Kapitel 5.4.1.4). Nachdem alle Objekte transformiert und gesendet wurden, wird die WebSocket-Verbindung getrennt.

6.2.2.3.2 Specific Transformer für die Rovereto-Daten

Der *Specific Transformer* für die Rovereto-Daten soll als externer Transformer implementiert werden. Er wird daher als eigenes Maven-Projekt angelegt und in einer anderen WildFly-Instanz bereitgestellt.

Die Rovereto-Daten werden über vier URI zur Verfügung gestellt (siehe Tabelle 7), weshalb vier Subkomponenten implementiert werden. Drei der vier Subkomponenten transformieren statische Objekte, welche an denselben URI gesendet werden. Der *Specific Transformer* für die Rovereto-Daten implementiert die gleichen Mechanismen wie der *Specific Transformer* für die goBerlin-Daten. Lediglich das Parsen entfällt, da die Rovereto-Daten bereits im JSON-Format vorliegen.

6.2.2.4 Transformer Manager

Der *Transformer Manager* verfügt über zwei Server WebSocket Endpoints, `MapObjectManager` und `EventManager`, die mit der Konfiguration `@ServerEndpoint` und der Angabe des entsprechenden URI `/manager/map Object/{sourceKey}` beziehungsweise `/manager/event/{sourceKey}` erzeugt werden. Sie implementieren mit der Schnittstellen `Decoder.Text<T>` einen Decoder zur Umwandlung der kodierten JSON-Objekte zurück in Java-Objekte. Sendet ein *Specific Transformer* ein transformiertes und kodiertes Objekt an den URI des Server WebSocket Endpoints, wird es von diesem dekodiert und durch `@OnMessage` zusammen mit dem Rohdatenschlüssel des Objekts empfangen (Vgl. Kapitel 2.5.4.3).

6.2.2.5 Validator

Die `valid`-Funktion des *Validators* wird vom *Transformer Manager* zur Validierung jedes transformierten Objekts genutzt. Die transformierten *MapObjects* und *Events* werden mit der ID „null" empfangen, da die ID erst bei der Persistierung in die Datenbank generiert wird. Sie können daher nicht direkt über eine ID mit einem in der Datenbank persistierten Objekt verglichen werden. Stattdessen nutzt der *Validator* den Rohdatenschlüssel, der innerhalb eines `DescriptionEntry` vorliegt, sowie die von der *Persistence Engine* bereitgestellten Query-Funktionen (Vgl. Kapitel 6.2.2.1).

Der Ablauf eines Validationsvorgangs ist in Abbildung 26 in Anhang A für die Validierung eines *MapObjects* in einem Sequenzdiagramm veranschaulicht. Mithilfe des Rohdatenschlüssels und dessen Wert wird über die `MapObjectFacade` eine Datenbankabfrage durchgeführt. Im Fall einer erfolglosen Anfrage wird das transformierte *MapObject* persistiert. Wird jedoch ein bereits persistiertes Objekt mit demselbem Rohdatenschlüssel und -wert gefunden, so wird dieses zurückgeben und geprüft, ob die Objekte identisch sind, wobei die *Events* eines *MapObjects* vom Vergleich ausgeschlossen werden. Sind das persistierte Objekt und das transformierte *MapObject* identisch, so wird das transformierte *MapObject* verworfen. Sind sie nicht identisch, dann werden die `Description`, der `MapObjectType` und die Geometrie des persistierten Objekts aktualisiert. Eventuell vorhandene Beziehungen zu *Events* bleiben bestehen.

Die Validierung von *Events* verläuft analog (siehe Abbildung 27, Anhang A). Sind die *Events* identisch, so wird das neu transformierte *Event* verworfen. Sind sie jedoch unterschiedlich, so wird stattdessen das persistierte *Event* gelöscht und das neu transformierte *Event* persistiert.

Da transformierte *MapObjects* und *Events* unabhängig voneinander validiert und persistiert werden können, müssen mehrere Instanzen des *Validators* erzeugbar sein. Er ist daher als Stateless EJB implementiert.

6.2.2.6 Data Arranger

Der *Data Arranger* wird als Singleton EJB implementiert und mit der Annotation `@StartUp` versehen. Die Funktion `arrange` des *Data Arrangers* wird durch die Annotation `@PostConstruct` direkt nach der Instanziierung der Komponente ausgeführt. Mithilfe der Funktion kann die Beziehung zwischen *Events* und ihrem zugehörigen *MapObject* in der Datenbank hergestellt werden.

Die Funktion besteht aus verschachtelten Loopmechanismen. Zur Veranschaulichung ist der Ablauf in Abbildung 28 in Anhang A in einem Sequenzdiagramm dargestellt. Zunächst fordert der *Data Arranger* die Datenquellenreferenzen vom *Configurator* an, welche zurückgegeben werden. Mittels der Scheduled Periodic Task `findEventWithoutRelationship` wird mithilfe der *Persistence Engine* eine Abfrage in der Datenbank nach beziehungslosen *Events* durchgeführt. Die `EventKeys` der Datenquellenreferenz werden iterativ mit dem Rohdatenschlüssel in den *DescriptionEntries* des *Events* verglichen (Vgl. Abbildung 28). Der entsprechende `MapObjectKey` der Datenquellenreferenz und der Wert des Rohdatenschlüssels werden als Parameter der Funktion `getMapObjectForEvent` verwendet, um das zugehörige *MapObject* anzufordern. Das *Event*

wird mit der Funktion `addEventToMapObject` zum zurückgegebenen *MapObject* hinzugefügt. Über die `MapObjectFacade` wird das persistierte *MapObject* in der Datenbank aktualisiert. Durch die kaskadierende Beziehung zwischen den Klassen *MapObject* und *Event* wird die Beziehung ebenfalls im persistierten *Event* vermerkt. Die Beziehung zwischen dem *Event* und seinem zugehörigen *MapObject* ist damit in der Datenbank hergestellt.

6.2.2.7 Function Invoker

Der *Function Invoker* wird als Stateless EJB implementiert, da mehrere Instanzen vom *Resource Manager* benötigt werden, um Funktionen für mehrere Klienten ausführen zu können. Der Ablauf des Aufrufs einer spezifischen Auswertungsfunktion ist in Abbildung 19 dargestellt. Mit der ID der aufzurufenden spezifischen Auswertungsfunktion und dem Dictionary der zu betrachtenden dynamischen Daten wird ein Funktionsergebnis angefordert (Vgl. Kapitel 5.4.2). Der *Function Invoker* durchsucht mithilfe eines Loopmechanismus die Funktionsreferenzen nach der Referenz mit der angegebenen ID. Über den in hinterlegten Klassenpfad und den Methodennamen kann die Methode der spezifischen Auswertungsfunktion mit Reflection aufgerufen werden. Die entsprechende Methode wird auf das Dictionary angewendet. Das Ergebnis wird ebenfalls mit Reflection als Zeichenkette an den *Function Invoker* zurückgegeben. Der *Function Invoker* gibt das Ergebnis an den *Resource Manager* zurück.

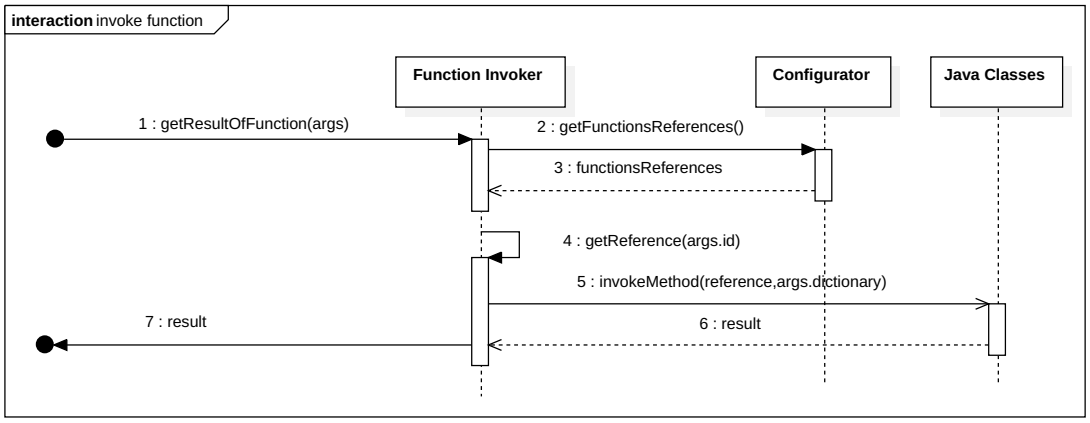

Abbildung 19: Sequenzdiagramm des Aufrufs einer spezifischen Auswertungsfunktion

Die spezifischen Auswertungsfunktionen sind als Java-Klassen implementiert Sie werden als JAR zur Verfügung gestellt und als Dependency in Maven eingebunden.

Im Zuge dieser Arbeit wurde eine spezifische Auswertungsfunktion exemplarisch implementiert. Die Funktion MaximumOnAWednesday gibt den höchsten Wert, der an einem Mittwoch beobachtet wurde sowie den Zeitstempel des Werts aus. Sie ist spezifisch für den MapObjectType ParkingStation.

6.2.2.8 Resource Manager

Der *Resource Manager* stellt Services über HTTP-Methoden zur Verfügung (siehe Tabelle 11), welche als RestEndpoint-Subkomponenten implementiert werden. Sie werden unter Verwendung von JAX-RS Annonationen implementiert und unter dem URL http://server[47]/visualizationsystem-resourceManager/ zur Verfügung gestellt.

Der Aufruf der Operation getMapObjects gibt eine Liste von *MapObjects* zurück. Der Inhalt dieser Liste soll durch eine Filterung anpassbar sein, damit nicht alle in der Datenbank verfügbaren *MapObjects* aufgerufen werden, da dies zu langen Ladezeiten führen kann. Daher wird ein Parameter konzipiert, der die Anfrage spezifiziert. Mit dem Query-Parameter {MapCoordinates} werden die *MapObjects* danach gefiltert, ob sie im aktuellen Kartenausschnitt lokalisiert sind. Dadurch werden nur die tatsächlich darstellbaren *MapObjects* angefordert und die Ladezeit verkürzt. Die Umsetzung wurde aus zeitlichen Gründen nicht durchgeführt.

