VDV Die Verkehrsunternehmen

VDV-Mitteilung 4016

16/2014

Kundeninformation für Smarte Informationsdienste

Ontologien

Gesamtbearbeitung

Ausschuss für Kundenservice, -information und -dialog (K3)

Gefördert durch:

Bundesministerium für Wirtschaft und Energie

aufgrund eines Beschlusses des Deutschen Bundestages

VDV-Mitteilung 4016 16/2014

Kundeninformation für Smarte Informationsdienste

Ontologien

Gesamtbearbeitung

Ausschuss für Kundenservice, -information und -dialog (K3)

Autorenverzeichnis

Dipl.-Inf. Christine Keller, Dresden
Dipl.-Ing. Berthold Radermacher, Köln
Dipl.-Ing. (FH) Andreas Wehrmann, Köln

Gefördert durch:

Bundesministerium für Wirtschaft und Energie

aufgrund eines Beschlusses des Deutschen Bundestages

Vorwort

Auf Initiative des VDV und gefördert durch das BMWi begann im September 2010 das Forschungs- und Standardisierungsprojekt

Internet Protokoll basierte Kommunikationsdienste im öffentlichen Verkehr (IP-KOM-ÖV).

Das Projekt wird von 14 Partnern aus Industrie, Universitäten und Verkehrsunternehmen getragen. Es dient der Erarbeitung moderner Kommunikationskonzepte für die umfassende und kontinuierliche Fahrgastinformation.

Eine umfassende Fahrgastinformation stellt heutzutage ein entscheidendes Wettbewerbsmerkmal im öffentlichen Personenverkehr dar, nicht nur im Vergleich mit anderen Verkehrsunternehmen, sondern auch im Vergleich zum Individualverkehr.

Bereits heute ist es üblich, dass Verkehrsunternehmen ihre Fahrgäste nicht nur über die geplanten Fahrten informieren, sondern auch Echtzeitinformationen z. B. zu Verspätungen, Störungen oder Fahrtzieländerungen bereitstellen. Diese Informationen werden zum einen über öffentliche Anzeiger bzw. Ansagen in Fahrzeugen oder an Haltestellen allen dort befindlichen Personen zur Verfügung gestellt. Zum anderen lassen sich solche Informationen mit speziellen Applikationen oder über Web-Angebote individuell abfragen.

Bislang ist es aber nicht möglich, Fahrgäste im Öffentlichen Verkehr direkt mit Informationen zu Ihrer persönlich relevanten Fahrt zu versorgen, den Fahrgast also auch im Störungsfall mit Hilfe des öffentlichen Verkehrs auf dem schnellsten Weg zu seinem Ziel zu führen.

Die weit verbreiteten Smartphones und Tablets bieten hierfür vielfältige Möglichkeiten und ermöglichen eine hohe Akzeptanz der Benutzer. Die Informationsübertragung erfolgt dabei IP-basiert und sollte bevorzugt zwischen einem zentralen Informations-Server und dem Kundenendgerät erfolgen. Für den Fall, dass der zentrale Datenserver nicht erreichbar ist, sollte auch eine Kommunikation zwischen Kundenendgerät und Fahrzeug möglich sein.

Das Forschungs- und Standardisierungsprojekt IP-KOM-ÖV arbeitet deshalb an drei Schwerpunkten (vgl. Abbildung 1).

Erster Schwerpunkt (grün in Abbildung 1) ist die Spezifikation eines performanten IP-basierten Kommunikationsprotokolls im Fahrzeug (IBIS-IP). Dabei geht es zum einen darum, den gewachsenen Bedürfnissen der Fahrgastinformation gerecht zu werden und zum anderen um die Definition einer IP-basierten Schnittstelle zur Übertragung der Informationen vom Fahrzeug zum mobilen Kundenendgerät. Hierzu wird der in den achtziger Jahren entwickelte IBIS-Wagenbus aus der VDV 300 auf eine moderne Ethernet-Informationsarchitektur umgesetzt.

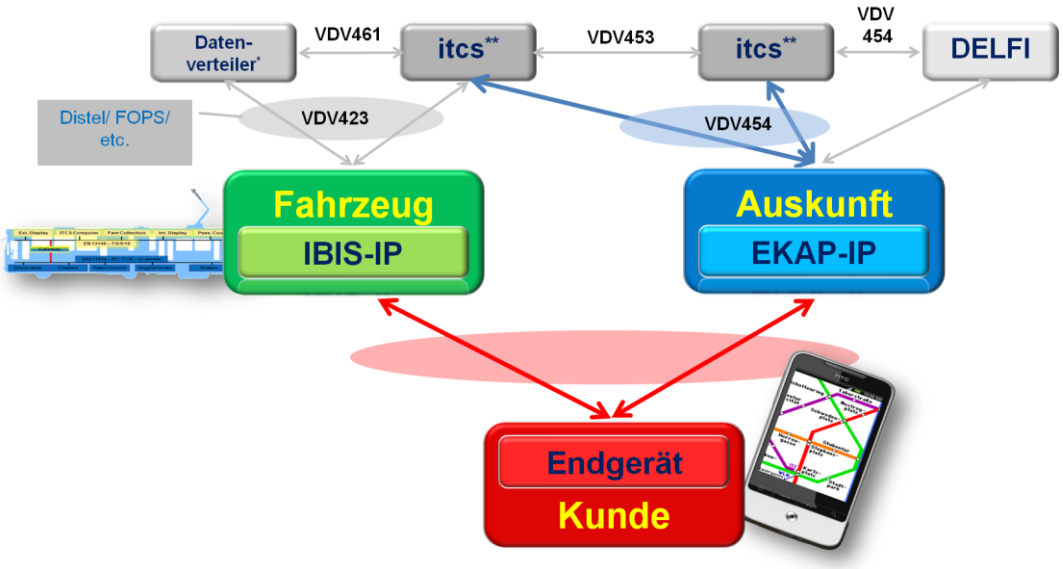

Abbildung 1: Umfeld und Schwerpunkte im Projekt IP-KOM-ÖV

Zweiter Schwerpunkt (rot in Abbildung 1) ist die individuelle Fahrgastinformation unter Verwendung mobiler Geräte des Fahrgasts (Smartphones, Tablet-PC u.ä.) Hierzu wurden im ersten Schritt die Bedürfnisse von Fahrgästen zu individuellen Informationen ermittelt. Im zweiten Schritt wurden einheitliche Schnittstellen zwischen der Echtzeit-Kommunikations- und Auskunftsplattform (EKAP) und den mobilen Kundenendgeräten bzw. zwischen der EKAP und den Hintergrundsystemen entwickelt. Hierbei werden ausschließlich die Datenmodellierungen und Architekturen erforscht und spezifiziert. Aufbauend auf diesen Datenmodellierungen werden semantische Modelle erarbeitet, die helfen, die Fahrgastinformationsdaten für Kommunikationsdienste auf Basis von innovativen Technologien des Semantic Web zur Verfügung zu stellen. Die Entwicklung einer durch Fahrgäste nutzbaren Applikation für mobile Endgeräte war ausdrücklich nicht vorgesehen.

Dritter Schwerpunkt (blau in Abbildung 1) ist die Definition und Schaffung einer Echtzeit-Kommunikations- und Auskunftsplattform (EKAP). Die EKAP bündelt Informationen von ITCS- und anderen Auskunfts- und Informationssystemen und stellt die Vielzahl an Informationen über geeignete Schnittstellen den Applikationen auf den Kundenendgeräten zur Verfügung. Diese Plattform ermöglicht es, Applikationen auf den mobilen Kundenendgeräten, und damit letztendlich Kunden, dynamisch mit individuellen Störungsmeldungen versorgen zu können.

Neben den Forschungsarbeiten ist die vorliegende Standardisierung der Ergebnisse ein wesentliches Ziel des Projektes um eine nachhaltige Nutzung zu gewährleisten.

Darüber hinaus wird die Praxistauglichkeit dieses neuen Standards in Labor- und Feldtests verifiziert.

Inhaltsverzeichnis

Abkürzungen und Begriffe

Es gelten die in (VDV-Schrift 430, 2013), (VDV-Schrift 431, Teil 1, 2013) und in (VDV-Schrift 431 Teil 2, 2013) angegebenen Abkürzungen. Folgende Abkürzungen werden darüber hinaus in diesem Dokument, gemäß der angegebenen Definition, genutzt:

Abkürzung	Beschreibung
API	Application Programming Interface - Programmierschnittstelle
OWL	Web Ontology Language, Sprache zur Beschreibung von Ontologien, Standard des W3C
RDF	Resource Description Framework, ein Standard des W3C zur Repräsentation von Metadaten.
RDFS	Resource Description Framework Schema, Sprache zur Beschreibung von Vokabularen für RDF, W3C Standard
RDF/S	Beschreibt RDF und RDFS als gemeinsam genutzten Technologieverbund.
SPARQL	SPARQL Protocol and Query Language, Anfragesprache für RDF Daten, W3C Standard
TRIAS	Travellers‘ Realtime Information and Advisory Standard (Reisenden Echtzeit-Informations- und -Auskunftssystem)
TPEG (PTI)	Von der Transport Protocol Experts Group definierter Standard für „Public Transport Information“ (PTI)
URI	Uniform Resource Identifier, Identifikatoren nach einem Schema aus RFC: 3986 des W3C zur eindeutigen Identifikation von Ressourcen im World Wide Web.
W3C	World Wide Web Consortium, Gremium zur Standardisierung von Technologien für das World Wide Web
XML	Extensible Markup Language, Auszeichnungssprache für den textbasierten, plattformunabhängigen Datenaustausch zwischen Rechnern, durch das W3C spezifiziert.

Es gelten die Begriffe, die in (VDV-Schrift 430, 2013) und (VDV-Schrift 431, Teil 1, 2013) festgelegt wurden. Folgende Tabelle definiert die darüber hinaus Begriffe, die zusätzlich in diesem Dokument verwendet werden.

Begriff	Beschreibung
Aussage	Eine Aussage in RDF ist von der Form „Subjekt Prädikat Objekt“ und beschreibt Wissen über das Subjekt der Aussage.
DatatypeProperty	In RDF/S und OWL verwendete Property, die ein Konzept mit einem Literal (Ausprägung eines Basis-Datentyps) verknüpft (Beispiel: „Wurde veröffentlicht“).
Datenbasis	Menge von Aussagen über Ressourcen, auch *Wissensbasis* genannt.
DBpedia	DBpedia ist ein Projekt, das die Daten der Wikipedia strukturiert als RDF Daten zur Verfügung stellt (http://de.dbpedia.org/).
Domäne	Fachgebiet, das eine Ontologie beschreibt und in dem sie angewandt wird.

Begriff	Beschreibung
Entität	Als Entität (auch Informationsobjekt genannt, englisch entity) wird in der Datenmodellierung ein eindeutig zu bestimmendes Objekt bezeichnet, über das Informationen gespeichert oder verarbeitet werden sollen.
Instanz	Konkrete Ausprägung einer Klasse (Beispiel: Klasse „Film“ und Instanz „Das Fenster zum Hof“).
Instanzgraph	Verknüpfung mehrerer Aussagen über Instanzen zu einem RDF-Graph.
Kardinalität	Die Kardinalität einer Menge oder Klasse ist die Anzahl der Elemente dieser Menge oder Klasse.
Klasse	Im Rahmen dieser Mitteilung eine Klasse nach Definition von RDFS und OWL. Modelliert ein abstraktes Konzept, dessen konkrete Ausprägungen Instanzen sind (Beispiel: Klasse „Film“ und Instanz „Das Fenster zum Hof“).
Konzepte	Ein Konzept ist eine abstrakte Objektbeschreibung, auch Klasse genannt.
LinkedGeoData	Bei LinkedGeoData handelt es sich um ein Projekt, das die Daten des Open Street Map-Projektes als RDF Daten zur Verfügung stellt (http://linkedgeodata.org).
Literal	Literale repräsentieren feste Datenwerte eines Basis-Datentyps.
Namespace/Namensraum	Gruppiert Benennungen (z. B. in Ontologien). Bezeichner sind innerhalb des Namensraums eindeutig. In anderen Namensräumen können sie jedoch neu verwendet werden. Der Namensraum wird den Bezeichnern, die ihm zugeordnet sind, vorangestellt, so dass die vollständigen Bezeichner eindeutig bleiben.
Object (Objekt)	Ein Objekt ist der Wert einer Eigenschaft (Prädikat) in einer Aussage nach dem Muster „Subjekt Prädikat Objekt“.
ObjectProperty	In RDF/S und OWL verwendete Property, die Konzepte, d.h. Klassen miteinander verknüpft (Beispiel: „Ist Schauspieler in“).
Ontologie (Meta-Modell)	Formale Spezifikation von Wissen, Wissensrepräsentation.
Ontologiesprache (Meta-Meta-Modell)	Bildet ein formales, gemeinsames Vokabular, das dazu verwendet werden kann, untereinander kompatible Vokabulare zu erstellen.
Open Street Map	Das Open Street Map Projekt ist ein Crowd-Sourcing Projekt, in dem die Benutzer gemeinsam Karten erstellen und editieren (http://www.openstreetmap.org/).
Property (Prädikat)	In Aussagen ist das Prädikat die Eigenschaft mit der das Element beschrieben wird. Eine Property, engl. Eigenschaft, bezeichnet in der Modellierung von Ontologien eine Relation zwischen Konzepten (Beispiel SubClassOf – siehe Abbildung 19) oder eine Relation zwischen Konzepten und Literalen (Beispiel: „ist Regisseur von“ –siehe Abbildung 17).
RDF-Graph	Werden über Ressourcen verschiedene Aussagen getroffen, lassen sich diese miteinander verknüpfen und es entsteht ein Graph – ein Geflecht von Aussagen über eine Menge von Ressourcen.
Reasoning	Automatisches, maschinengesteuertes Schlussfolgern von neuem Wissen auf der vorhandenen Wissensbasis.
Relationen	Beziehungen zwischen Instanzen, modelliert durch Eigenschaften

Begriff	Beschreibung
	von Konzepten, d.h. synonym zu Eigenschaft und Property.
Ressource	Ressourcen sind diejenigen Elemente, über die in einer Wissensbasis Aussagen getroffen werden.
Semantisches Modell	Modell, das Bedeutung (Wissen) erfasst. Für diese Mitteilung synonym zu Ontologie. Wird auch als Meta-Modell bezeichnet.
Subject (Subjekt)	Das Subjekt stellt in einer Aussage das zu beschreibende Element dar.
Taxonomie	Definitionen von Konzepthierarchien, in Ontologie als Klassenhierarchie modelliert, d.h. Klassen und Subklassen über die Vererbung.
Triple	Aussage in der Form *Subjekt Prädikat Objekt.* Mit Hilfe solcher Aussagen kann Wissen repräsentiert werden.
Vokabular (Meta-Modell)	Eine formale Definition eines Vokabulars stellt ein Meta-Modell dar und kann gemeinsam von verschiedenen Systemen verwendet werden. Ein Vokabular oder Meta-Modell bietet die Grundlage für das gemeinsame Verständnis von RDF-Aussagen.
Typen	Es werden die Begriffe Literal, Basisdatentyp und Typen verwendet. Wie ist der Zusammenhang oder sind dies Synonyme? Typen von Instanzen der Ontologie sind deren Klassen. Typen von Literalen sind Basis-Datentypen.
Wissensbasis	Menge von Aussagen über Ressourcen, auch Datenbasis genannt.

1 Einleitung

Standardisierte Kommunikationsdienste für Fahrgastinformationssysteme können von hochinnovativen und neuartigen Technologien profitieren, die auf den in IP-KOM-ÖV entwickelten und durch den VDV standardisierten Kommunikationsdiensten basieren (VDV-Schrift 430, 2013), (VDV-Schrift 431 Teil 2, 2013). Für das World Wide Web gehören zu den wichtigsten Innovationen der letzten Jahre die Technologien des Semantic Web. Von Tim Berners-Lee bereits 2001 in einer Vision beschrieben, werden die Technologien für das Semantic Web nun marktreif und gewinnen für kommerzielle Anwendungen an Bedeutung (Berners-Lee, Hendler, & Lassila, 2001). Sie lassen sich dabei nicht ausschließlich auf das World Wide Web anwenden, sondern ermöglichen beispielsweise auch die hochflexible und dynamische Individualisierung von Fahrgastinformationssystemen und deren Ergänzung durch Mehrwert- und Zusatzdienste für den Fahrgast. Voraussetzung für den Einsatz von semantischen Technologien ist ein semantisches Datenmodell für den entsprechenden Einsatzbereich, die Domäne. Semantische Modelle, auch Ontologien genannt, sind solche Datenmodelle zur Wissensrepräsentation. Sie bilden die Gegenstände, Konzepte und Personen bzw. Rollen einer Domäne und deren Beziehungen untereinander ab. Auf diese Weise wird Wissen über eine Domäne, beispielsweise den öffentlichen Verkehr, maschinenverständlich gespeichert. Solche semantisch beschriebenen Daten können von Programmen verstanden und automatisch verarbeitet werden. Innovative Anwendungen und Dienste, die Semantic Web Technologien einsetzen, können auf Grundlage semantischer Daten Schlüsse ziehen und intelligent auf Ereignisse oder Situationen reagieren.

1.1 Beispiel für die Verwendung von Ontologien

Ein kleines Beispiel soll die Anwendung von Ontologien illustrieren. Als Beispielontologie dient eine Ontologie, die für Restaurants beschreibt, welche Gerichte diese servieren. Zusätzlich definiert die Ontologie, welche Zutaten diese Gerichte enthalten.

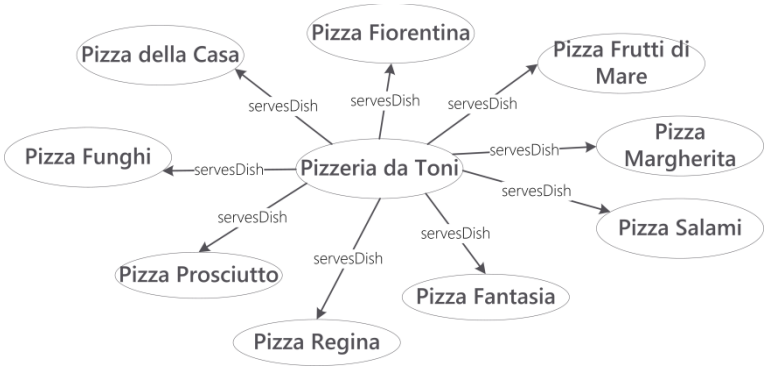

Abbildung 2: Beispiel – Gerichte in der Pizzeria da Toni

Ein Beispiel-Datensatz wird in Abbildung 2 gezeigt. Die Pizzen, die in der Pizzeria da Toni serviert werden, werden dort definiert – die Pfeile, die mit „servesDish“ beschriftet sind, geben an, welche Gerichte die Pizzeria serviert.

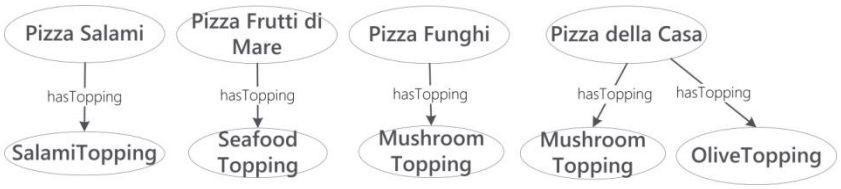

Abbildung 3: Beispiel – Pizzabeläge

Die Gerichte, hier im Beispiel verschiedene Pizzen, haben bestimmte Zutaten, d.h. Beläge, die in Abbildung 3 und Abbildung 4 durch die Pfeile mit der Beschriftung „hasTopping“ angegeben werden.

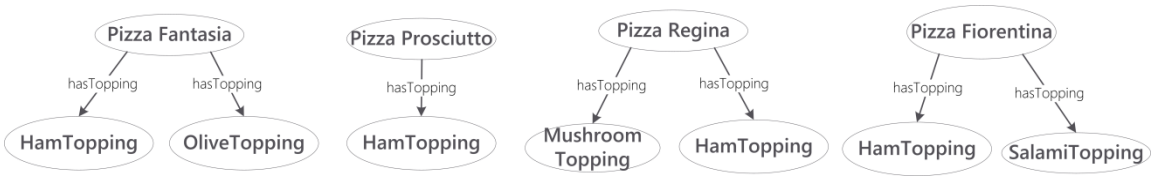

Abbildung 4: Beispiel - Pizzabeläge

Für verschiedene Beläge kann dann angegeben werden, welche Zutaten sie enthalten, wie Abbildung 5 bis Abbildung 7 zeigen. Auf dieser Datenbasis lässt sich nun programmatisch erkennen, ob beispielsweise in der Pizzeria da Toni vegetarische Gerichte zu finden sind. Vegetarische Gerichte (Vegetables) werden dabei in der Ontologie als solche definiert, die kein Fleisch (Meat) und keine Meeresfrüchte oder Fisch (Seafood) enthalten. Es ist dann nicht mehr nötig, für jede Pizza anzugeben, ob sie vegetarisch ist, dies kann das System selbständig erkennen.

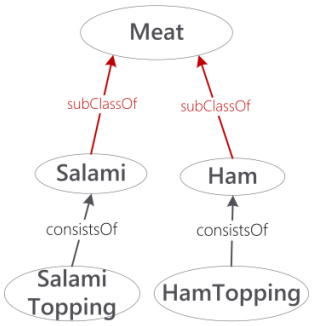

Abbildung 5: Beispiel - Pizzabeläge mit Fleisch

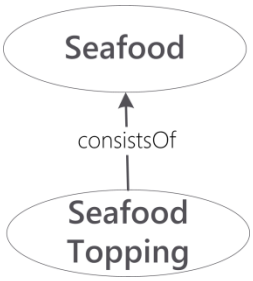

Abbildung 6: Beispiel – Pizzabeläge mit Meeresfrüchten

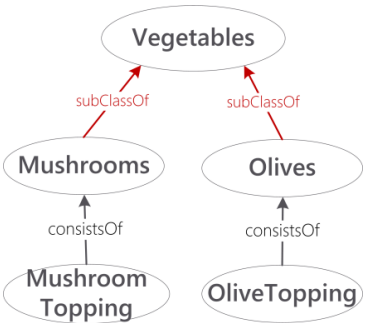

Abbildung 7: Beispiel – Pizzabeläge mit Gemüse

Automatisch kann also berechnet werden, dass die Pizza Funghi, Pizza della Casa und die Pizza Margherita diesen Kriterien entsprechen. Möchte nun jemand zwar kein Fleisch, aber durchaus Meeresfrüchte oder Fisch essen, kann auch dies auf der Datenbasis berechnet werden. Die Pizza Frutti di Mare kommt dann ebenfalls in Frage für ein Abendessen in der Pizzeria da Toni. Natürlich lassen sich auch weitere Wünsche erfüllen. So kann zum Beispiel die Frage beantwortet werden, ob die Pizzeria Pizzen serviert, die zwar Schinken, aber keine Oliven enthalten. Nützlich sind solche Aussagen beispielsweise für eine Mobilapplikation für Smartphones, die Restaurants in der Nähe vorschlägt, die bestimmte Gerichte oder Arten von Gerichten servieren – zum Beispiel Restaurants, die vegetarische oder vegane Gerichte führen - oder aber feststellt, ob spezielle Wünsche erfüllt werden können, zum Beispiel ob die Lieblingspizza mit Oliven und Pilzen erhältlich ist.

1.2 VDV-Ontologien zur Fahrgastinformation

Im vorliegenden Dokument werden Ontologien definiert und beschrieben, die als Basis für intelligente Kommunikationsdienste im öffentlichen Verkehr dienen. Es handelt sich dabei um drei sich ergänzende Modelle, die leicht erweiterbar sind, um für die jeweiligen Anwendungsfälle einfach spezialisiert werden zu können. Die Ontologien wurden im Forschungs- und Standardisierungsprojekt IP-KOM-ÖV entwickelt und basieren auf den im Projekt erarbeiteten Standards, die in den VDV-Schriften 430 und 431 definiert werden (VDV-Schrift 430, 2013) (VDV-Schrift 431, Teil 1, 2013) (VDV-Schrift 431 Teil 2, 2013). Die Ontologien

ergänzen die dort beschriebenen Kommunikationsdienste um die grundlegendenden Datenmodelle, die die Erweiterung eines TRIAS-Systems mit semantischen Technologien ermöglichen. Dazu gehören ein Klassifikationsmodell, ein Kontextmodell und ein Interaktionsmodell. Jede Ontologie hat dabei ein bestimmtes Aufgabengebiet.

Das **Klassifikationsmodell** bildet den Kern, da es die grundlegenden *Entitäten* und *Konzepte* des öffentlichen Verkehrs abbildet, wie beispielsweise Fahrzeuge oder Störungen. Es ist daher eine klassische Domänen-Ontologie, deren Aufgabe es ist, für einen spezifischen Einsatzbereich die nötigen Konzepte zur Verfügung zu stellen.

Das **Kontextmodell** bildet den Kontext eines Fahrgastes im öffentlichen Verkehr ab. Damit wird einem Ontologie-basierten Fahrgastinformationssystem ermöglicht, die Situation eines Fahrgastes zu erfassen, zu beschreiben und darauf zu reagieren. Das Kontextmodell stützt sich dabei auf das Klassifikationsmodell, um beispielsweise abzubilden, dass ein Fahrgast sich auf einer bestimmten Reise im ÖV befindet.

Da der öffentliche Verkehr sehr viele Facetten hat und von vielen Fahrgästen zu verschiedenen Zwecken genutzt wird, dient das **Interaktionsmodell** dazu, Erweiterungspunkte für zusätzliche, ergänzende Modelle zu bieten. Verschiedene Ontologien können hier eingebunden und zusammen mit den beschriebenen Ontologien verwendet werden. Im vorliegenden Dokument werden ein „Point-of-Interest“-Modell, ein Wetter-Modell und ein Barrierefreiheits-Basismodell beschrieben, die aufzeigen, wie die vorhandenen Ontologien erweitert und gemeinsam verwendet werden können. In dieser Schrift werden die notwendigen Grundlagen zur Anwendung und Erweiterung der Ontologien gelegt. Weiterhin werden die Kernkonzepte der Ontologien und die Besonderheiten der Modellierung beschrieben.