Eine spezifische Auswertungsfunktion kann über den *Resource Manager* vom *Function Invoker* ausgeführt werden. Mittels der Operation getEnumValues greift der *Resource Manager* auf die Enumeration zu und gibt die Konstanten aus, die beispielsweise als Filterkategorien für die *MapObjects* dienen. Über die Operation getFunctionsReferences gibt der *Configurator* eine Liste aller in der Konfigurationsdatei vorhandenen spezifischen Auswertungsfunktionen zurück. Mittels der Operation getResultOfFunction wird das Ergebnis einer spezifischen Auswertungsfunktion vom *Function Invoker* angefordert. Die Parameter der Funktion, welche die ID der spezifischen Auswertungsfunktion und das Dictionary umfassen, werden in einem JSON-Objekt als Parameter (args) gekapselt. Dadurch werden Probleme durch die Implementierung von spezifischen Auswertungsfunktionen, welche unterschiedlich viele Parameter erfordern, vermieden.

[47] vps94307.ovh.net.

Tabelle 11: HTTP-Methoden des *Resource Managers*

Methode	URI	Rückgabe
GET	getMapObjects/[?{parameter}]	Liste einer Menge von *MapObjects*
GET	getEvents/[?{parameter}]	Liste aller *Events* eines *MapObjects*
GET	getEnumValues/[?{parameter}]	Liste der Konstanten einer Enumeration
GET	getFunctionsReferences/	Liste aller spezifischen Funktionen
POST	getResultOfFunction/[?{parameter}]	Ergebnis der aufgerufenen spezifischen Funktion

6.2.3 Komponente der Präsentationsschicht (Visualization Manager)

Entsprechend der Komponentenspezifikation wird der *Visualization Manager* als SPA umgesetzt (Vgl. Kapitel 5.4.2). Die Implementierung wird unter Verwendung des MVC-Patterns realisiert.

Die Ordnerstruktur des Teilprojekts kann beliebig gestaltet werden. Sie wird an der Spezifikation des Visualisierungssystems ausgerichtet. Die Funktionalitäten des MVC-Patterns werden in einem AngularJS-Modul eingebunden, wodurch alle Elemente bei der Initialisierung des Moduls geladen werden.

Abbildung 20 zeigt die Struktur des *Visualization Managers*. Aus der Abbildung ist die Aufteilung der einzelnen Komponenten der Module, Direktiven, Views und Controller in einzelne JavaScript-Dateien ersichtlich.

Durch die Direktive ng-app im HTML-Wurzelelement der index.html wird von AngularJS erkannt, dass sich die AngularJS-Anwendung in diesem Element befindet und diese ausgeführt. Das Hauptmodul initialisiert den *Visualization Manager* sowie sämtliche Bibliotheken und Services zum Aufruf der Daten vom *Resource Manager* (Vgl. Kapitel 5.4.1.8). Beim Aufruf der Anwendung werden sämtliche Controller mithilfe der Router instanziiert. Die spezifizierten Funktionalitäten werden als Funktionen innerhalb der Controller auf der Benutzeroberfläche zur Verfügung gestellt.

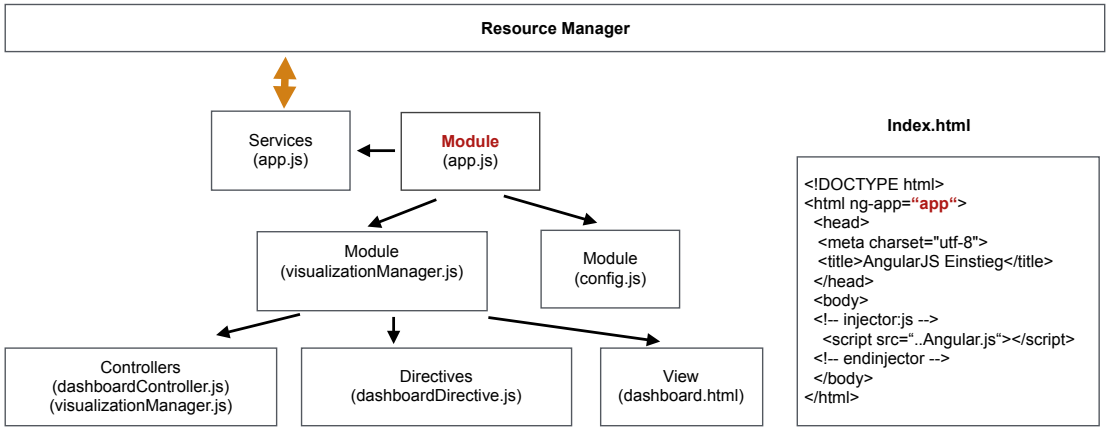

Abbildung 20: Modulkonzept des *Visualization Managers*

Geoobjekte nach Kategorien filtern

Beim Aufruf der SPA werden sämtliche `MapObjectTypes` über den *Resource Manager* geladen (Vgl. Tabelle 11) und in Form einer Liste angezeigt. Sie dient dem Benutzer als Filter zur Auswahl des gewünschten `MapObjectTypes`. Bei der Auswahl eines `MapObjectTypes` werden die entsprechenden *MapObjects* geladen (Vgl. Tabelle 11) und auf der Karte angezeigt.

Es wurde ein Translator für die Konstanten der Enumerationen `MapObjectType` und `EventType` implementiert, um auf der Benutzeroberfläche eine verbesserte Darstellung auf Deutsch zu ermöglichen.

Auf der Karte navigieren und ein Geoobjekt selektieren

Durch die Verwendung der Drittanbieterbibliothek angular-google-maps werden die Funktionen des Google Maps API in Form von Direktiven zur Verfügung gestellt. Ihre Funktionalität wird durch eine Geolocator-Funktion im Controller erweitert, um den Benutzerstandort anhand der IP-Adresse zu lokalisieren und den entsprechenden Kartenausschnitt zu laden. Zusätzlich wird ein Menü zur Schnellauswahl einer Stadt angeboten.

Die Geoobjekte werden mit einer Klickfunktion versehen, welche die statischen Informationen aus den *DescriptionEntries* eines *MapObjects* mithilfe der Direktive <ui-gmap-window> in einem Info-Window anzeigt. Existieren keine dynamischen Daten zu einem Geoobjekt, so wird eine Meldung im Info-Window angezeigt. Bei der Selektion eines *MapObjects* werden die zugehörigen *Events* über den URI `getEvents/{mapObject-ID}` geladen.

Visualisieren und auswerten

Die `EventTypes` der geladenen *Events* werden mithilfe einer Funktion in eine Liste gefiltert. Die Auswahl einer Eigenschaft wird analog implementiert. Nach der Auswahl einer Ereignisart werden die statischen Informationen der dynamischen Daten angezeigt. Bei der Iteration der Elemente wird das erste, welches den Primärschlüssel der Rohdaten enthält, übersprungen, da dieser für den Nutzer in der Regel irrelevant ist und daher nicht angezeigt werden soll.

Mittels eines DateTimePickers[48] kann der Auswertungszeitraum festgelegt werden. Die *EventEntries* werden bei Selektion einer Eigenschaft entsprechend gefiltert. Es findet eine Vorauswahl der Zeitpunkte durch einen Watcher statt, sodass die Zeitstempel des ältesten und des neuesten verfügbaren *EventEntries* angezeigt werden.

Die gefilterten Daten werden in einem Diagramm visualisiert. Im Rahmen der Implementierung werden nur ratioskalierte Attribute betrachtet. Prinzipiell sind jedoch auch Visualisierungsmechanismen und spezifische Auswertungsfunktionen für nicht-ratioskalierte Attribute umsetzbar. Für die Darstellung der ratioskalierten Werte wird ein Punktdiagramm gewählt, in welchem die Zeit auf der x-Achse und die absolute Häufigkeit auf der y-Achse aufgetragen wird. Die Darstellung der Werte ist durch eine Aggregation der Zeiteinheiten anpassbar.

Die Diagrammerstellung wurde zunächst mithilfe der Direktive chart.js implementiert. Die Zeitachse wurde jedoch nicht korrekt skaliert, da die Zeitstempel von Chart.js als Zeichenketten interpretiert wurden und die Abstände der Messpunkte daher nicht den realen Zeitabständen entsprachen (siehe Abbildung 21).

[48] https://github.com/Eonasdan/bootstrap-datetimepicker.

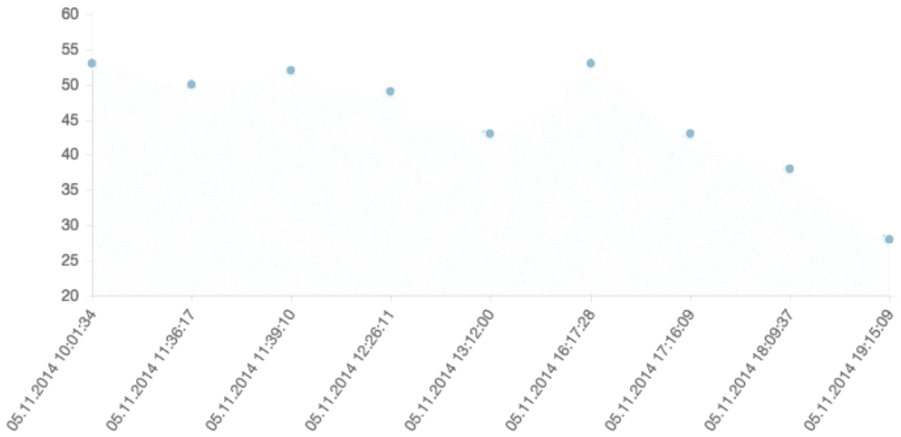

Abbildung 21: Mit Chart.js erstelltes Punktdiagramm

Zur Lösung dieses Problems wurde die Diagrammerstellung stattdessen mithilfe der D3.js Direktive implementiert (siehe Abbildung 22). Das Chart.js Diagramm ist zwar ansprechender gestaltet, jedoch unproportional skaliert. Dagegen ist der reale zeitliche Verlauf der Veränderung der Eigenschaft im D3.js Diagramm korrekt abgebildet.