Diese Mitteilung und die darin beschriebenen Ontologien, richten sich damit an Entwickler, die intelligente Ontologie-basierte Kommunikationsdienste unter Nutzung von Semantic Web Technologien planen und implementieren. Die Ontologien bilden die Datengrundlage für Design und Umsetzung von „Smart Apps“, wie sie in Kapitel 2 beschrieben sind. Sie ermöglichen auch die Entwicklung weitergehender intelligenter Dienste. Fahrgastinformation und Reisebegleitung können durch standardisierte Ontologien von zukunftsweisenden Technologien profitieren, sowohl in den Bereichen adaptiver, personalisierter oder spezialisierter Anwendungen, als auch im Bereich von Linked Open Data und Open Services, in dem der VDV bereits tätig wird. Entwicklern dienen die vorliegenden Ontologien als Datengrundlage zum Entwurf von Diensten und Smart Apps. Die Modelle bieten Erweiterungspunkte, sollten für die gewünschten Anwendungsbereiche zusätzliche Modellierungen nötig sein oder externe Ontologien eingebunden und externe Datenquellen damit nutzbar gemacht werden. Die Ontologien müssen dabei nicht alle zusammen in einem System verwendet werden und können je nach Anwendungsfall kombiniert werden. Die Architekturen der Dienste und Anwendungen werden hier nicht standardisiert, da sie, individuell nach Anwendungsfall umgesetzt werden sollten. In der TRIAS-Architektur aus (VDV-Schrift 430, 2013) werden „semantische Dienste“ bereits vorgesehen. Die genaue Ausgestaltung und Implementierung der Dienste hängt von deren gewünschter Funktionalität ab. Die Erfahrungswerte aus der Umsetzung eines prototypischen „semantischen Portalsystems“ mit Nutzung der Dienste in einer „Smart App“ für mobile Endgeräte finden sich, mit einer Beschreibung der beispielhaften Architektur und verwendeter Komponenten, in Anhang I**Fehler! Verweisquelle konnte nicht gefunden werden.**. Es wurde im Rahmen des Projektes IP-KOM-ÖV umgesetzt und getestet und dient als Beispiel für den Einsatz der Ontologien für intelligente Kommunikationsdienste im öffentlichen Verkehr.

1.3 Hinweise zum nachfolgenden Text

Im vorliegenden Dokument werden einige Konventionen zur besseren Verständlichkeit eingehalten. *Kursiv* geschriebene Wörter sind neu eingeführte Benennungen, die in Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.** in Form eines Glossars zusammengefasst werden.

Namen von *Properties* (Eigenschaften), *Klassen* oder anderen Modellierungsaspekten werden normal geschrieben, wenn sie schwer vom übrigen Text zu unterscheiden sind, werden sie in Anführungszeichen gesetzt.

In bildhaften Darstellungen der Modellierung stellen Ellipsen Klassen dar. Sind die Ellipsen grau hinterlegt, stellen sie *Instanzen* dar. Rechtecke sind *Literale*, Pfeile stellen Properties dar. Dabei sind schwarze Pfeile in den Ontologien definierte Properties, während rote Pfeile Properties sind, die durch die *Ontologiesprache* definiert sind und die Klassenhierarchie (*Taxonomie*, Subklassenbeziehungen) oder Instanz-Beziehungen beschreiben.

2 Anwendungsbereich der Ontologien

Die hier beschriebenen Ontologien bilden die Grundlage für intelligente und innovative Fahrgastinformationsdienste, die die Fahrgäste im öffentlichen Verkehr in jeder Situation möglichst gut unterstützen sollen. Diese Dienste können beispielsweise mobil mit Applikationen (Apps) auf mobilen persönlichen Endgeräten genutzt werden. Solche Apps werden im Folgenden als „Smart Apps“ bezeichnet – Applikationen für Mobilgeräte, die innovative Kommunikationsdienste des ÖV nutzen und damit den Fahrgast intelligent unterstützen. Zur Illustration der Anwendung von Ontologien für intelligente Fahrgastinformationsdienste und Smart Apps werden in diesem Kapitel potenzielle Anwendungsbeispiele vorgestellt.

2.1 Beispiele für Smart Apps auf Basis der VDV-Ontologien für Fahrgastinformation

Die Anwendungsbeispiele für Smart Apps wurden anhand der Persona entwickelt, die im IP-KOM-ÖV Projekt erstellt wurden (VDV-Mitteilung 7023, 2012). Als beispielhafte Einsatzgebiete für Smart Apps dienen dabei der Bereich Tourismus und eine um intelligente Funktionen erweiterte Reisebegleitung für Vielfahrer.

Abbildung 8: Persona Carla Alvarez

Abbildung 9: Persona Michael Baumann

2.2 Persona Carla Alvarez

Carla Alvarez (Abbildung 8) ist mit ihrem Mann Fabio als Touristin auf einer Städtetour in Deutschland unterwegs. Beide sind daher nicht vertraut mit dem lokalen öffentlichen Nahverkehr. Sie möchten für ihren Besuch allerdings die öffentlichen Verkehrsmittel nutzen und damit in der ihnen zur Verfügung stehenden Zeit möglichst viel von der Stadt sehen. Carla und Fabio haben keine Ortskenntnisse und möchten daher möglichst umfassend begleitet werden. Beide sind interessiert an moderner Kunst und guter, lokaler Küche, wobei Fabio vegetarisch essen möchte. Carla erkundet außerdem bei schönem Wetter sehr gerne botanische Gärten und Parkanlagen in den Städten, die sie besucht. Sie kann ebenfalls angeben, wie viel sie in Restaurants, Bars und auch Museen bereit ist zu zahlen.

2.3 Smart App: Tourist Guide

Carla und Fabio sind mittlerweile auf ihrer Städtereise in Stuttgart angelangt. Auf der Zugreise nach Stuttgart findet Carla eine App für ihr Smartphone mit dem Namen „Smart Stuttgart Guide“ (Abbildung 10). Sie installiert die App auf ihrem Smartphone. Noch während der Zugfahrt gibt Carla ihr Hotel als Startpunkt für ihre Erkundungsrouten in Stuttgart ein. Sie gibt außerdem ein, dass sie gerne Parks und Gärten besucht sowie moderne Kunst mag. Außerdem kann sie ihre und Fabios Präferenzen für Restaurants und die Dauer ihres Aufenthaltes einstellen. Die App schlägt ihr für die Dauer ihres Aufenthaltes verschiedene Tarifoptionen für zwei Personen vor. Carla

entscheidet sich, für den nächsten Tag ein Gruppen-Tages-Ticket zu kaufen, da sie für den Weg zum Hotel noch das City-Ticket ihres Bahntickets nutzen können.

Da Carla und Fabio abends in Stuttgart ankommen, sucht sich Carla direkt eine Verbindung zu ihrem Hotel Garni in Stuttgart Vaihingen heraus. Da dieses voreingestellt ist, geht das ganz einfach von ihrem aktuellen Standpunkt aus über eine Schaltfläche „Zum Hotel“. Im Umkreis des Hotels sucht Carla noch nach möglichen Restaurants für ein Abendessen (Abbildung 11). Sie kann sich direkt Restaurants in der Umgebung anzeigen lassen. Die Restaurants wurden nach den voreingestellten Vorlieben und den Preiswünschen der beiden ausgesucht.

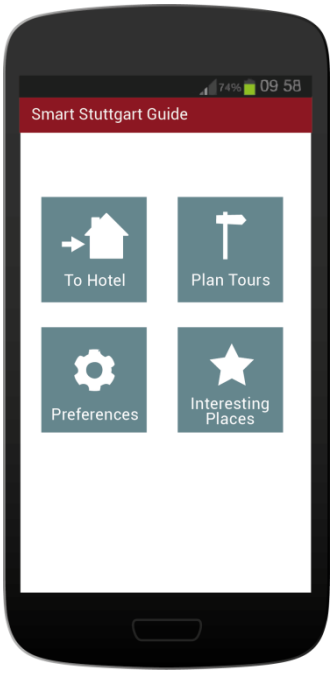

Abbildung 10: Beispiel für eine Smart Stuttgart Guide App - Übersicht

Abbildung 11: Beispiel für eine Smart Stuttgart Guide App - erreichbare Restaurants

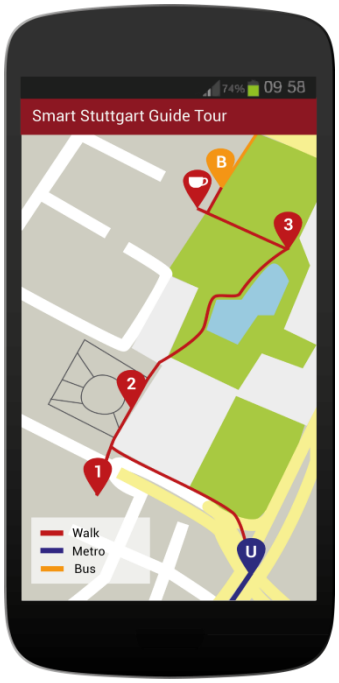

Abbildung 12: Beispiel für eine Smart Stuttgart Guide App – ein Tourenvorschlag

Während sie auf ihr Abendessen warten, plant Carla mit ihrer App den nächsten Tag in Stuttgart. Die App bietet ihr die Möglichkeit, für eine von ihr bestimmte Zeitspanne ganze Touren mit dem ÖV zu planen, die sie zu verschiedenen, für sie interessante Orte (Points of Interests, POI) bringen. Da Fabio und Carla für den nächsten Tag noch nichts vorhaben, lässt sich Carla eine Tour für den ganzen Tag planen. Die App plant dabei mit ein, dass Carla und Fabio einen Snack zum Mittag essen möchten. Carla bekommt verschiedene Thementouren zur Auswahl: eine Museumstour, die sie zu zwei der Kunstmuseen in Stuttgart führt, dazwischen aber noch einige Stuttgarter Sehenswürdigkeiten einplant, wie das neue und das alte Schloss in Stuttgart. Da die App die Wettervorhersage für den nächsten Tag mit einbezieht und diese schönes Wetter verheißt, wird ihr eine Tour durch Gärten und Parks in Stuttgart angeboten, die ebenfalls an den Schlössern vorbei durch den Schlossgarten führt, einen Spaziergang durch den Rosensteinpark und einen Besuch im botanisch-zoologischen Garten Wilhelma vorsieht. Eine dritte Route ist gemischt und bezieht das neue Kunstmuseum ebenso ein wie einen Spaziergang durch Schlossgarten und Rosensteinpark. Zwischen den einzelnen Sehenswürdigkeiten auf den Touren schlägt die App jeweils einen passenden Ort für ein Mittagessen und Einkaufsmöglichkeiten für Souvenirs vor. Die einzelnen Stationen einer Tour sind mit öffentlichen Verkehrsmitteln zu erreichen und die App berechnet die jeweiligen Verbindungen bereits mit ein. Auch die Öffnungszeiten der einzelnen Points of Interest werden mit einberechnet. Carla und Fabio können sich die einzelnen Touren und die eingeplanten Sehenswürdigkeiten auf der Karte in der App anschauen. Für jeden Zwischenstopp werden Details angezeigt, eine Beschreibung und

Besonderheiten. Hier sieht Carla, dass die App ihr für den Besuch der Wilhelma ein spezielles Angebot vorschlägt. Da Carla angegeben hatte, mit einem Tages-Ticket unterwegs zu sein, kann sie auf den Eintrittspreis der Wilhelma einen Rabatt bekommen.

Carla und Fabio entscheiden sich für die gemischte Tour, merken sich aber einen Besuch bei der Wilhelma für den darauffolgenden Tag. Carla wählt die Tour in der App aus. Als sie am nächsten Tag ihre Tour am Hotel Garni beginnen, weist die App Carla per Benachrichtigung darauf hin, dass auf der ersten geplanten Verbindung in die Stadt eine Störung vorliegt, da eine S-Bahn ausfällt. Die App schlägt ebenso eine Alternativverbindung mit der Stadtbahn vor. Geleitet durch die App finden Carla und Fabio zu ihrem ersten Stopp und schauen sich den Schlossplatz, das alte Schloss und das neue Schloss an (Abbildung 12). Von dort finden sie leicht zum neuen Kunstmuseum und sehen sich dort die Ausstellungen an. Die App schlägt ihnen nun eine Mittagspause in einem Restaurant in der Nähe vor, die beiden haben jedoch noch keinen Hunger und gehen daher direkt zum nächsten Programmpunkt über, der sie auf einem Spaziergang über den Schlossgarten in den Rosensteinpark führt. Nach einer Weile bekommen sie doch Hunger. Sie können die Tour über die App modifizieren und direkt eingeben, dass sie ein Restaurant oder einen Imbiss für ein Mittagessen suchen. Die App unterbricht die Tour und schlägt verschiedene Alternativen vor. Nur eine Stadtbahnhaltestelle entfernt finden Carla und Fabio ein italienisches Restaurant und die App fügt einen Zwischenstopp dort zur Tour hinzu. Nach dem späten Mittagessen führt die App Carla und Fabio über einen weiteren kleinen Spaziergang bis nach Bad Cannstatt. Das letzte Stück legen die beiden dabei wieder mit der Stadtbahn zurück. Der gemütliche Ortskern von Bad Cannstatt begeistert die beiden und hier sehen sie auch zum ersten Mal, dass Stuttgart an einem Fluss gelegen ist. In der Fußgängerzone von Bad Cannstatt essen sie spontan ein Eis. Sobald sie den Rückweg antreten möchten, berechnet ihnen die App die beste Verbindung zurück zum Hotel. Diesmal fahren Carla und Fabio mit der S-Bahn zurück nach Stuttgart Vaihingen. Carla ist begeistert von der flexiblen App und plant bereits den nächsten Tag. Dafür gibt sie den Besuch im botanisch-zoologischen Garten vor, den sie sich vorgenommen haben. Die App schlägt als Abendgestaltung eine Musical-Veranstaltung im SI-Centrum vor, das vom Hotel aus gut erreichbar ist. Carla und Fabio können direkt in der App Tickets reservieren und sehen, mit welchen Verkehrsmitteln sie das SI-Centrum erreichen und wann sie die Wilhelma spätestens verlassen müssen, um nicht zu spät zum Musical zu kommen. Der Musical-Besuch rundet den gelungenen Besuch in der Baden-Württembergischen Landeshauptstadt ab.

2.4 Persona Michael Baumann

Michael Baumann (Abbildung 9) lebt im Großraum Stuttgart und pendelt täglich mit dem ÖPNV zu seinem Büro in Stuttgart Vaihingen. Er ist als Unternehmensberater tätig und muss häufig beruflich in Deutschland reisen. Geschäftsreisen unternimmt er, soweit möglich, ebenfalls mit öffentlichen Verkehrsmitteln. Er schätzt es, komfortabel zu reisen und pünktlich seine Ziele zu erreichen und ist außerdem sehr umweltfreundlich eingestellt. Für ihn gehört die Planung von ÖV-Verbindungen von und zum Büro, aber auch zu Terminen zu seinem Arbeitsalltag. Da Michael außerdem seine tägliche Strecke sehr gut kennt, möchte er nur Informationen erhalten, die er wirklich benötigt, zum Beispiel zu Störungen. Ticketinformationen benötigt er hingegen keine.

2.5 Smart App: Intelligente Agendaplanung

Michael Baumann bekommt von einem Geschäftspartner eine Terminanfrage für den nächsten Dienstag, 15:30 Uhr, für ein Arbeitstreffen in Frankfurt am Main. Die Anfrage enthält auch die Adresse des Termins, Börsenstraße 2, Zimmer Nummer 1050. Michael nimmt die Terminanfrage in seiner App zur Agendaplanung an (Abbildung 13). Unter Einbeziehung seiner aktuellen Terminlage und anstehender Aufgaben, wie sie zum Beispiel aus seinem Kalender entnommen werden können, werden Vorschläge von Verbindungen generiert und Michael gemeldet. Da Michael noch einen Termin am zeitigen Vormittag sowie ein paar kleinere Aufgaben zu erledigen

hat, berechnet die Anwendung den spätesten möglichen Abfahrtzeitpunkt, wobei jedoch ein gewisser zeitlicher Puffer eingeplant wird. Ein Planungsvorschlag sieht vor, dass Michael um 12:35 Uhr von der Haltestelle Schwabstraße, in der Nähe seines Büros, mit der S-Bahn zum Hauptbahnhof Stuttgart fährt, wo er vor Reiseantritt noch einen kleinen Mittagsimbiss einnehmen kann (Abbildung 14). Mit dieser Verbindung erreicht er den Tagungsort in Frankfurt 15:08 Uhr. Michael wählt diesen Vorschlag aus und bestätigt ihn gegenüber der Anwendung. Diese veranlasst daraufhin automatisch die Buchung der Fahrscheine für die involvierten Teilstrecken. Da Michael ein Monatsticket für Stuttgart besitzt und die Anwendung dieses kennt, werden nur Tickets für die Verbindungen gebucht, für die er keine Zeitkarte besitzt, d.h. vom Hbf. Stuttgart zum Hbf. Frankfurt a.M.im ICE um 13:27 und für die S-Bahn vom Hbf. Frankfurt a.M. zur Haltestelle Börsenstr. und jeweils zurück nach Stuttgart am selben Nachmittag. Die Reisebegleitungs-App kennt dabei Michaels persönliche Vorlieben aufgrund früherer Nutzereingaben und expliziter Einstellungen. Entsprechend werden bei der Kartenbuchung seine Wünsche berücksichtigt, sodass im ICE explizit ein Fensterplatz mit Tisch in der 1. Klasse reserviert wird. So kann Michael während der Reise bequem nochmals einige Dokumente in Vorbereitung auf das Arbeitstreffen durchschauen.

Um 12:00 Uhr erinnert die Anwendung Michael an seine bevorstehende Abreise. Er packt seine Sachen zusammen und begibt sich zur Haltestelle Schwabstr., wo er eine S-Bahn früher erreicht. Er steigt bereits um 12:20 Uhr in die S 1. Die Anwendung erkennt dies und startet automatisch die Reisebegleitung. Zunächst verläuft die Reise nach Plan, Michael erreicht den Hauptbahnhof und kauft sich dort etwas zu Essen. Da er eine frühere S-Bahn erreicht hat und zusätzliche 15 Minuten Zeit hat, erinnert ihn seine App daran, dass er noch ein Hustenmittel für seine Frau kaufen muss. Die App weist ihn darauf hin, dass sich im Bahnhof eine Apotheke befindet und Michael kann diese Besorgung noch erledigen. Dann jedoch wird er von seiner Anwendung über eine Gleisänderung des ICE informiert. Die Anwendung reagiert flexibel auf dieses Ereignis und berechnet automatisch die Wegänderung. Michael wird von seiner App zum Gleis und innerhalb des Zugs weiter bis zu seinem Sitzplatz navigiert. Er verliert somit keine Zeit beim Suchen nach dem richtigen Zug oder Wagen und erreicht seinen Sitzplatz pünktlich und komfortabel.

Abbildung 13: Beispiel für eine App „Smart Agenda“: Übersicht

Abbildung 14: Beispiel für eine App „Smart Agenda“: Tagesplan

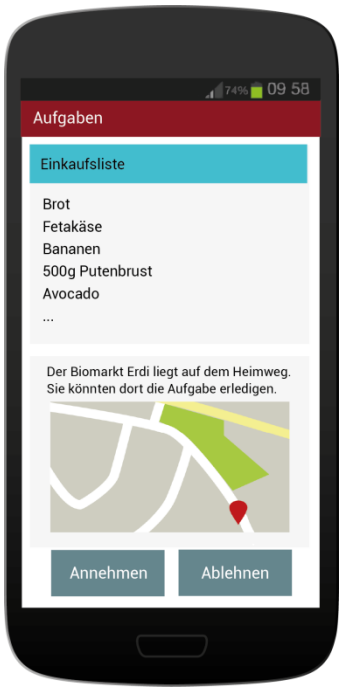

Abbildung 15: Beispiel für eine App „Smart Agenda“: Aufgaben

Während der Fahrt kommt es an Bahnhöfen zu Verzögerungen, die sich auf die Ankunftszeit seines ICE auswirken. Der ICE erreicht Frankfurt derart verspätet, dass Michael die Anschlussverbindung nicht mehr rechtzeitig erreichen kann. Er muss somit auf die nächste S-Bahn warten, die ihn zur Börsenstr. bringt. Die intelligente Reisebegleitung erkennt die Kollision mit dem Geschäftstermin und informiert daher automatisch alle beteiligten Geschäftspartner über Michaels Verspätung. Michael, der fremd in Frankfurt ist, wird derweil nahtlos von seiner Anwendung zur Anschlussbahn S1 Richtung Rödermark-Ober-Roden navigiert. Dort angekommen wird Michael weiter geführt bis zur Börsenstraße 2, wo sein Termin stattfindet.

Michaels Termin dauert länger als erwartet und da die App erkennt, dass er sich nicht rechtzeitig auf den Weg zum Hauptbahnhof macht, storniert sie nach kurzer Bestätigung durch Michael seine Reservierung und bucht eine Reservierung für einen späteren ICE. Als Michael diese Reise antritt, weist ihn seine Anwendung darauf hin, dass seine Nichte, die in Frankfurt studiert, sich im selben Zug befindet. Michael trifft sich mit Maria im Bordbistro des Zuges und die beiden lachen über den Zufall. Michaels Frau fügt indessen einige Produkte zur gemeinsamen Familien-Einkaufsliste hinzu (Abbildung 15). Michaels Anwendung erkennt, dass er auf dem Heimweg vom Hauptbahnhof in Stuttgart noch seinen Einkauf in einem lokalen Bio-Supermarkt, der zu seinen bevorzugten Einkaufsorten gehört, tätigen kann. Die Anwendung weist Michael darauf hin und fragt ab, ob er den Einkauf einplanen möchte. Er bestätigt dies und hierdurch wird Michaels Frau auf ihrem Gerät benachrichtigt, dass der Einkauf erledigt wird und sie sich nicht weiter darum kümmern braucht.

3 Ontologien: Wissensrepräsentation für intelligente Systeme

Die meisten Softwaresysteme arbeiten heutzutage auf Basis syntaktischer Beschreibungen von Daten. Das bedeutet, dass die Struktur von Daten und Programmcode zur Zeit der Entwicklung eines Systems festgelegt wird. Die Bedeutung der Daten, die das System verarbeitet, sowie der Funktionen, die das System anbietet, ist dem System selbst nicht bekannt. Eine semantische Modellierung von Daten sieht nun vor, die Bedeutung der Daten, die ein System verarbeiten kann, explizit zu machen, d.h. maschinenlesbar darzustellen. So kann das System die Daten nicht nur an Hand ihrer Struktur verarbeiten, sondern diese auch „verstehen". Eine syntaktische, also strukturelle Beschreibung der Daten für ein System wird Datenmodell genannt. Kommt zu der Beschreibung der Struktur auch eine maschinenlesbare Beschreibung der Bedeutung der Daten hinzu, ist dies ein *semantisches Datenmodell* – also ein Modell zur Repräsentation von Wissen.

Rein syntaktische Datenmodelle sind schwer erweiterbar, da eine Veränderung am Datenmodell aufwändige Anpassungen am Programmcode des Systems nach sich ziehen kann. Semantische Datenmodelle können dagegen flexibler und leichter erweitert werden, da die Struktur, die die Semantik der Datenmodelle beschreibt, gleich bleibt und nur die Datenmodelle selbst ergänzt werden können. Eine Erweiterung des semantischen Datenmodells verursacht daher keine aufwändige Anpassung eines Systems mehr. Dazu kommt, dass die semantischen Datenmodelle unabhängig von darunter liegenden Technologien sind. Für ein System verwendete Programmiersprachen, Datenbanktechnologien oder Kommunikationstechnologien können von Umsetzung zu Umsetzung variieren, während die semantischen Datenmodelle unabhängig davon eingesetzt und verarbeitet werden können. Auf Grundlage semantischer Datenmodelle können Systeme selbst Schlüsse ziehen und daher zu Schlussfolgerungen kommen, die nur implizit in den Daten vorhanden sind. Auf Basis solcher Schlussfolgerungen lässt sich beispielsweise die Darstellung von Fahrplandaten für den Benutzer flexibel anpassen oder Entscheidungen zur Filterung von Daten treffen. Solche Systeme können auf Basis des Wissens, das ihnen zur Verfügung steht, intelligent reagieren.

Ein Instrument für die Wissensrepräsentation für informationstechnologische Systeme sind sogenannte Ontologien. Ihren Ursprung haben Ontologien bereits in der Antike, als Instrument zur Beschreibung des „Seienden". In der Informatik bezeichnet man als Ontologie ein formales Datenmodell, das zur Wissensrepräsentation dient. Tom Gruber definierte in einem Artikel von 1993 Ontologien folgendermaßen: *„An ontology is a formal specification of a shared conceptualization"*[1] (Gruber, 1993). Daraus lassen sich als Merkmale extrahieren, dass es sich bei einer Ontologie um eine formalisierte Darstellung handeln muss und dass ihr eine Konzeptualisierung, also eine Beschreibung der Konzepte zu Grunde liegt und diese auf einem Konsens basiert. Letzteres deutet an, dass eine Ontologie, insbesondere für eine spezielle Domäne, immer einen Konsens bei den Domänenexperten finden muss, damit sie das Fachgebiet adäquat beschreiben kann. In diesem Dokument werden Ontologien für die Anwendung im öffentlichen Verkehr definiert, die durch eine breite Basis der Standards und Erfahrung des VDV gestützt sind und damit auf den Konsens der Domänenexperten aufbauen. Möglichkeiten zur Formalisierung von Ontologien werden in Kapitel 5.4 aufgezeigt.

Die in diesem Dokument beschriebenen Ontologien ermöglichen die Umsetzung intelligenter Kommunikationsdienste im öffentlichen Verkehr und dienen zur Repräsentation von Wissen aus dem ÖV. Die Ontologien bleiben dabei leicht erweiterbar und tragen der Tatsache Rechnung, dass zum Zeitpunkt der Definition nicht absehbar ist, welche Entwicklung sich auf den jeweiligen

[1] Deutsche Übersetzung: „Ein Ontologie ist eine formale Spezifikation einer miteinander geteilten Konzeptualisierung."

Gebieten anschließt und welche Bereiche daher in den einzelnen Modellen zu berücksichtigen und abzubilden sind.

3.1 VDV-Ontologien für Fahrgastinformation

In diesem Dokument werden drei große Ontologien definiert, die durch drei kleine Ontologien ergänzt werden, was die Erweiterbarkeit des Ontologie-basierten Ansatzes zeigt. Es handelt sich dabei um:

- **Klassifikationsmodell**: Zentrale Konzepte des ÖPV z. B. Haltestellen, Linien, Störungen, die auch in der TRIAS-Schnittstelle verwendet werden
- **Interaktionsmodell**: Zusätzliche *Relationen* und Zusammenhänge zwischen den klassifizierten Entitäten, Erweiterungsmöglichkeiten durch zusätzliche Domänenontologien, d.h. Ontologien, die auf bestimmte Einsatzbereiche spezialisiert sind. In diesem Dokument beschrieben werden eine Point-of-Interest Ontologie, eine Wetterontologie und eine Barrierefreiheits-Basisontologie.
- **Kontextmodell**: Kontextfaktoren, die Einfluss auf die Interaktion eines Nutzers mit dem System haben. Dabei sind Kontextfaktoren Eigenschaften der systemrelevanten Umgebung, inklusive des Nutzers und des Systems selbst.