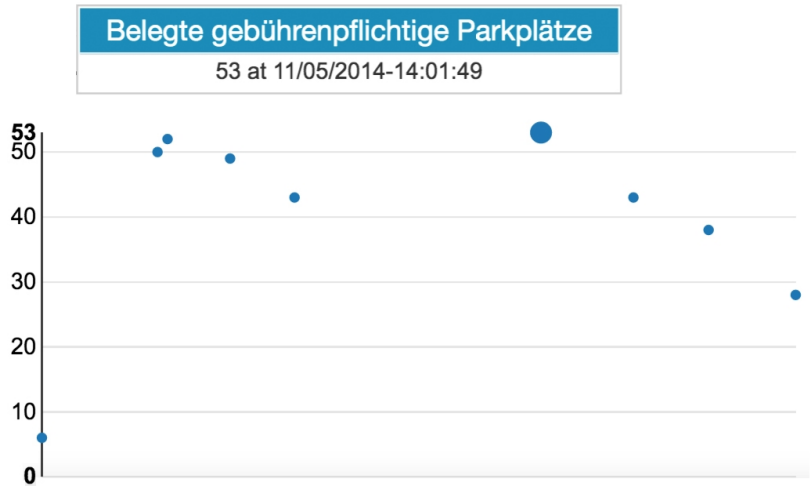

Abbildung 22: Mit D3.js erstelltes Diagramm

Die elementaren Auswertungsfunktionen sind fest implementiert und werden automatisch ausgeführt. Die spezifischen Auswertungsfunktionen werden in einer Auswahlliste bereitgestellt. Dazu werden die Funktionsreferenzen des selektierten `EventTypes` über den URI `getFunctionsReferences/` geladen. Die Ergebnisse aller Funktionen werden als Zeichenketten zurückgegeben und angezeigt.

7 Ergebnisse und Evaluation

Das Ziel dieser Arbeit war der Entwurf und die Entwicklung eines Visualisierungssystems für eine Verkehrsplanungsplattform, welches die Nutzung heterogener und verteilter Datenbestände ermöglicht. Abschließend gilt es nun, die Ergebnisse der Arbeit zusammenzufassen und zu evaluieren.

Das Visualisierungssystem wurde in zwei Teilanwendungen entwickelt: einer Java-EE-Anwendung und einer JavaScript-Anwendung. Die Umsetzung der Teilanwendungen und ihre Funktionalitäten werden im Folgenden bewertet.

Evaluation der Java-EE-Anwendung

Die Java-EE-Anwendung wurde nach dem Vorbild eines Data Warehouse Systems entwickelt. Das DWS erwies sich als sinnvoller Ausgangspunkt für die Entwicklung des Visualisierungssystems, da es die Nutzung heterogener und verteilter Datenbestände unterstützt und eine Vielzahl von Realisierungsmöglichkeiten bekannt ist, deren Vor- und Nachteile in die Überlegungen zur Konzeption des Visualisierungssystems eingeflossen sind.

Die Realisierung der Java-EE-Anwendung erforderte die Bewältigung mehrerer anspruchsvoller Problemstellungen. Zunächst musste ein Transformationskonzept entwickelt werden, welches eine möglichst verlustfreie Transformation heterogener Daten in ein generisches Datenmodell gewährleistet, um eine datensatzübergreifende Visualisierung und Analyse zu ermöglichen. Es gelang die Entwicklung eines universellen Transformationskonzepts, das alle Informationen der Rohdaten verlustfrei abbilden kann. Ferner musste ein Datenmodell erstellt werden, dass sowohl statische Geodaten als auch deren dynamische Informationen erfassen kann. Dies gelang unter Verwendung der ISO-Normen des Simple Feature Modells und des entwickelten Transformationskonzepts.

Um die Abhängigkeit der Datengeber vom Visualisierungssystem zu reduzieren, wurden die Funktionen der Datenextraktion und -transformation in einer eigenen Komponente, dem spezifischen Transformer, gekapselt. Spezifische Transformer sind vom Visualisierungssystem unabhängig und können extern implementiert werden. Dies hat den Vorteil, dass ein Datengeber einen eigenen spezifischen Transformer für die Bereitstellung seiner Daten entwicklen kann, dessen Entwicklung nur wenigen Restriktionen unterliegt. Es kann eine beliebige Programmiersprache verwendet werden. Die transformierten Objekte müssen lediglich das vorgegebene Strukturschema einhalten und im JSON-

Format an die definierte Schnittstelle gesendet werden. Die Intention hinter der konzeptionellen Idee der spezifischen Transformer ist es, die Anbindung neuer Datenquellen an das Visualisierungssystem möglichst einfach und flexibel zu gestalten, da der Nutzen des Systems vom Umfang der Datenbasis abhängt.

Es konnte in dieser Arbeit durch die exemplarische Entwicklung eines internen und eines externen spezifischen Transformers der Nachweis zur Umsetzbarkeit der föderalisierten Transformation erbracht werden.

Die Implementierung einer Push-basierten WebSocket-Kommunikation zum Empfang der transformierten Daten hat sich als vorteilhaft erwiesen. Sie verfügt über eine hohe Performanz, eine geringe Datenlatenz und einen geringen Konfigurationsaufwand.

Evaluation der JavaScript-Anwendung

Die Single-Page-Anwendung zur Visualisierung der Daten konnte mithilfe des AngularJS-Frameworks erfolgreich als JavaScript-Anwendung implementiert werden. AngularJS unterstützt die Entwicklung benutzerfreundlicher Webanwendungen, welche die Nutzererfahrung einer nativen Desktopanwendung bieten. Dirty-Checking ermöglicht die umgehende Reaktion auf Benutzereingaben, wodurch die Anwendung dynamisch wirkt. Die Benutzerfreundlichkeit und die Qualität der Nutzererfahrung, die durch die Verwendung von AngularJS erreicht werden, unterstreichen die Eignung des Frameworks für die Entwicklung des Visualisierungssystems.

Aus technischer Sicht erwies sich das AngularJS-Framework ebenfalls als geeinget. AngularJS unterstützt die Kommunikation mit einem RESTful API, wodurch eine REST-basierte Kommunikation zwischen der SPA und dem Visualisierungssystem erfolgreich implementiert werden konnte. Darüber hinaus kann AngularJS als JavaScript-basiertes Framework die im JSON-Format zur Verfügung gestellten Daten direkt nutzen. Die Anforderungen an die SPA konnten durch den Einsatz des AngularJS-Frameworks vollständig erfüllt werden.

Die SPA stellt dem Systemanwender die Funktionalitäten der Java-EE-Anwendung auf einer übersichtlich und benutzerfreundlich gestalteten Oberfläche bereit. Es konnten alle geforderten Funktionalitäten realisiert werden. Die statischen Daten aus den Projekten Streetlife und goBerlin werden anhand ihres Geobezugs korrekt mithilfe der definierten Geometrien und Farben auf der Karte verortet und visualisiert, wobei ihre statischen Informationen abruf-

bar sind. Die Visualisierung vieler Geoobjekte findet mit einer leichten zeitlichen Verzögerung statt, da die angeforderten Daten gebündelt zur Verfügung stellt werden. Hier besteht Optimierungsbedarf. In der Implementierung wurde dieses Problem aufgegriffen und ein Filtermechanismus konzipiert, der jedoch aus zeitlichen Gründen noch nicht implementiert wurde. Dieser beschränkt die Anforderung der Daten auf diejenigen, welche im aktuellen Kartenausschnitt lokalisiert sind und verkürzt somit die Ladezeit.

Die dynamischen Daten eines selektierten Geoobjekts werden korrekt geladen und gefiltert, wodurch sie für die Datenvisualisierung und -auswertung auswählbar sind. Die Diagrammerstellung zur Visualisierung der dynamischen Daten wurde zunächst mit Chart.js umgesetzt. Das Tool stellte sich jedoch angesichts der inkorrekten Abbildung des zeitlichen Verlaufs als ungeeignet für die generische Visualisierung von zeitbezogenen Daten heraus. Da in dieser Arbeit die funktionalen Aspekte im Vordergrund standen, wurde zugunsten der Funktionalität auf die gestalterischen Vorteile von Chart.js verzichtet und stattdessen D3.js zur Diagrammerstellung verwendet. Die mit D3.js erstellen Diagramme sind korrekt skaliert, jedoch nicht responsiv. Diese Lösung genügt den derzeitigen funktionalen Anforderungen an das Visualisierungssystem, sollte jedoch zukünftig optimiert werden. Mögliche Lösungsansätze sind die Erstellung eines benutzerdefinierten Diagramms oder die Verwendung eines anderen responsiven Visualisierungstools. Im zeitlichen Rahmen der Arbeit konnte dies nicht mehr umgesetzt werden.

Um dem Systembenutzer die Auswertung der dynamischen Daten für bestimmte Fragestellungen nach Minuten, Stunden, Tagen und Monaten zu ermöglichen, wurde eine Aggregationsfunktion für die Zeiteinheit implementiert. Dabei werden auch die Darstellungen der Werte und der Zeitachse des Diagramms angepasst, um die Auswertung visuell zu unterstützen.

Neben der Visualisierung stellt die funktionsgestützte Auswertung der Daten einen Schwerpunkt dieser Arbeit dar. Die implementierten elementaren Auswertungsfunktionen zeigen zuverlässig die richtigen Werte für das Maximum, das Minimum und den Durchschnittswert an. Ferner wurde der Plug-In-Mechanismus zur Anbindung spezifischer Auswertungsfunktionen entworfen und erfolgreich exemplarisch implementiert. Er ermöglicht die einfache Integration nutzerseitig programmierter Auswertungsfunktionen.

Die Usability der SPA wird durch einen Guide erhöht, der eine Kurzanleitung zur Datenauswertung gibt.

Fazit

Die Aufgabenstellung der Arbeit konnte im Zuge des Enwurfs und der Entwicklung des Visualisierungssystems vollständig erfüllt werden. Die Anforderungen an diese Arbeit konnten durch die Konzeption und Entwicklung eines universellen verlustfreien Transformationsmechanismus und der spezifischen Tansformationskomponenten sogar übertroffen werden. Insbesondere ist herauszustellen, dass die Anwendung des Transformationsmechanismus nicht auf die Datentransformation im Rahmen des Visualisierungssystems beschränkt ist, sondern ein universell anwendbares Konzept darstellt.

Die spezifischen Transformatoren ermöglichen die volle Ausschöpfung des Potenzials des Visualisierungssystems als Werkzeug zur Unterstützung der informationsbasierten Verkehrs- und Stadtplanung. Als relevant erachtete Daten beliebiger Formate und Herkunft können mit geringem Aufwand angebunden werden, um die Informationsgrundlage zu erweitern. Es existieren konzeptionell keinerlei Beschränkungen hinsichtlich des Datenformats, der Struktur, der Zugriffstechnologie, der Art der zu betrachtenden statischen Objekte und der Art der beobachteten Ereignisse, wodurch eine hohe Flexibilität gegeben ist.