Das Interaktionsmodell dient dabei, wie Abbildung 16 zeigt, als Klammer um die übrigen Modelle. Es integriert Klassifikations- und Kontextmodell genauso wie erweiterte Ontologien, die Relationen und Entitäten für erweiterte Anwendungsfälle modellieren. Das bedeutet jedoch nicht, dass die Modelle nicht auch nur teilweise oder sogar einzeln genutzt werden können. Besonders das Klassifikationsmodell kann auch einzeln Anwendung finden. Ebenso besteht die Möglichkeit, einzelne Ontologien wegzulassen, indem diese für die Anwendung aus dem Interaktionsmodell entfernt werden. Werden wiederum zusätzliche Schemata oder Ontologien benötigt, können diese einfach in das Interaktionsmodell integriert und gemeinsam mit den weiteren Ontologien verwendet werden.

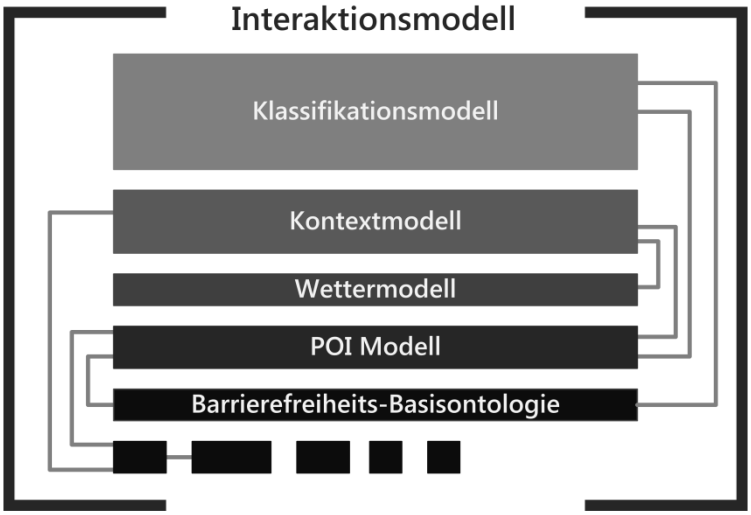

Abbildung 16: Integration einzelner Ontologien im Interaktionsmodell zur gemeinsamen Nutzung

Die kurz beschriebenen Ontologien werden ab Kapitel 6 ausführlich vorgestellt. Dazu ist es notwendig, einige Konventionen, Zusammenhänge und technische Grundlagen zu schaffen, die die Funktionsweise und Modellierung verdeutlichen sollen. Auf diese Aspekte wird in den folgenden Kapiteln 4 und 5 näher eingegangen.

4 Konventionen

Für die Modellierung von Ontologien müssen verschiedene technische Konventionen eingehalten werden, um beispielsweise die Lesbarkeit, Nachvollziehbarkeit und Wartbarkeit zu gewährleisten oder zu erhöhen. Im Folgenden werden die Konventionen vorgestellt, die bei der Modellierung der VDV-Ontologien für Fahrgastinformation eingesetzt wurden. Sollten die Ontologien für erweiterte Einsatzzwecke ergänzt werden, ist zu empfehlen, die Konventionen weiter einzuhalten, um Erweiterungen transparent und übersichtlich handhabbar zu halten.

4.1 Referenzen auf Schemata

Die Ontologien basieren auf den Schemata der TRIAS-Schnittstellen (VDV-Schrift 430, 2013), (VDV-Schrift 431, Teil 1, 2013) sowie häufig auf weiteren Standards und Schemata, wie beispielsweise Siri oder TPEG (CEN, TS 15531 Part 1). Wenn bestimmte Elemente in den Ontologien Entsprechungen in anderen Schemata haben, wurden sie mit dem rdfs:seeAlso Tag versehen, worüber auf die korrespondierende Klasse verwiesen werden kann (Listing 1).

```
1 :SituationStatus rdf:type owl:Class ;
2         rdfs:subClassOf :Status ;
3         rdfs:seeAlso http://www.siri.org.uk/siri/siri_situationv1.1.xsd#StatusGroup" .
```

Listing 1: Beispiel für die Verwendung von rdfs:seeAlso

Es ist möglich, mit mehreren rdfs:seeAlso Tags Referenzen auf verschiedene Vorkommen dieser Klasse in unterschiedlichen Schemata zu verweisen, beispielsweise in Siri und TPEG gleichzeitig.

4.2 Uniform Resource Identifier

Die Elemente der Ontologien werden durch Uniform Resource Identifier (URI) eindeutig identifiziert. URIs sind verwandt mit Bezeichnern für Webseiten, URLs (Uniform Resource Locator), können aber im Gegensatz zu diesen auch als Bezeichner für *Ressourcen* verwendet werden, die nicht über das Web abrufbar sind. Eine URI für ein Element in der Ontologie setzt sich dabei aus der Basis-URI der Ontologie zusammen, in der das Element definiert wurde, und der eindeutigen Benennung des Elementes selbst. Die Basis-URIs der Ontologien spiegeln wider, um welche Ontologie es sich handelt. Sie haben wiederum eine gemeinsame "Wurzel“, die URI http://vdv.de/ofi. Die URIs der Ontologien sollten nicht verändert werden, da sie die Ontologien eindeutig kennzeichnen, insbesondere, wenn sie mit weiteren Ontologien zusammen verwendet werden. Für die einzelnen Ontologien wird definiert:

- Das Interaktionsmodell hat die Basis-URI http://vdv.de/ofi, da es die weiteren Ontologien kapselt (siehe hierzu Kapitel 7)
- Das Klassifikationsmodell hat die Basis-URI http://vdv.de/ofi/ontology
- Das Kontextmodell hat die Basis-URI http://vdv.de/ofi/context

Dieses Schema wird so auch in anderen Projekten gelebt, beispielsweise in *DBpedia* (dort gibt es z. B. /resource und /ontology) und hat sich dort zur Übersichtlichkeit und Identifikation der Ontologien bewährt.

4.3 Benennungen und Namen von Elementen

Sämtliche Benennungen von Elementen in den Ontologien werden im sogenannten CamelCase geschrieben, um Leerzeichen in Bezeichnern zu vermeiden. Dabei werden zwei oder mehrere Wörter nicht als Wortgruppen, sondern als ein zusammenhängendes Wort beschrieben, wobei jedes neue Wort durch einen Großbuchstaben gekennzeichnet wird. Klassen- und Instanznamen werden groß geschrieben, Properties beginnen klein. Die Benennungen werden, wo es möglich ist, an die Benennungen von Elementen in den TRIAS-Schemata angelehnt.

4.4 Natürlichsprachige Namen

Das rdfs:label Tag kann verwendet werden, um natürlichsprachige Entsprechungen für Namen von Elementen in den Ontologien zu vergeben. Dabei kann mit dem Sprachen-Tag @de oder @en auf verschiedensprachige Namen verwiesen werden. Hier wird auf bereits vorhandene Übersetzungen zurückgegriffen, beispielsweise in den TPEG Übersetzungstabellen, falls eine TPEG Referenz besteht. Die rdfs:label Tags können in Programmen genutzt werden, um semantische Daten verständlich in natürlicher Sprache zu beschreiben.

4.5 Kommentare

Um Erläuterungen zu Klassen, Instanzen oder Properties festzuhalten, wird das rdfs:comment Tag genutzt. Hier sollte mit einem entsprechenden Sprachen-Tag (@de oder @en) wiederum die Sprache des Kommentars angegeben werden (siehe 4.4). Die Kommentare sollen insbesondere genutzt werden, wenn z. B. Namen nicht mit den entsprechenden Namen der XML Schemastrukturen der TRIAS-Schnittstellen übereinstimmen oder falls Klassennamen nicht selbsterklärend genug sein sollten.

5 Technische Grundlagen

Die Repräsentation von Wissen für informationstechnische Systeme kann auf verschiedene Arten erfolgen, die vom jeweiligen Einsatzzweck abhängen. Mit der Entwicklung des Semantic Web als Erweiterung des World Wide Web durch Wissensrepräsentationen wurden durch das World Wide Web Consortium (W3C) verschiedene Sprachen zur Darstellung von Ontologien für das Web und web-basierte Kommunikation standardisiert. Ebenso werden durch dieses Gremium Technologien entwickelt und standardisiert, die in Ontologie-basierten Systemen zur Anwendung kommen. Die technischen Grundlagen für *semantische Modelle* und semantische Daten werden im Folgenden ausführlich vorgestellt.

5.1 RDF

Das Resource Description Framework (RDF) ist eine formale Sprache zur strukturierten Darstellung von Informationen (Manola & Miller, 2004). Ursprünglich wurde RDF zur flexiblen Beschreibung von Webseiten durch Metadaten entworfen. Im Zuge der Entwicklung des Semantic Web wurde diese Sprache jedoch stark erweitert. RDF ermöglicht es, *Aussagen* im Format *„Subjekt Prädikat Objekt“* formal zu beschreiben. Solche Aussagen werden auf Grund ihrer Struktur auch als *Tripel* bezeichnet. Es können auf diese Weise beliebige Aussagen dargestellt werden. Das *Subjekt* stellt in einer Aussage das zu beschreibende Element dar - das Prädikat ist eine Eigenschaft mit der dieses Element (Subjekt) beschrieben wird (es wird daher als *Property* bezeichnet) und das *Objekt* enthält den Wert der Eigenschaft. Zur Darstellung werden Objekte und Subjekte, die auch *Ressourcen* genannt werden, als Ovale (auch: Knoten) dargestellt. Werden sie durch ein Prädikat verknüpft, verbindet man sie mit einem Pfeil.

In Abbildung 17 werden solche Aussagen beispielhaft dargestellt. Eine einzelne Aussage ist beispielsweise: „Alfred Hitchcock ist Regisseur von Das Fenster zum Hof“, wobei „Alfred Hitchcock“ das Subjekt, „ist Regisseur von“ das Prädikat und „Das Fenster zum Hof“ das Objekt der Aussage ist. Das Objekt dieser Aussage lässt sich wiederum als Subjekt einer weiteren Aussage nutzen, beispielsweise: „Das Fenster zum Hof wurde veröffentlicht 1954“. Als Rechteck dargestellt werden Basis-Datentypen (Integer, String, Boolean, usw.), wie im Beispiel das Veröffentlichungsjahr, auch *Literal* genannt. Bildet man mehrere Aussagen grafisch ab, erhält man einen RDF-*Graph* (Abbildung 17 zeigt einen solchen Graph).

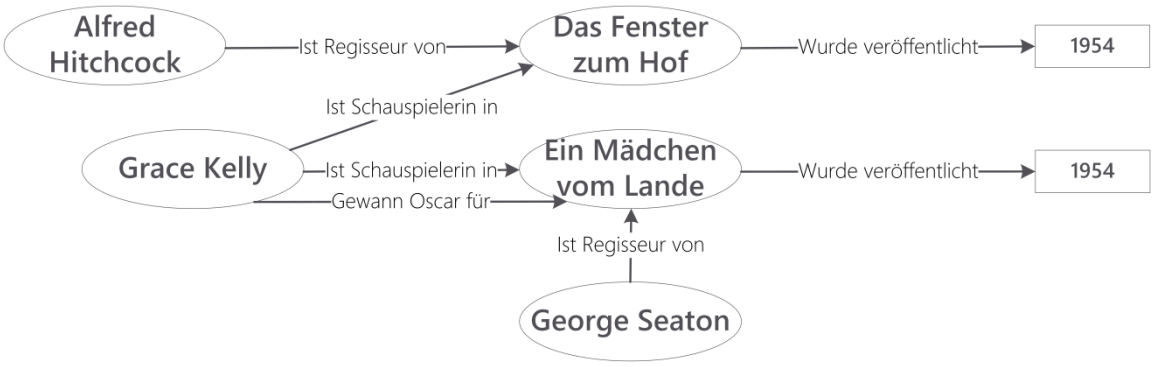

Abbildung 17: Beispiel für Aussagen in RDF

Als Ressourcen in Aussagen können beliebige Dinge dienen, die durch einen Bezeichner eindeutig identifiziert werden. RDF verwendet Uniform Resource Identifier (URI) als Bezeichner für Ressourcen.

Die Darstellung als Graph zeigt, dass in RDF Beziehungen zwischen Ressourcen beschrieben werden können. Ein solcher Graph von Aussagen ist die grundlegende Datenstruktur von RDF. Sogenannte *Triple* Stores sind die bevorzugten Datenspeicher für RDF-Daten und bilden die Tripel

– also die Aussagen - aus RDF auf entsprechende Speicherstrukturen (beispielsweise herkömmliche, relationale Datenbanken) ab.

Die Graphdarstellung von RDF ist zwar zur Speicherung gut geeignet, aber weniger gut für den Austausch von Daten. Aus diesem Grund existieren mehrere Formate zur Serialisierung von RDF. Das wichtigste Format, das auch von allen RDF-Implementierungen unterstützt werden muss, ist RDF/XML, welches die RDF-Aussagen mit Hilfe von XML darstellt.

Ein weiteres, für Menschen einfacher als XML zu lesendes Format zur Darstellung von RDF, ist die Terse RDF Triple Language (Turtle), die einzelne Aussagen wie Sätze in natürlicher Sprache durch einen Punkt trennt und diverse Möglichkeiten bietet, um die Darstellung von Tripel kürzer und übersichtlicher zu machen (siehe Listing 2) (Beckett & Berners-Lee, 2011).

```
1 :Film rdf:type owl:Class .
2
3 :istRegisseurVon rdf:type owl:ObjectProperty .
```

Listing 2: Serialisierung einer Klassendefinition und einer Property-Definition im Turtle Format

Aufgrund der für Menschen besseren Lesbarkeit und der steigenden Verbreitung wird Turtle als Serialisierungsformat für die Entwicklung der VDV-Ontologien für Fahrgastinformation verwendet.

5.2 Sprachen für Vokabulare und Ontologien

Indem Ressourcen durch benannte Eigenschaften und zugehörige Werte beschrieben werden, lassen sich, wie gezeigt, Aussagen durch RDF maschinenlesbar darstellen. Mit Hilfe dieser Aussagen kann Wissen repräsentiert werden, beispielsweise über Filme, Regisseure und Schauspieler. Dies wird als *Datenbasis* oder Modell bezeichnet (siehe Abbildung 18). Um dieses Wissen allerdings zwischen Programme oder, Systemen zu teilen, ist es notwendig, ein gemeinsames *Vokabular* zu finden. Eine formale Definition eines solchen Vokabulars stellt ein *Meta-Modell* dar und kann gemeinsam von verschiedenen Systemen verwendet werden. Ein Vokabular oder Meta-Modell bietet die Grundlage für das gemeinsame Verständnis von RDF-Aussagen. Eine solche Definition lässt sich in RDF selbst darstellen und damit von Maschinen verarbeiten. Wissen der gleichen Domäne, z. B. über Filme, das mit verschiedenen Vokabularen ausgedrückt wird, kann allerdings nicht ohne weiteres zusammengeführt werden – zum Beispiel wegen unterschiedlicher Benennungen und Strukturen. Aus diesem Grund werden Ontologiesprachen definiert, die ein formales, gemeinsames Vokabular bilden, das dazu verwendet werden kann, untereinander kompatible Vokabulare zu erstellen. Sie werden als *Meta-Meta-Modelle* bezeichnet, wie Abbildung 18 verdeutlicht. Die Definition einer Ontologiesprache stellt sicher, dass alle Vokabulare, d.h. Ontologien, die in dieser Sprache ausgedrückt sind, miteinander kompatibel sind, weil dieselben Strukturen zur Definition genutzt werden. Ein Beispiel für eine Ontologiesprache ist das Resource Description Framework Schema (RDFS). Ein Beispiel für ein RDFS-Vokabular wäre eine Film-Ontologie, die die grundlegenden Konzepte und Eigenschaften zum Thema „Film“ beschreibt. Eine Datenbasis hierfür ist beispielsweise der RDF-Graph aus Abbildung 17, der konkrete Filme, Regisseure und Schauspieler in RDF beschreibt.

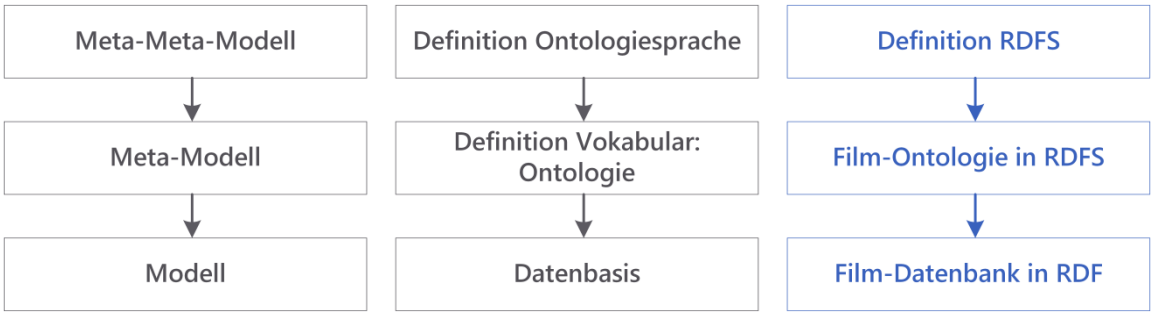

Abbildung 18: Ebenen der Wissensrepräsentation

Die abstrakte Definition von Wissen über ein Themengebiet in einer Ontologiesprache bildet eine Ontologie. Wichtige Bestandteile einer Ontologie sind:

Konzepte: Abstrakte Objektbeschreibung, auch Klasse genannt.

Instanzen: Konkrete Ausprägung einer Klasse.

Typen: Definierte Datentypen einer Ontologie. Der Typ einer Instanz ist der Klasse, von der die Instanz Ausprägung ist. Der Typ eines Literals ist ein Basis-Datentyp.

Relationen: Beziehungen zwischen Instanzen, modelliert durch Eigenschaften von Konzepten. Relationen werden als Properties modelliert.

Vererbung: Relationen und Eigenschaften von Konzepten können auf Unterkonzepte (z. B. Subklassen) vererbt werden.

Axiome: In die Ontologie manuell eingebrachte allgemeingültige Fakten, welche nicht aus Konzepten abgeleitet werden können. In den vorliegenden Ontologien werden keine Axiome definiert.

Ontologien stellen damit ein Werkzeug zur expliziten, maschinenverarbeitbaren Repräsentation von Wissen dar. Anwendungen können die von Ontologien bereitgestellte Semantik verwenden, um neue Aussagen mittels automatischen Schlussfolgerns, *Reasoning* genannt, aus bereits vorhandenen abzuleiten. Zwei Sprachdefinitionen von Ontologiesprachen sollen im Folgenden betrachtet werden.

5.3 RDFS

Um RDF Vokabulare zu erstellen, wurde die RDF Vocabulary Description Language (RDF Schema, kurz RDFS) als Metasprache für RDF entworfen (Brickley & Guha, 2004). Sie bildet ein Vokabular, das einige grundsätzliche Ressourcen einführt und deren Semantik festlegt und ist damit eine Ontologiesprache nach Abbildung 18. Mit RDFS können nun wiederum eigene Vokabulare – d.h. Ontologien – erstellt werden. Mit RDFS ist es beispielsweise möglich, bestimmte Ressourcen als Klassen zu kennzeichnen. Andere Ressourcen können einer Klasse zugewiesen werden und werden damit zu Instanzen (Individuen, Objekten) der Klasse. Mehrere Klassen können durch eine Subklassen-Beziehung in einer Taxonomie hierarchisch angeordnet werden Mit RDFS lassen sich Properties definieren und Definitions- und Wertebereich (Domain und Range) der Property festlegen. Damit macht es RDFS möglich, einfache Ontologien zu erstellen.

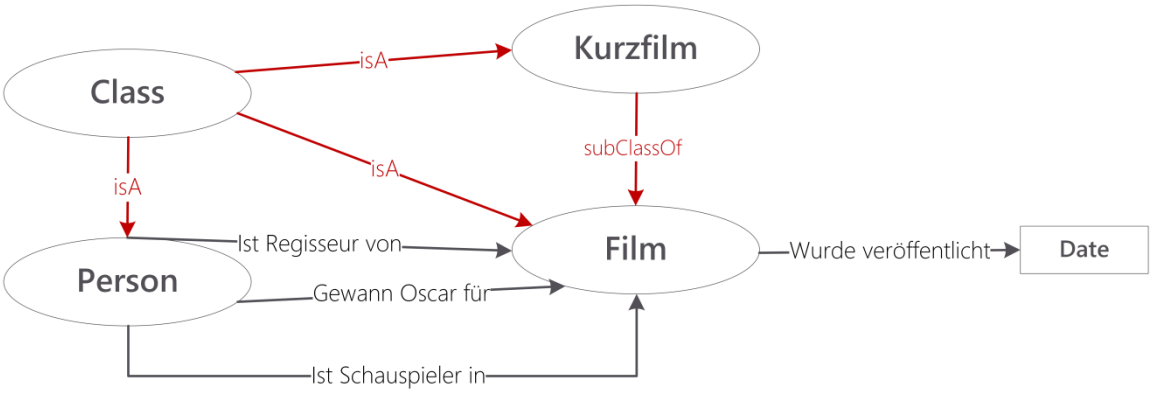

Abbildung 19: ein RDFS-Vokabular für Filme

Abbildung 19 zeigt eine einfache Film-Ontologie. Darin sind die Klassen „Person“, „Film“ und „Kurzfilm“ definiert. Es wird durch die Property subClassOf festgelegt, dass „Kurzfilm“ eine Subklasse von „Film“ ist – dies sagt aus, dass Kurzfilme eine speziellere Art Film sind. Es werden außerdem die Properties „Ist Regisseur von“, „Gewann Oscar für“ und „Ist Schauspieler in“ definiert. Es sind sogenannte *ObjectProperties*, da sie Subjekte mit Objekten verknüpfen. „Wurde veröffentlicht“ ist dagegen eine sogenannte *DatatypeProperty*, da sie Subjekte mit Literalen verknüpfen. Als Definitionsbereich von Properties „Ist Regisseur von“, „Gewann Oscar für“ und „Ist Schauspieler in“ ist die Klasse „Person“ festgelegt, der Wertebereich ist die Klasse „Film“. Definitionsbereich von „Wurde veröffentlicht“ ist wiederum „Film“, während als Wertebereich der Datentyp „Date“ definiert ist.

Neben dieser Möglichkeit führt RDFS außerdem noch weitere Konstrukte ein. So lassen sich zum Beispiel Ressourcen mit der vordefinierten Eigenschaft `rdfs:label` mit einem menschenlesbaren Namen versehen. Auch RDFS lässt sich im Turtle-Format serialisieren, wie Listing 3 zeigt.

```
:Film     rdf:type owl:Class .
:Kurzfilm rdf:type owl:Class ;
          rdfs:subClassOf :Film .
:Person   rdf:type owl:Class .

:istRegisseurVon rdf:type owl:ObjectProperty ;
                 rdfs:domain :Person ;
                 rdfs:range :Film .
:wurdeVeroeffentlicht rdf:type owl:DatatypeProperty ;
                      rdfs:domain :Film ;
                      rdfs:range xsd:Date .
```

Listing 3: Darstellung von RDF- und RDFS Definitionen im Turtle-Format

Die Definition aus Listing 2 wird hier durch RDFS-Elemente erweitert. So wird die Klasse „Kurzfilm“ als Subklasse von „Film“ definiert und für die Properties „istRegisseurVon“ und „wurdeVeroeffentlicht“ werden Domain und Range angegeben.

5.4 Web Ontology Language – OWL

Die Möglichkeiten von RDFS Semantik auszudrücken sind begrenzt, da beispielsweise keine negierenden Konstrukte möglich sind. Es kann zum Beispiel nicht definiert werden, dass zu einer Klasse „VegetarischePizza“ keine Pizzen gehören, die einen Belag mit Fleisch oder Fisch haben. Auch können in RDFS keine *Kardinalitäten* ausgedrückt werden, zum Beispiel um auszusagen, dass zu einer Klasse „ErstklassigeSchauspieler“ nur solche Personen gehören, die mindestens drei Oscars gewonnen haben. Mit der Web Ontology Language (OWL) wurde daher eine ausdrucksstärkere Sprache für die Erstellung von Ontologien entwickelt (Hitzler, Krötzsch, Parsia, Patel-Schneider, & Rudolph., 2009). OWL setzt auf RDFS auf und erweitert es um neue Konstrukte zur genaueren Beschreibung von Klassen, Eigenschaften und Beziehungen von Elementen einer Ontologie. Dadurch kann eine genauere und ausführlichere Semantik der beschriebenen Domäne festgelegt und die automatische Erschließung von abgeleitetem Wissen durch Reasoning verbessert werden. Mit OWL gibt es beispielsweise die Möglichkeit, die Klassenzugehörigkeit von Instanzen anhand bestimmter Kriterien festzulegen. Ein Beispiel ist die Klassendefinition der „VegetarischenPizza“, wobei nur Pizzen zu dieser Klasse gehören, die keinen Fisch und kein Fleisch als Belag haben. Ein *Reasoner* könnte so bei entsprechender Modellierung automatisch ableiten, welche Pizzen zu dieser Klasse gehören, d.h. vegetarisch sind oder nicht. Durch die stärkere Expressivität von OWL steigt allerdings auch die Komplexität des *Reasonings*, was bei großen Datenmengen zu Problemen bei der Skalierbarkeit führen kann. Diese Mitteilung gibt daher Reasoning nicht als singuläre Technologie vor, sondern ermöglicht herstellerspezifische Auswertung von herstellerunabhängigen / -übergreifenden semantischen Modellen.

5.5 SPARQL

Die SPARQL Protocol and RDF Query Language (SPARQL) ist eine Anfragesprache für RDF (Prud’hommeaux & Seaborne, 2008). Mit SPARQL lassen sich Anfragen auf RDF- Graphen ausführen und Teilinformationen effizient aus den RDF-Daten extrahieren. Die Grundstruktur von RDF basiert, wie unter Kapitel 5.1 beschrieben, auf Aussagen in Form von Subjekt, Prädikat und Objekt, die einen Graphen bilden. Mit SPARQL lassen sich nun Graphmuster beschreiben, die ähnlich wie in RDF aus Tripeln bestehen. Die Graphmuster können jedoch an der Stelle von Subjekt, Prädikat oder Objekt auch Variablen enthalten. Bei einer Anfrage mit einem Graphmuster wird nun im RDF-Graphen jedes Tripel auf Übereinstimmung mit einem Graphmuster überprüft und, im Fall einer Übereinstimmung, der entsprechende Wert in die Variable eingesetzt. Die Syntax von SPARQL ist ähnlich der von SQL (Structured Query Language), dem Anfragestandard für relationale Datenbanken. Die Anfrage wird von einem Befehl eingeleitet, gefolgt von einer WHERE-Klausel mit entsprechenden Graphmustern. Modifikatoren können das Ergebnis sortieren und begrenzen. Ein Beispiel für eine solche Anfrage an die oben beschriebene Filmdatenbank wäre eine Anfrage nach allen Filmen, in denen der Regisseur des Films auch Schauspieler des Films war und für die Regie einen Oscar bekommen hat und die vor 1970 veröffentlicht wurden.