Die konzipierten und realisierten Funktionalitäten gewährleisten eine umfassende Analyse von Daten mit Orts- und Zeitbezug unabhängig von ihrer Herkunft oder ihres Formats. Die funktionsgestützte Analyse ist unbegrenzt um anwendungsspezifische Auswertungsfunktionen erweiterbar, wodurch Verkehrsdaten auf eine konkrete Fragestellung bezogen ausgewertet werden können. Das in dieser Arbeit entworfene und entwickelte Visualisierungssystem ist daher in besonderem Maße zur Analyse verkehrsbezogener Daten geeignet und stellt ein wertvolles Informationswerkzeug für eine moderne und nachhaltige Verkehrs- und Stadtplanung dar.

Das Visualisierungssystem wurde durch Partner des Streetlife-Projekts am Fraunhofer FOKUS getestet. Die ersten Reaktionen waren überaus positiv. Das Visualisierungssystem erfüllt alle konzipierten funktionalen und nichtfunktionalen Anforderungen, läuft stabil und funktioniert zuverlässig. Die Entscheidung über die finale Integration in die Streetlife-Plattform steht jedoch noch aus, da das Visualisierungssystem noch den Entscheidungsträgern des Streetlife-Projekts präsentiert werden muss.

8 Ausblick

Das im Rahmen dieser Arbeit entwickelte Visualisierungssystem stellt eine erste prototypische Implementation der Anforderungen dar. Zukünftig soll es weiterentwickelt und optimiert werden, um eine größere Bandbreite an Funktionalitäten zu bieten und die Verkehrsplanung effektiver zu unterstützten. Das Visualisierungssystem wurde derart konzipiert und implementiert, dass eine hohe Flexibilität gegeben ist und zahlreiche Weiterentwicklungsmöglichkeiten umsetzbar sind, um es an zukünftige Anforderungen anpassen zu können. Abschließend sollen einige Ideen zur Potentialausschöpfung des Visualisierungssystems diskutiert werden.

Um die effektive Nutzung des Visualisierungssystems zu gewährleisten, soll der Konfigurationsaufwand minimiert werden, um die Fokussierung auf die Datenanalyse zu erlauben. Ein Ziel der Optimierungen ist daher eine weitere Reduktion des Konfigurationsaufwands durch Automatisierung, um die Anbindung neuer Datenquellen effizienter zu gestalten, und die Einführung einer grafischen Benutzerschnittstelle zur benutzerfreundlicheren Integration spezifischer Auswertungsfunktionen.

Der Umfang der Möglichkeiten zur Datenanalyse und -visualisierung kann erheblich erweitert werden. Durch eine Weiterentwicklung der Filtermechanismen können Geoobjekte unterschiedlicher Kategorien gemeinsam auf der Karte visualisiert werden. Ein Mechanismus zur aggregierten Auswertung mehrerer Geoobjekte bietet die Möglichkeit einer globalen Analyse des Datenbestands. Von besonderem Interesse ist eine vergleichende Auswertungsmöglichkeit mehrerer Geoobjekte, Ereignisse oder Ereigniseigenschaften. Die Nebeneinanderstellung der Ergebnisse ermöglicht einen direkten Vergleich der Daten.

Zur Verbesserung der Usability kann eine responsive und interaktive Visualisierung der dynamischen Daten beitragen. Denkbar sind hier vor allem Möglichkeiten zur Darstellungsanpassung, die Auswahl einer alternativen Diagrammart, die Änderung der Achsenskalierung oder die Option zur vergrößerten Darstellung des Diagramms.

Im Rahmen dieser Arbeit wurden nur ratioskalierte dynamische Daten visualisiert und ausgewertet. Unter Verwendung anderer Diagrammarten kann die Visualisierung von Daten anderer Skalenniveaus realisiert werden. Die Analyse nicht-rationskalierter Daten ist jedoch nicht auf die Visualisierung in Dia-

grammen beschränkt, sondern durch die Integration benutzerentwickelter anwendungsspezifischer Auswertungsfunktionen erweiterbar.

Nützlich ist ferner die Möglichkeit zur Erstellung von Benutzerkonten, in denen die Ergebnisse einer Datenauswertung gespeichert werden können, um zu einem späteren Zeitpunkt auf sie zurückgreifen zu können. Zusätzlich ist eine Exportfunktion der Ergebnisse wünschenswert.

Durch die implementierte REST-Schnittstelle ist die Java-EE-Anwendung prinzipiell von anderen Anwendungen nutzbar. Sie ermöglicht zum Beispiel die zukünftige Expansion auf mobile Endgeräte. Dies erhöht die Benutzerfreundlichkeit und Flexibilität für den Anwender, der dadurch Auswertungen unterwegs oder vor Ort vornehmen kann, um sich einen Eindruck der örtlichen Gegebenheiten zu verschaffen.

Das hier entwickelte Visualisierungssystem verfügt über eine solide, skalierbare Architektur und implementiert hochentwickelte moderne Technologien, da die Konzeption und Entwicklung stets unter Berücksichtigung künftiger, visionärer Weiterentwicklungsmöglichkeiten stattfanden. Die vorliegende Version des Systems stellt ein zukunftsorientiertes, stabiles Fundament dar, auf welchem mit geringem Aufwand weitere Funktionalitäten aufgebaut werden können. Obwohl das System bereits zum gegenwärtigen Zeitpunkt einen bedeutungsvollen Beitrag zur Unterstützung der Verkehrs- und Stadtplanung leisten kann, verdeutlichen die zahlreichen Ideen zur Optimierung und Erweiterung des Funktionsumfangs des Visualisierungssystems nachdrücklich dessen außerordentliches Potential zur nachhaltigen Gestaltung der Städte der Zukunft.

Literaturverzeichnis

Abts, D. (2015). *Masterkurs Client/Server-Programmierung mit Java* (4. Auflage.). Wiesbaden: Springer Fachmedien Wiesbaden. doi:10.1007/978-3-658-09921-3

Akremi, L., Baur, N., & Fromm, S. (2011). *Datenanalyse mit SPSS für Fortgeschrittene 1 - Datenaufbereitung und uni- und bivariate Statistik* (3. Auflage.). Wiesbaden: VS Verlag für Sozialwissenschaften | Springer. doi:10.1007/978-3-531-93041-1

Andrae, C., Fitzke, J., & Zipf, A. (2009). *OpenGIS essentials - Die Geo-Standards von OGC und ISO im Überblick* (1, Auflage.). Heidelberg: Herbert Wichmann Verlag.

Balzert, H. (2011). *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb* (3. Auflage.). Heidelberg: Spektrum Akademischer Verlag. doi:10.1007/978-3-8274-2246-0

Bartelme, N. (2005). Einführung. In *Geoinformatik* (4. Auflage., pp. 1–42). Berlin Heidelberg: Springer. doi:10.1007/b138747

Bauer, A., & Günzel, H. (2013). *Data Warehouse Systeme - Architektur, Entwicklung, Anwendung* (4. Auflage.). dpunkt Verlag. Retrieved from http://www.dpunkt.de/buecher/4507/-data-warehouse-systeme.html

Befeld, M., & Klein, S. (2010). *Technische Unterschiede zwischen Glassfish, JBOSS, WebLogic und Websphere*. Fachhochschule für Ökonomie & Management Hamburg.

Bower. (2015). Bower. Retrieved July 25, 2015, from http://bower.io

Brink, S. (2015). *AngularJS Online-Buch*. Essen. Retrieved from https://angularjs.de/buch/gesamt#grundlagen

Brinkhoff, T. (2008). *Geodatenbanksysteme in Theorie und Praxis - Einführung in objektrelationale Geodatenbanken unter besonderer Berücksichtigung von Oracle Spatial* (2. Auflage.). Heidelberg: Wichmann-Fachmedien.

Burke, B. (2014). *RESTful Java with JAX-RS 2.0 - Designing and Developing Distributed Web Services* (3. Auflage.). Sebastopol: O'Reilly.

Card, S. K., Mackinlay, J. D., & Schneidermann, B. (1999). *Readings in Information Visualization: Using Vision to think* (1. Auflage.). San Diego: Academic Press.

chart.js. (2015). Chart.js Documentation. Retrieved August 13, 2015, from http://www.chartjs.org/docs/

Conrad, S. (1997). *Föderierte Datenbanksysteme* (1. Auflage.). Heidelberg: Springer-Verlag. doi:10.1007/978-3-642-59028-3

Davenport, R. J. (2008). *ETL vs ELT - A Subjective View*. *Insource Commercial Aspects of BI discussion*. Berkshire. Retrieved from http://www.dataacademy.com/files/ETL-vs-ELT-White-Paper.pdf

Dincer, A., & Uraz, B. (2013). *Google Maps JavaScript API Cookbook* (1. Auflage.). Birmingham: Packt Publishing.

Dunkel, J., & Holitschke, A. (2003). *Softwarearchitektur für die Praxis*. Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-55552-7

Farkisch, K. (2011). *Data-Warehouse-Systeme kompakt*. Heidelberg Dordrecht London New York: Springer.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. University of California. Retrieved from http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Frigeri, D., Valetto, G., Wollina, M., Kelpin, R., Pezzato, T., Thiemer, F., … Marconi, A. (2014). *WP6 – City Pilot Planning and Evaluation – D6.1 Specification of city pilots for the first STREETLIFE operation and evaluation*.

GoldenGate Software Inc. (2009). *Going Real-Time for Data Warehousing and Operational BI - Enabling Real-Time Data Integration*. San Francisco. Retrieved from http://media.techtarget.com/Syndication/APP_DEVELOPMENT/GoldenGate_Software_GoingRealTime.pdf

Goll, J. (2014). *Architektur- und Entwurfsmuster der Softwaretechnik* (2. Auflage.). Wiesbaden: Springer Fachmedien Wiesbaden. doi:10.1007/978-3-658-05532-5

Goncalves, A. (2013). *Beginning Java EE 7* (1. Auflage.). Berkeley, CA: Apress. doi:10.1007/978-1-4302-4627-5

Google Inc. (2015a). Google Developers - Google Charts. Retrieved July 25, 2015, from https://developers.google.com/chart/

Google Inc. (2015b). Google Maps API Geocoding. Retrieved August 21, 2015, from https://developers.google.com/maps/documentation/geocoding/intro

Google Inc. (2015c). Google Maps FAQs. Retrieved August 23, 2015, from https://developers.google.com/maps/faq#usage_pricing

Grude, U. (2005). *Java ist eine Sprache* (1. Auflage.). Wiesbaden: Vieweg+Teubner Verlag. doi:10.1007/978-3-322-80263-7

Grunt. (2015). Grunt. Retrieved July 25, 2015, from http://gruntjs.com/getting-started

Hartmann, B., Teusch, A., & Wolf, M. (2013). Spezifikation von funktionalen und nichtfunktionalen Systemanforderungen auf Basis von Geschäftsprozessmodellen. In *11th International Conference on Wirtschaftsinformatik* (pp. 1293–1307). Leipzig.

Inmon, W. H. (2005). *Building the Data Warehouse* (4. Auflage.). New York: John Wiley & Sons, Inc.

ISO. ISO 8601:2004 - Representation of dates and times (2004). Retrieved from https://www.iso.org/obp/ui/#iso:std:iso:8601:ed-3:v1:en

ISO/TC 211. ISO 19107:2003 - Geographic information — Spatial schema (2003). Retrieved from https://www.iso.org/obp/ui/#iso:std:iso:19107:ed-1:v1:en

ISO/TC 211. ISO 19125-1:2004 - Geographic information — Simple feature access — Part 1: Common architecture (2004). Retrieved from https://www.iso.org/obp/ui/#iso:std:iso:19125:-1:ed-1:v2:en

ISO/TC 211. ISO 19125-2:2004 - Geographic information — Simple feature access — Part 2: SQL option (2004). Retrieved from https://www.iso.org/obp/ui/#iso:std:iso:19125:-2:ed-1:v1:en

John Joseph, R. (2015). Single Page Application and Canvas Drawing. *International Journal of Web & Semantic Technology*, *6*(1), 29–37. doi:10.5121/ijwest.2015.6103

Joyent Inc. (2015). Node.js. Retrieved July 25, 2015, from https://nodejs.org/about/

Kakish, K., & Kraft, T. (2012). ETL Evolution for Real-Time Data Warehousing. *Proceedings of the Conference on Information Systems Applied Research*, 1–12. Retrieved from www.aitp-edsig.org

Kelonye, M. (2014). *Mastering Ember.js*. Packt Publishing.

Kimball, R., & Caserta, J. (2004). *The Data Warehouse ETL Toolkit*. Indianapolis: Wiley Publishing, Inc.

Kimball, R., & Ross, M. (2002). *The Data Warehouse Toolkit - The Complete Guide to Dimensional Modeling* (2. Auflage.). New York: Wiley Computer Publishing.

Kloster, T., & Martin, O. (2004). *Vergleich SOAP und REST Komponentenorientierte Softwareentwicklung und Hypermedia*. Dortmund. Retrieved from https://www.edvsz.hs-osnabrueck.de/fileadmin/compass/Lehrveranstaltungen/SoSe_04/Soap-Rest_Ausarbeitung.pdf

Kurt, B., Ihlenburg, P., & Ott, R. (2007). *Beschreibende Statistik* (2. Auflage.). Rinteln: Merkur-Verlag.

Lerner, A., & Powell, V. (2014). *D3 on AngularJS - Create Dynamic Visualizations with AngularJS*. Fullstack.io. Retrieved from https://leanpub.com/d3angularjs

Leser, U., & Naumann, F. (2007). *Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen*. Heidelberg: dpunkt-Verlag.

Mertens, P., Bodendorf, F., König, W., Picot, A., Schumann, M., & Hess, T. (2012). Daten, Information und Wissen. In *Grundzüge der Wirtschaftsinformatik SE - 3* (pp. 37–63). Springer Berlin Heidelberg.

Mikowski, M., & Powell, J. (2014). *Single Page Web Applications: JavaScript End-to-End*. Shelter Island: Manning.

Oestreich, M., & Romberg, O. (2014). *Keine Panik vor Statistik! Keine Panik vor Statistik!* (5. Auflage.). Wiesbaden: Springer Fachmedien Wiesbaden. doi:10.1007/978-3-658-04605-7

Osmani, A. (2013). *Developing Backbone.js Applications* (1. Auflage.). Sebastopol: O'Reilly.

Pilgrim, P. A. (2013). *Java EE 7 Developer Handbook* (1. Auflage.). Birmingham: Packt Publishing.

Ramm, F., & Topf, J. (2010). *OpenStreetMap Die freie Weltkarte nutzen und mitgestalten* (3. Auflage.). Lehmanns Media.

Red Hat Inc. (2013). Red Hat Reveals Plans for its Next Generation Java Application Server Project - WildFly Announced as Community Successor to the JBoss Application Server. Retrieved September 7, 2015, from http://www.redhat.com/en/about/press-releases/red-hat-reveals-plans-for-its-next-generation-java-application-server-project?utm_source=feedly

Richardson, L., & Ruby, S. (2007). *RESTful Webservices*. (M. Loukides, Ed.) (1. Auflage.). Sebastopol: O'Reilly.

Rupp, H. W. (2007). *EJB 3 für Umsteiger* (1. Auflage.). Heidelberg: dpunkt Verlag.

Shokeen, M. (2015). Fancy, Responsive Charts with Chart.js. Retrieved August 13, 2015, from http://www.sitepoint.com/fancy-responsive-charts-with-chart-js/

Simitsis, A., Vassiliadis, P., & Sellis, T. (2005). Optimizing ETL Processes in Data Warehouses. In *21st International Conference on Data Engineering* (pp. 564–575). IEEE. doi:10.1109/ICDE.2005.103

Springer, S. (2013). *Node.js Das umfassende Handbuch* (1. Auflage.). Bonn: Galileo Press.

Stevens, S. . S. (1946). On the theory of scales of measurement. *Science (New York, N.Y.)*, *103*(2684), 677–680. doi:10.1126/science.103.2684.677

Svennerberg, G. (2010). *Beginning Google Maps API 3*. Apress.

Tarasiewicz, P., & Böhm, R. (2014). *AngularJS Eine praktische Einführung in das JavaScript-Framework* (1. Auflage.). Heidelberg: dpunkt Verlag.

The jQuery Foundation. (2015). jQuery. Retrieved September 6, 2015, from http://api.jquery.com

Tho, M. N., & Tjoa, A. M. (2003). Zero-Latency Data Warehousing for Heterogeneous Data Sources and Continuous Data Streams. *Proc. of the Fifth Int. Conf. on Information Integration and Web-Based Applications Services*.

Wang, V., Salim, F., & Moskovits, P. (2013). *The Definitive Guide to HTML5 WebSocket* (1. Auflage.). Berkeley, CA: Apress. doi:10.1007/978-1-4302-4741-8

Weber, R. (2012). *Technologie von Unternehmenssoftware*. Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-24423-0

Weil, D. (2012). *Java EE 6 Enterprise-Anwendungsentwicklung leicht gemacht*. Frankfurt am Main: Software & Support Media GmbH.

Wilde, M. (2015). Die Re-Organisation der Verkehrssysteme. *Standort*, *39*(1), 22–25. doi:10.1007/s00548-015-0364-2

WildFly. (2015). WildFly. Retrieved July 25, 2015, from http://wildfly.org/about/

Zillgens, C. (2013). *Responsive Webdesign* (1. Auflage.). München: Carl Hanser Verlag.

Zwattendorfer, B. (2013). *Analyse SOAP vs . REST*. Graz.

Glossar

Dependency Injection ist ein Konzept in der objektorientierten Programmierung, bei dem Abhängigkeiten von einer externen Instanz zugewiesen werden.

Metadaten sind Informationen zu Daten, welche diese beschreiben.

Mockup ist eine Visualisierungsmodell für die Konzeption der Benutzeroberfläche einer Anwendung. Es visualisiert den Entwurf des Layouts, des Formats und des Inhalts.

MVC steht für Model-View Controller. Es handelt sich um ein Entwurfsmuster für die Realisierung von GUI-Komponenten mithilfe objektorientierter Programmierung.

URI Uniform Resource Identifier (URI) ist eine eindeutige Kennzeichnungsform für alle im globalen Web vorhandenen Ressourcen und Objekte, einschließlich des benutzten Protokolls und Dienstes.

URL Uniform Resource Locator (URL) ist eine eindeutige Adressierungsform für Internetadressen bestehend aus dem Protokoll und dem Domainnamen.

Use Case ist eine Methodik in der Systemanalyse zur Identifizierung, Spezifizierung und Organisation der Systemanforderungen. Es beinhaltet alle relevanten Interaktionsmöglichkeiten des Nutzers mit dem System.