Die SPARQL Spezifikation definiert zudem ein HTTP basiertes Protokoll (SPARQL Protocol for RDF), das die Struktur von Anfragen an einen SPARQL Endpoint, d.h. einen semantischen Datenspeicher, der SPARQL unterstützt, festlegt. Dadurch wird ein standardisierter Zugriff auf semantische Daten ermöglicht, der auch von den meisten Systemen unterstützt wird. In der Version 1.1 von SPARQL wird auch der schreibende Zugriff auf RDF-Graphen per SPARQL Update standardisiert. Weiterhin standardisiert SPARQL 1.1 mit dem Graph Store HTTP Protocol einen REST Webservice zur Verwaltung von RDF-Graphen (Ogbuji, 2012). Es definiert verschiedene Operationen zum Erstellen, Löschen und Modifizieren von RDF-Graphen. SPARQL dient in einem semantischen Portalsystem dem standardisierten Zugriff auf semantische Daten und wird auch zum Schreiben semantischer Daten in den Datenspeicher verwendet.

6 Klassifikationsmodell

Das Klassifikationsmodell ist die Domänenontologie für den öffentlichen Verkehr, die den Kern der hier vorgestellten Ontologien darstellt. Im Klassifikationsmodell werden die zentralen Konzepte und Beziehungen der Domäne des öffentlichen Verkehrs abgebildet. Dabei dienen die Schnittstellenbeschreibungen der TRIAS-Schnittstellen, definiert in (VDV-Schrift 431 Teil 2, 2013) als Grundlage. Die Basisstrukturen und Konzepte wurden aus den TRIAS-Schnittstellen übernommen und im Klassifikationsmodell semantisch modelliert. Damit kann die TRIAS-Architektur (VDV-Schrift 430, 2013) unter Nutzung der vorliegenden Ontologien um semantische Funktionalität erweitert werden und die Dienste sind mit dem Klassifikationsmodell kompatibel. Eine prototypische Umsetzung einer solchen semantischen Erweiterung einer TRIAS-Architektur ist in der VDV-Schrift 430-1 beschrieben. Über die TRIAS-Schnittstellenbeschreibungen hinaus umfasst das Klassifikationsmodell Konzepte, die teilweise in Siri, Transmodel oder IFOPT definiert wurden (CEN, TS 15531 Part 1), (CEN, EN 12896:2006, 2006), (CEN, EN 28701:2012, 2012). Wie in Kapitel 4.1 beschrieben, sind Referenzen auf diese Schemata in der Ontologie entsprechend gekennzeichnet.

Die folgenden Kapitel beschreiben einige Kernelemente des Klassifikationsmodells und einige grundlegende Designentscheidungen, um einen schnellen Überblick zur Verwendung der Ontologie und ihrer Konzepte zu geben. Es werden nicht alle Details der Ontologie beschrieben, hierzu wird auf die HTML-Dokumentation der Ontologie verwiesen.

6.1 Modellierung von Zeit

Die Modellierung von Zeit in Ontologien kann sehr komplex, aber auch sehr pragmatisch gelöst werden. Im komplexen Fall würden Uhrzeiten, Kalenderdaten und Zeitdauern einzeln als Klassen und Instanzen modelliert werden. Im Klassifikationsmodell wurde jedoch der pragmatische Ansatz gewählt, der die Datentypen aus den TRIAS Schnittstellen übernimmt und damit mit der Schnittstelle kompatibel bleibt. Die entsprechenden XML-Datentypen xsd:dateTime und xsd:duration werden im Klassifikationsmodell als Datentypen für Kalenderdaten und Zeitdauern genutzt. Diese Datentypen können übernommen werden, da sie als Datentypen für DatatypeProperties in RDFS und OWL genutzt werden können. Die Verwendung von xsd:duration in RDFS/OWL ist vom W3C zwar nicht empfohlen und problematisch, da der Wertebereich nicht wohldefiniert ist. Der Datentyp wird dennoch verwendet, weil die Kompatibilität mit den TRIAS-Schnittstellen als wichtiger eingestuft wird.

Einzelne weitere Klassen ergänzen die Modellierung von Zeit. Zur Abbildung von Tagen wird im Klassifikationsmodell eine Klasse DayType eingeführt, wie in Abbildung 20 dargestellt. Davon abgeleitet existiert eine Klasse Weekday, die die Instanzen der verschiedenen Wochentage enthält, mit den jeweiligen Referenzen auf ihre Entsprechungen in der TPEG-Tabelle.

Ebenfalls von DayType abgeleitet gibt es die Klasse Holiday, die Feiertage modelliert. Auch die Feiertage werden von der TPEG-Tabelle Nr. 34 übernommen und referenzieren über eine rdfs:seeAlso Annotation darauf. Verschiedene Instanzen sind hier beispielsweise PublicHoliday oder ReligiousHoliday.

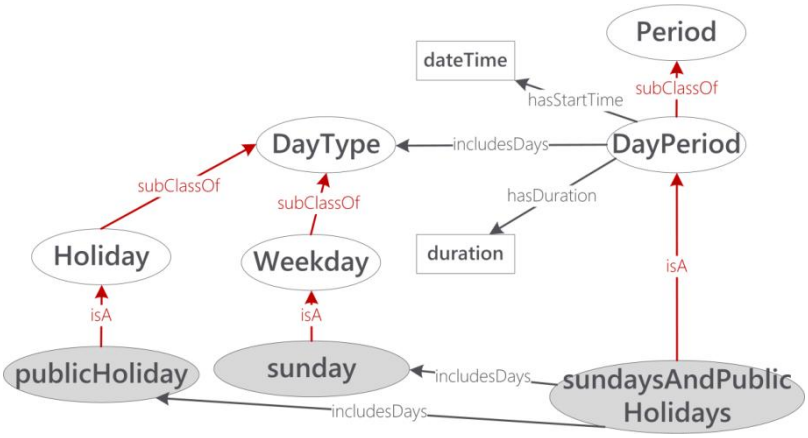

Abbildung 20: Modellierung der Klassen DayType und DayPeriod

Die Klasse Period stellt Zeiträume dar. Davon abgeleitet gibt es die Klasse DayPeriod, die Zeiträume in Angabe von Tagen definiert. Das Klassifikationsmodell kann hier die Semantik einer Menge an verschiedenen Tagen nachbilden. Instanzen einer DayPeriod sind beispielsweise MondayToFriday oder EveryDay, abgeleitet aus der TPEG PTI Tabelle Nr. 34. Die *ObjectProperty* includesDays bildet dann die Bedeutung der Zeiträume ab, indem angegeben wird, welche Typen von Tage ein solcher Zeitraum enthält, beispielsweise enthält sundaysAndPublicHolidays die entsprechenden Instanzen der Klassen Weekday und Holiday. Diese genauere Definition über die Property includesDays kann nicht immer angegeben werden, beispielsweise für den Zeitraum SchoolDays, der eine weitere Instanz der Klasse DayPeriod ist. Durch die Property hasStartTime kann, wie in der WeekdayTimePeriodStructure, der Beginn einer DayPeriod festgelegt werden, während hasDuration genutzt wird um die Dauer des Zeitintervalls zu beschreiben.

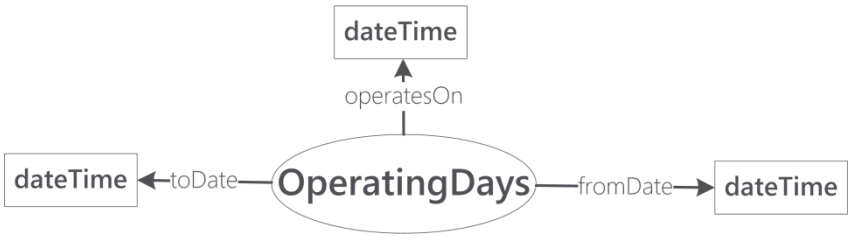

Abbildung 21: Modellierung der Klasse OperatingDays

Die Klasse OperatingDays modelliert Verkehrstage in einem Zeitraum. DatatypeProperties geben Beginn und Ende des Betriebszeitraums als xsd:dateTime an. Die DatatypeProperty operatesOn bekommt als Domain OperatingDays und als Range xsd:dateTime zugewiesen. Die Tage, die Verkehrstage sind, können dann mit operatesOn an einen OperatingDays-Zeitraum gehängt werden.

6.2 Modellierung von geographischen Entitäten

Das Klassifikationsmodell bildet Orte auf verschiedene Arten ab, wie Abbildung 22 zeigt. Dabei werden zunächst Entitäten modelliert, die synonym zu einem Ort verwendet werden können, indem die Klassen StopPoint, Locality, StopPlace und PointOfInterest als Subklassen einer allgemeinen Klasse Site (Ort, Stätte) definiert werden. Eine Site und damit alle ihre Subklassen, haben eine Location (Standort), die ihnen mit der Property hasLocation zugewiesen wird. Ein Standort, modelliert durch die Klasse Location, kann wiederum auf verschiedene Arten beschrieben werden. Auch in den TRIAS Schnittstellen wird eine Geoposition als primäre Beschreibung eines Ortes genutzt. Eine Geoposition als GPS-Position wird durch die DatatypeProperty geo:asWKT angegeben. Ein Standort kann auch durch eine Adresse dargestellt

werden. Mittels der Property hasAddress wird auf die Klasse Address verwiesen, die zur Angabe von Ort, Postleitzahl, Straßennamen und Hausnummer DatatypePropertys nutzt.

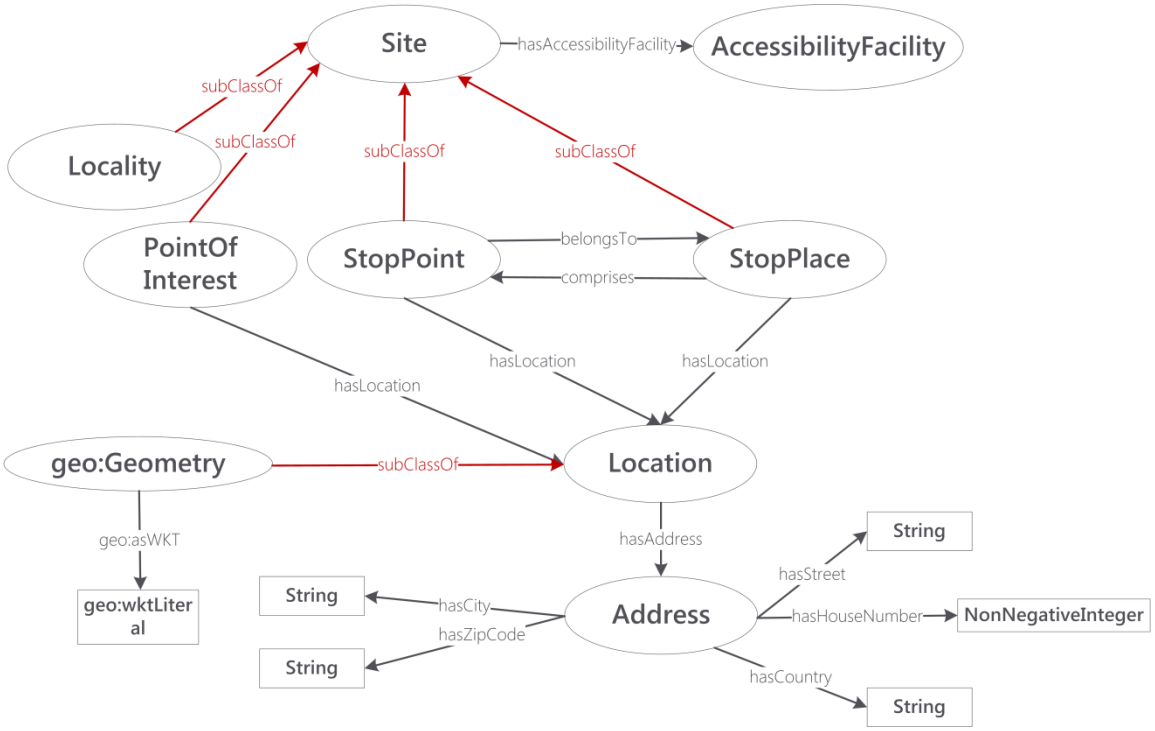

Abbildung 22: Modellierung von geographischen Entitäten und Ortsbezug.

Eine Location (Standort) kann aber auch beide Beschreibungsformen für Standorte in sich vereinen, da auch einer Adresse eine Geoposition zugeordnet werden kann. Geokoordinaten werden mit Hilfe des GeoSPARQL Standards modelliert ((OGC), 2011). Eine Location ist dazu eine Subklasse der GeoSPARQL Klasse geo:Geometry. Eine Instanz dieser Klasse für eine konkrete Location verweist über die Property geo:asWKT auf ein Literal vom Typ geo:wktLiteral, das die Geokoordinaten der Location im Well Known Text Format enthält (ISO 19125-1:2004 , 2004). Mit Well Known Text lassen sich einfache Geokoordinaten darstellen, aber auch komplexere geografische Geometrien wie Polygone. So können auch die Geometrien von Objekten wie Haltestellen mit mehreren Haltepunkten dargestellt werden. Die ObjectProperty hasLocality kann zur Abbildung der Referenzbeziehung auf eine Ortschaft genutzt werden.