A. Abbildungen

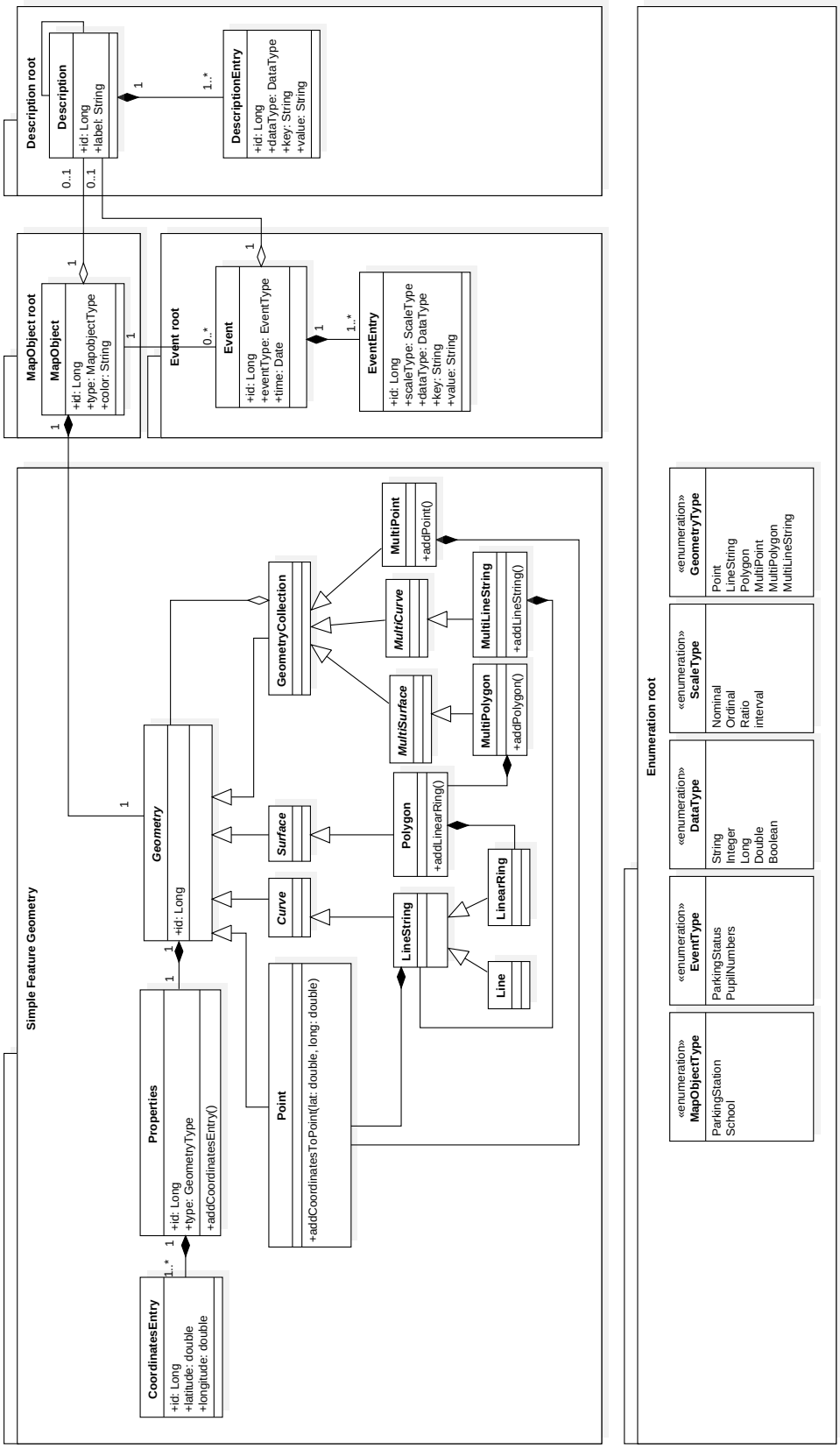

Abbildung 23: Klassendiagramm des Datenmodells

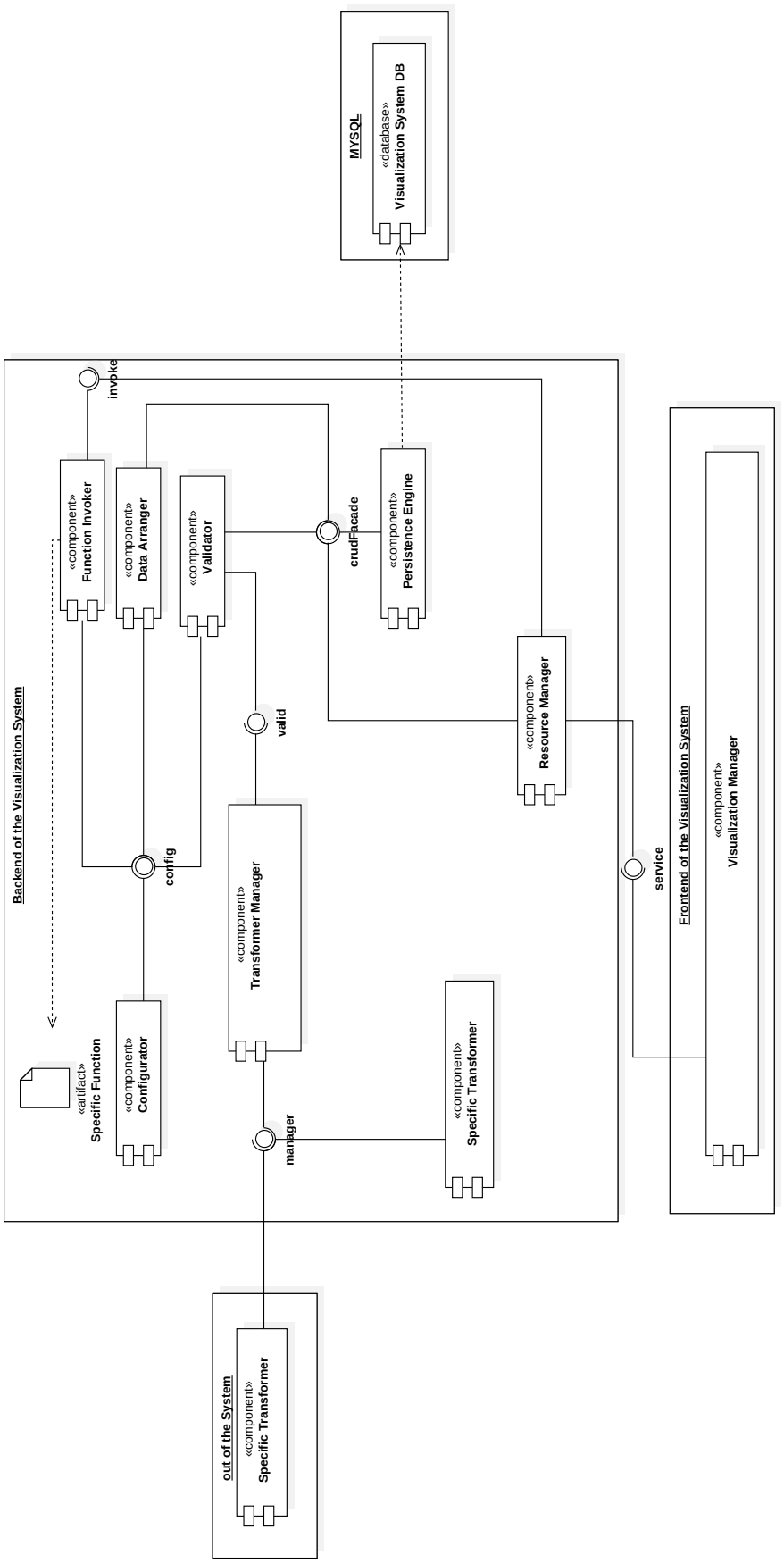

Abbildung 24: Komponentendiagramm des Visualisierungssystems

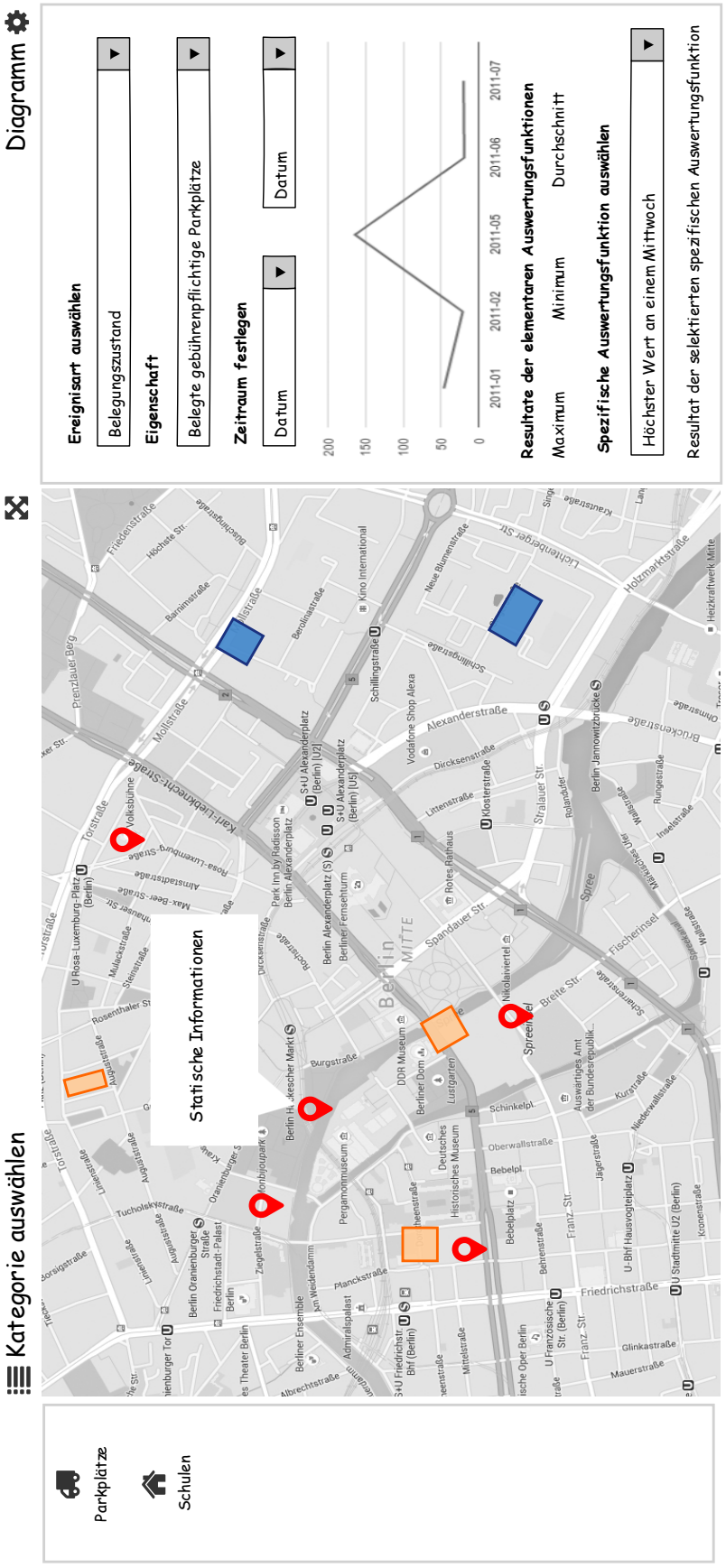

Abbildung 25: Mockup der Benutzeroberfläche

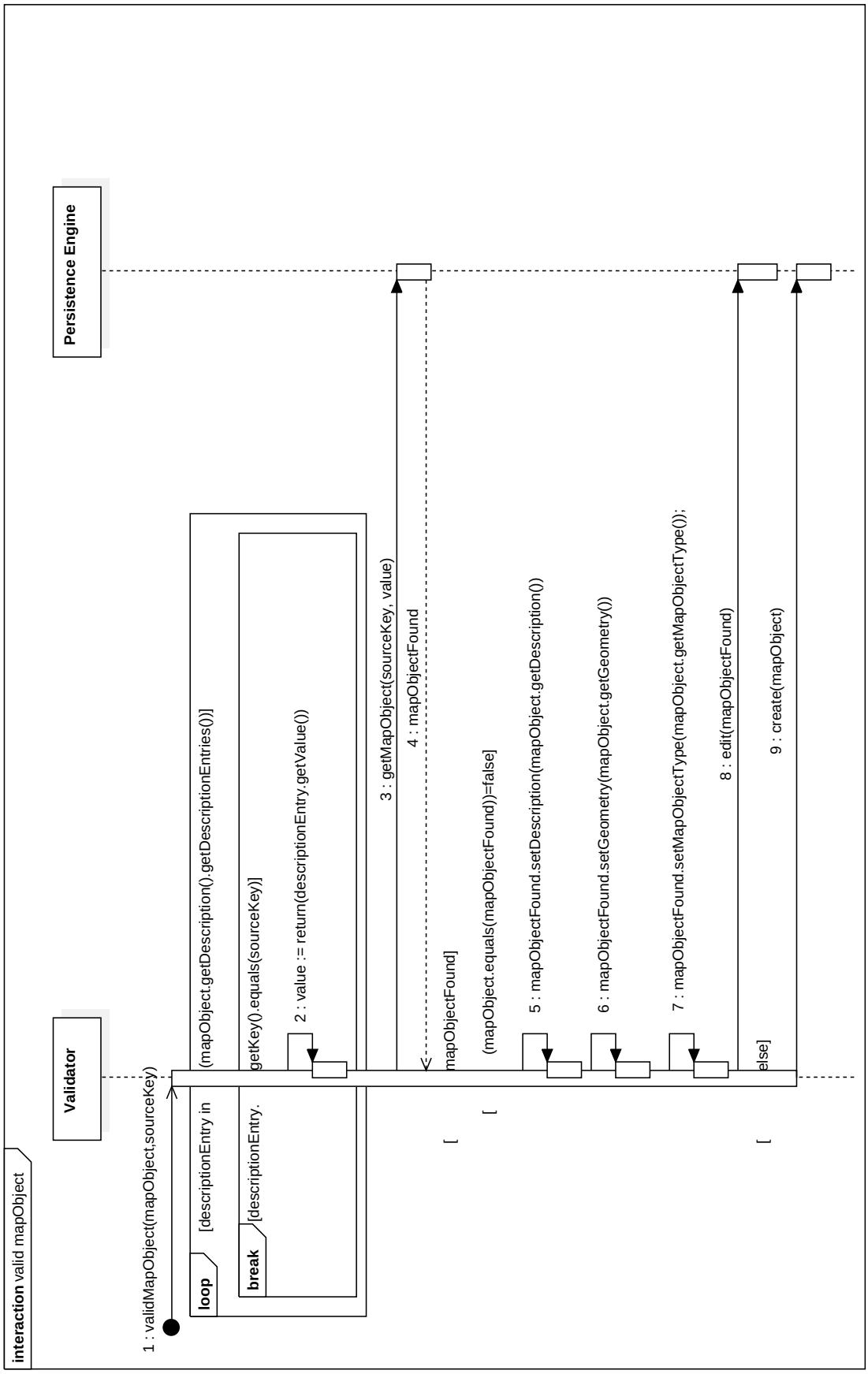

Abbildung 26: Sequenzdiagramm der Validierung eines *MapObjects*

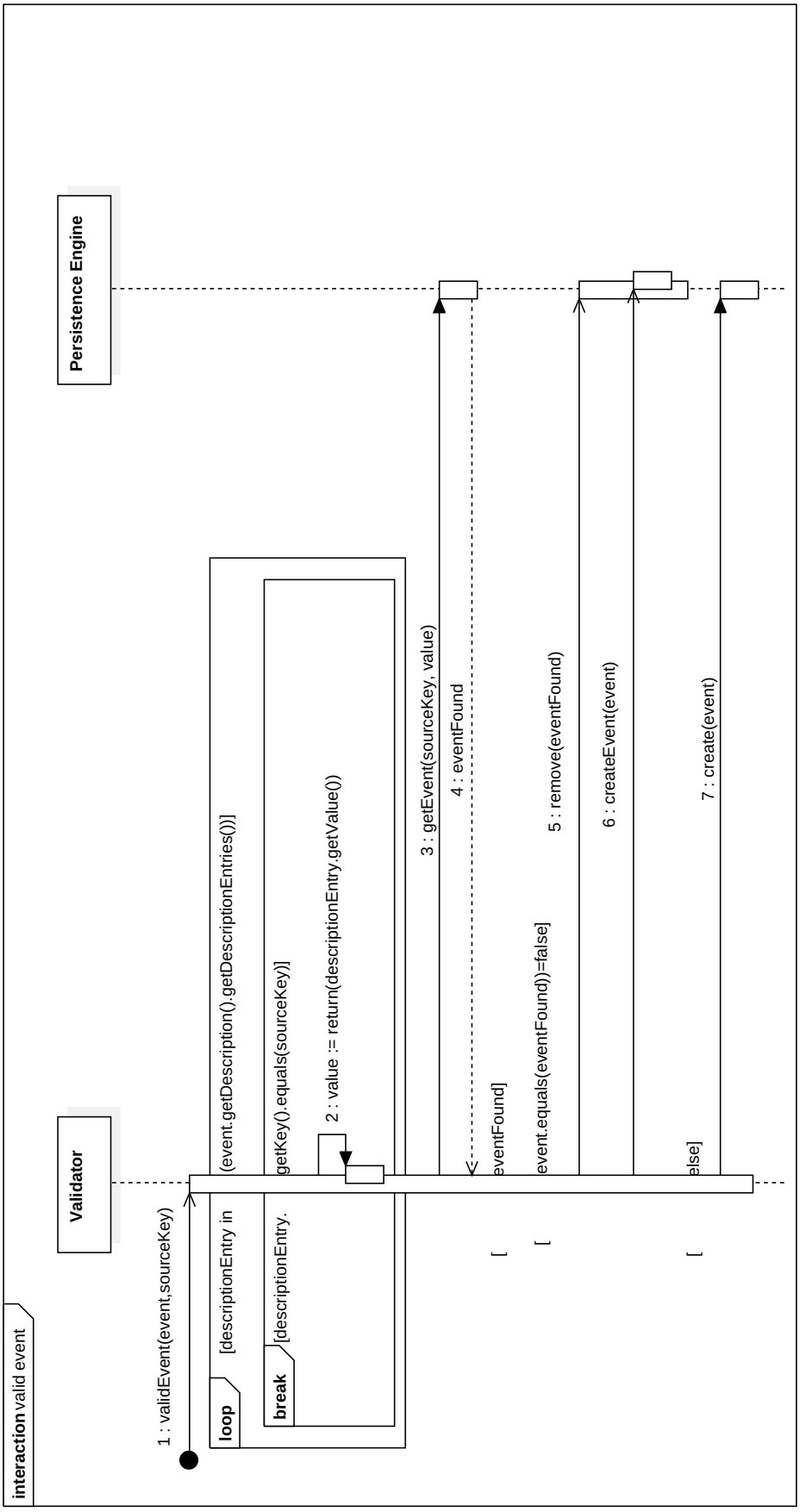

Abbildung 27: Sequenzdiagramm zur Validierung eines *Events*

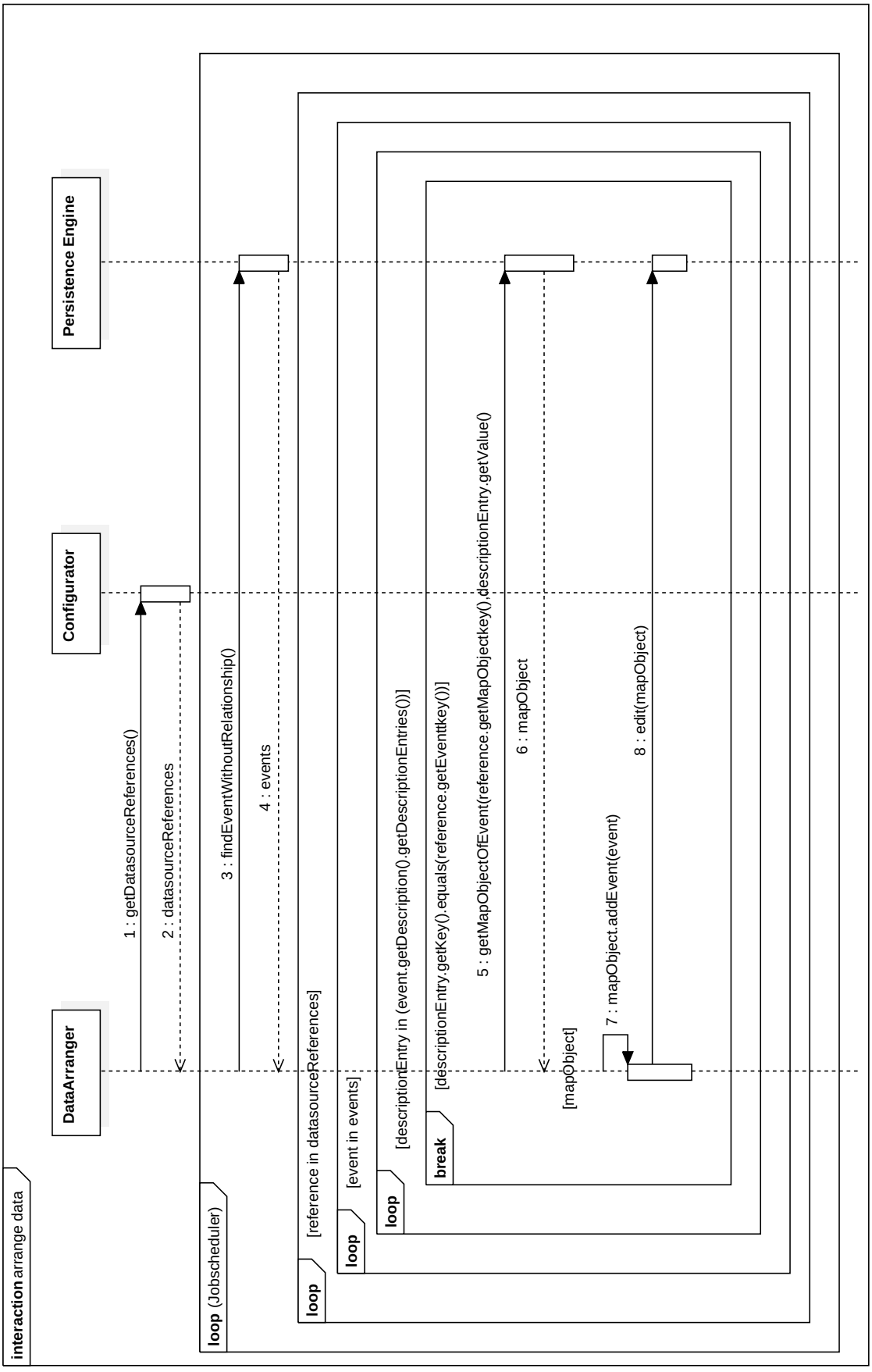

Abbildung 28: Sequenzdiagramm zur Rekonstruktion von Beziehungen

B. Beispielhafte Transformation

Hier soll das entwickelte Konzept zur Datentransformation (Vgl. Kapitel 4.2.1) beispielhaft an den bereitgestellten Datensätzen demonstriert werden. Es wird je ein Beispiel für die Transformation eines statischen und eines dynamischen Objekts gegeben.

B.1 – Fallbeispiel Rovereto

In Tabelle 12 ist die konzeptionelle Transformation eines statischen Objekts aus dem Rovereto-Datensatz gezeigt. Die Struktur des transformierten Objekts zeigt, dass jedes Attribut des statischen Objekts als `Description` abgebildet werden kann.

Der `MapObjectType` und der `GeometryType` werden anhand der Analyse des statischen Objekts festgelegt (Vgl. Kapitel 3.2). In diesem Fall handelt es sich um eine `ParkingStation` vom `GeometryType` "Polygon". Die Koordinaten unter "location" kennzeichnen ein Stützpunkt, nicht die Geometrie, weshalb sie als `Description` abgebildet werden.

Tabelle 12: Konzeptionelle Transformation eines statischen Objekts von Rovereto in der JSON-Repräsentation

Statisches Objekt	Transformiertes Objekt (*MapObject*)

`{`
`"type": "mapobject",`
`"objectID": "Parteli _ 01",`
`"objectType": "ParkingStation",`
`"objectSubtype": "cardblock",`
`"description": "Parkingslot",`
`"location": {`
`"type": "Point",`
`"coordinates": [`
`11.9898989898989,`
`45.897581527701135`
`]`
`},`
`"elements": [`
`{`
`"attribute": {`
`"label": "Microzone",`
`"value": "Parteli _ 01"`
`}`
`},`
`{`
`"attribute": {`
`"label": "Macrozone",`
`"value": "[2] _ STADIO _ B"`
`}`
`},`
`...`
	`{`
	`id: 1,`
	`mapObjectType: "ParkingStation",`
	`geometry: {`
	`id: 1,`
	`properties: {`
	`geometryType: "Polygon",`
	`coordinates: [`
	`{`
	`latitude: 45.897581527701135,`
	`longitude: 11.035346110849889`
	`},`
	`{`
	`latitude: 45.89747633052608,`
	`longitude: 11.036336738033501`
	`},`
	`...`
	`]`
	`}`
	`},`
	`description: {`
	`id: 1,`
	`label: null,`
	`descriptionEntries: [`
	`{`
	`key: "type",`
	`value: "mapobject"`

```
{
  "maparea": {
    "area": {
      "type": "Polygon",
      "coordinates": [
        [
         [
          11.035346110849889,
          45.897581527701135
         ],
         [
          11.036336738033501,
          45.89747633052608
         ]
         ...
         ]
        ]
    },
    "color": {
      "red": 125,
      "green": 125,
      "blue": 125,
      "alpha": 0.5
    }
  }
 }
]
}
```

```
      },
      {
        key: "objectID",
        value: "Parteli _ 01"
      },
      ...
      ],
   subDescription: {
      id: 2,
      label: ‘‘location’’,
      descriptionEntries: [
        {
          key: "type",
          value: "point"
        }
        ...
        ]
      },
   subDescription: {
      id: 3,
      label: ‘‘elements’’,
      subDescription: {
        id: 4,
        label: ‘‘attribute’’,
        descriptionEntries:
[
          {
           key: "Microzone",
           value: "Parteli _ 01"
           }
         ],
      subDescription: {
        id: 5,
        label: attribute,
        descriptionEntries: [
          {
            key: "Macrozone",
            value: "[2] _ STADIO _ B"
           }
         ]
      }
   }
   ...
},
color: "#f266ff"
}
```

In Tabelle 13 wird beispielhaft die konzeptionelle Transformation eines dynamischen Objekts gezeigt. Der EventType und der Zeitstempel stehen im transformierten Objekt, dem *Event*, als Attribute. Alle weiteren Attribute des dynamischen Objekts werden als EventEntries abgebildet. Die Attributswerte werden stets als Zeichenketten abgebildet. Der ursprüngliche Datentyp wird durch Metadaten angegeben.

Tabelle 13: Konzeptionelle Transformation eines dynamischen Objekts von Rovereto in der JSON-Repräsentation

Dynamisches Objekt	Transformiertes Objekt (*Event*)

<pre>{
  "_id": {
    "$oid": "54579094975a3b0c5b2e00f1"
  },
  "_class": "eu.trentorise.smartcampus.
      parcheggiausiliari.model.StreetLog",
  "author": "oreste",
  "time": "2014-11-03T14:26:28.277Z",
  "value": {
    "_id": "street@rovereto@Tolomei _ 01",
    "slotsFree": 10,
    "slotsOccupiedOnFree": 0,
    "polyline":"wpbwGohkbAYkDNCVlDM@"
    ...
    "position": [
      45.89340781713971,
      11.04024396478435
    ],
    "name": "Tolomei _ 01",
    "updateTime": {
      "$numberLong": "1414079018901"
    },
    "version": {
      "$numberLong": "16596"
    }
  }
}</pre>
	<pre>{
	eventType: "ParkingStatus",
	time: "2014-11-03T14:26:28.277Z",
	eventEntries: [
	  {
	   scaleType: "Ratio",
	   dataType: "Integer",
	   key: "slotsFree",
	   value: "10"
	  },
	  {
	   scaleType: "Ratio",
	   dataType: "Integer",
	   key: "slotsOccupiedOnFree",
	   value: "0"
	  },
	  ...
	],
	description: {
	  id: 6,
	  label: value,
	  descriptionEntries: [
	    {
	     key: "_class",
	     value: "eu.trentorise.smartcampus.
	     parcheggiausiliari.model.StreetLog"
	    },
	    {
	     key: "author",
	     value: "oreste"
	    }
	  ],
	  subDescription:{
	  id:7
	  label : _id
	   {
	    key: "_id",
	    value: "54579094975a3b0c5b2e00f1"
	   }
	  }
	  ...
	}
	}</pre>

B.2 – Fallbeispiel goBerlin

Die Datensätze aus dem Projekt goBerlin liegen im XML-Format vor. Der `MapObjectType` muss nach der Analyse der Daten manuell festgelegt werden. Alle Attribute werden als `Descriptions` abgebildet.

Da die Daten keine Koordinaten, sondern nur eine Adresse enthalten, müssen die Koordination der angegebenen Adresse zur Darstellung auf einer Karte mithilfe der Geocoding-Funktion des Google Maps API bestimmt werden. Das statische Objekt enthält keine Farbangabe, weshalb das Attribut color des transformierten Objekts den Wert "null" erhält. Es wird eine Standardfarbe zur Darstellung verwendet.

Tabelle 14: Konzeptionelle Transformation eines statischen Objekts von goBerlin in der XML-/JSON-Repräsentation

Statisches Objekt	Transformiertes Objekt (*MapObject*)

`<row id="1">`
`<schulId>199</schulId>`
`<name>BEST-Sabel-Oberschule</name>`
`<schulart>Gymnasien ( privat )</schulart>`
`<adresse>Lindenstr. 1, 12555 Berlin (Köpenick)</adresse>`
`<homepage>www.best-sabel.de</homepage>`
`<telefon>280360970</telefon>`
`<ortsteil>Köpenick</ortsteil>`
`<plz>12555</plz>`
`</row>`
	`{`
	`id: 2,`
	`mapObjectType: "School",`
	`geometry: {`
	`id: 11,`
	`properties: {`
	`geometryType: "Point",`
	`coordinates: [`
	`{`
	`latitude: 52.4697836,`
	`longitude: 13.3440679`
	`}`
	`]`
	`}`
	`},`
	`description: {`
	`id: 12,`
	`label: null,`
	`descriptionEntries: [`
	`{`
	`key: "ortsteil",`
	`value: "Köpenick"`
	`},`
	`{`
	`key: "name",`
	`value: "1. Gemeinschaftsschule Schöneberg"`
	`},`
	`{`
	`key: "adresse",`
	`value: "Lindenstr. 1, 12555 Berlin (Köpenick)"`
	`},`
	`{`
	`key: "schulart",`
	`value: "Gymnasien (privat)"`
	`},`
	`...`
	`]`
	`},`
	`color: null`
	`}`

In Tabelle 14 ist die konzeptionelle Transformation eines dynamischen Objekts aus dem goBerlin-Projekt dargestellt. Der EventType des Objekts wird nach der Analyse der Daten manuell festgelegt. Der Zeitstempel wird durch Umwandlung des Schuljahrs in das UTC-Zeitformat der ISO 8601 (ISO, 2004) umgewandelt.

Tabelle 15: Konzeptionelle Transformation eines dynamischen Objekts von goBerlin in der XML-/JSON-Repräsentation

Dynamisches Objekt	Transformiertes Objekt (*Event*)

`<row id="12880">`
`<schulId>495</schulId>`
`<schuljahr>2010/11</schuljahr>`
`<jahrgangsstufe>Jahrgangsstufe 03</jahrgangsstufe>`
`<anzahlSchueler>21</anzahlSchueler>`
`<anzahlSchueleri-nen>20</anzahlSchuelerinen>`
`<gesamtAnzahl>41</gesamtAnzahl>`
`</row>`
	`{`
	`  eventType: "PupilNumbers",`
	`  time: "2011-01-01T01:00:00.0Z",`
	`  eventEntries: [`
	`  {`
	`  scaleType: "Ratio",`
	`  dataType: "Integer",`
	`  key: "schulId",`
	`  value: "495"`
	`  },`
	`  {`
	`  scaleType: "Ratio",`
	`  dataType: "Integer",`
	`  key: "anzahlSchueler",`
	`  value: "21"`
	`  },`
	`  ...`
	`  ],`
	`  description: {`
	`  id: 12,`
	`  label: null,`
	`  descriptionEntries: [`
	`  {`
	`  key: "id",`
	`  value: "12880"`
	`  }`
	`  ],`
	`  }`
	`}`