Es ist aus Sicht des Klassifikationsmodells auch zulässig, dass eine Location mehrere Geopositionen haben kann – dies ist beispielsweise bei Zuordnung von Geopositionen zu Adressen der Fall, da durch eine Adresse ein größerer Bereich abgedeckt wird als durch eine präzise Geoposition. Umgekehrt kann eine Location auch mehrere Adressen haben, auch hierfür finden sich Anwendungsfälle, beispielsweise bei Nutzung verschiedener Adressschemata. Die Sicherstellung der Konsistenz (d.h. korrekte Zuordnung von Adressen zu GPS-Daten, keine Zuordnung verschiedener Adressen zu einer Koordinate etc.) obliegt dabei dem entsprechenden System, das semantische Daten erzeugt. Die Abbildung des Namens des Ortes, im XML-Schema über LocationName gegeben, ist im Klassifikationsmodell einfach über die Nutzung von rdfs:label zu lösen.

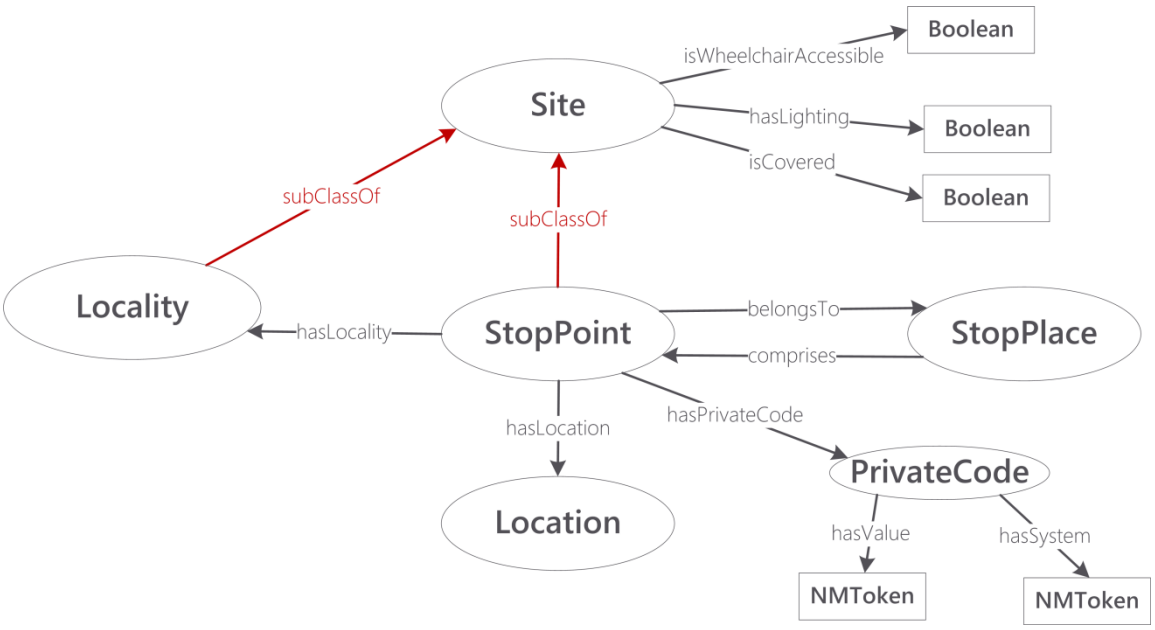

Abbildung 23: Modellierung von Haltepunkt und Haltestelle

Haltestelle und Haltepunkt werden durch die Klassen StopPlace und StopPoint modelliert. Im Modell wird die Abhängigkeit von Haltepunkt und Haltestelle durch die ObjectProperties comprises (beinhaltet) und belongsTo (gehört zu) modelliert. Per ObjectProperty hasLocality (hat Ortschaft) kann eine Ortschaft zugeordnet werden. Die Eigenschaften von Haltestellen und Haltepunkten, die in der TRIAS Schnittstelle per Boolean angegeben werden, werden im Modell ebenfalls über die DatatypeProperties isWheelchairAccessible, hasLightning und isCovered modelliert. Diese sind definiert mit Range Boolean und Domain Site, so dass theoretisch auch weitere Orte bzw. Gebäude diese Eigenschaften haben können.

6.3 Modellierung von Status

Verschiedene Entitäten sind nicht statisch, sondern dynamisch und können daher einen Status haben, der ihren aktuellen Zustand beschreibt. Dazu gehört beispielsweise der Status einer Störung oder der Status eines Services.

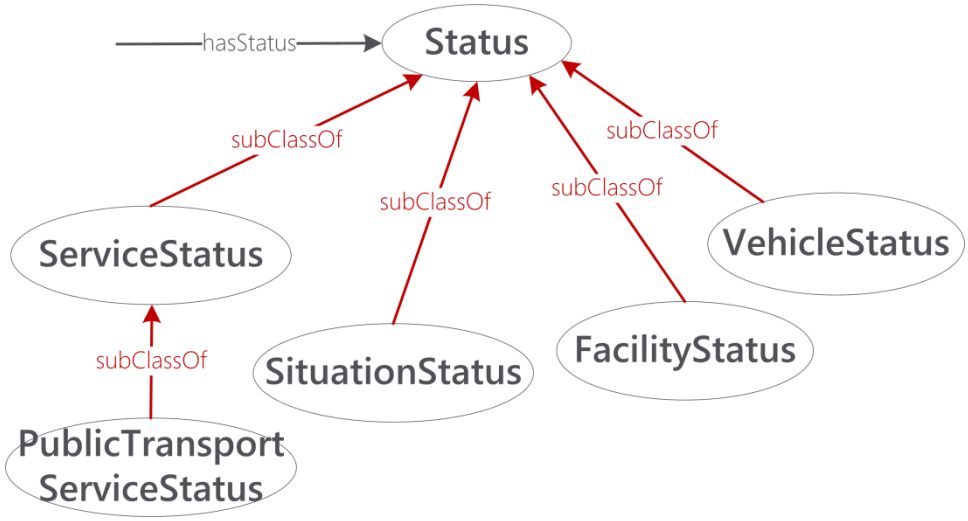

Abbildung 24: Status von verschiedenen Entitäten

In der Ontologie werden Statuselemente über jeweilige Status-Klassen modelliert und darüber den Entitäten zugeordnet. Die übergeordnete Klasse dazu ist die Klasse Status, während

entsprechende Subklassen für jeweils genauer definierte Statuselemente spezifiziert werden. Die Property hasStatus (hat den Status), deren Range (Wertebereich) die Klasse Status ist, kann genutzt werden, um einer Entität ihren Status zuzuweisen. Für die Subklassen, die den Status bestimmter Entitäten modellieren, können entsprechende, abgeleitete Properties mit genauer spezifiziertem Domain und Range genutzt werden. Ein Beispiel dafür ist die Klasse SituationStatus, die den Status einer Störung abbildet (siehe auch Kapitel 6.7), dementsprechend wurde die ObjectProperty hasSituationStatus als Subproperty von hasStatus modelliert. Deren Domain ist eine Störung, die diesen Status haben kann und der Range wird auf die Klasse SituationStatus eingegrenzt. Weiterhin werden die Klassen FacilityStatus, VehicleStatus und ServiceStatus definiert, wobei letztere wiederum zu PublicTransportServiceStatus spezialisiert wird.

6.4 Modellierung einer Reise (Trip)

Eine Verbindung, oder Reise im öffentlichen Verkehr, wird durch die Klasse Trip abgebildet. Dauer, Entfernung und Anzahl der Umstiege wird ebenso angegeben, wie die Start- und Ankunftszeit. Die verschiedenen Etappen eines Trips werden über TripLegs dargestellt, entsprechend der TRIAS Schnittstellen. Die Start- und Ankunftszeit des Trips wird durch die DatatypeProperties hasStartTime und hasEndTime angegeben, die Entfernung durch hasDistance und die Dauer der Reise durch hasDuration unter Nutzung des xsd:duration Datentyps. Die Klasse OperatingDays wird von der Klasse Trip referenziert, um die Verkehrstage anzugeben. Die Anzahl Umstiege wird durch die DatatypeProperty hasInterchanges modelliert. Um die verschiedenen möglichen Arten von Etappen innerhalb einer Reise abzubilden, wird zunächst die Klasse TripLeg eingeführt, von der zusätzlich verschiedene Sub-Klassen abgeleitet werden. Sowohl TimedLeg als auch InterchangeLeg und ContinuousLeg werden durch Subklassen der Klasse TripLeg modelliert. Entsprechend der TRIAS Schnittstellen stellt ein TimedLeg eine Fahrt mit einem ÖV-Verkehrsmittel dar, ein InterchangeLeg den Umstieg zwischen zwei solcher Fahrten und ein ContinuousLeg die Nutzung individueller Verkehrsmittel oder einen Fußweg.

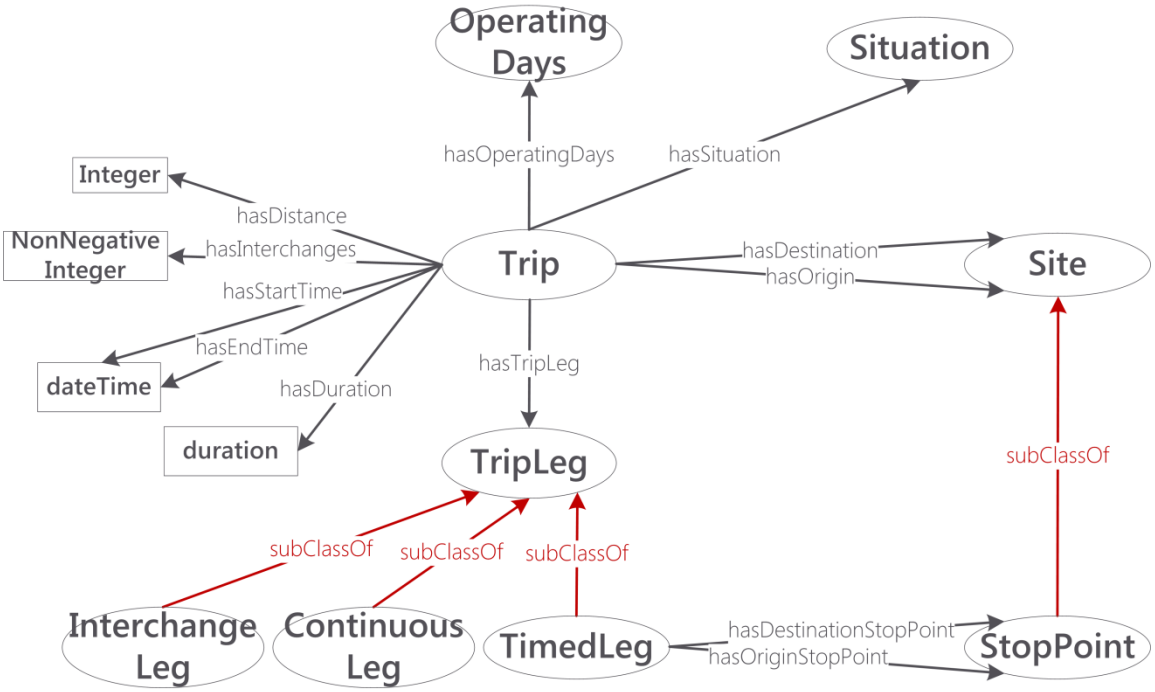

Abbildung 25: Modellierung eines Trips

Eine Reise hat einen Ursprung (modelliert über die Property hasOrigin) sowie ein Ziel (hasDestination). Beides wird durch eine Site dargestellt, welches eine abstrakte Stätte ist und eine Location, also einen definierten Ort, haben kann. Ein TimedLeg hingegen, ist eine Etappe

eines Trips, die mit einem ÖV Verkehrsmittel zurückgelegt wird und hat daher einen Ursprungshaltepunkt (hasDestinationStopPoint) und einen Ziel-Haltepunkt (hasOriginStopPoint). Diese beiden Properties sind als Subproperties von hasOrigin und hasDestination modelliert. Dies bildet ab, dass es sich um eine spezifischere Eigenschaft handelt, ebenso wie ein Haltepunkt (StopPoint) eine Spezialisierung einer Site ist und einer Haltestelle zugeordnet werden kann.

6.5 Modellierung von Services

Im Klassifikationsmodell wird das Konzept eines „Services“, einer Dienstleistung aufgegriffen und in einer Klasse Service modelliert. Diese Klasse bildet generell alle Arten von Services ab. Ein Service wird allgemein von einer Entität angeboten bzw. eine Entität erbringt einen Service/eine Dienstleistung. Dies wird dargestellt durch die Properties providedBy (angeboten von) und provides (anbieten), die invers zueinander sind und entsprechend als inverse Properties modelliert wurden.

Für providedBy wird hier kein Wertebereich angegeben, sondern hierdurch wird ermöglicht, dass verschiedene Entitäten Services erbringen können. Als Konsequenz hat provides keine Domain. providedBy wiederum hat die Domain des Services explizit angegeben. Da es sich bei provides um eine inverse Property handelt, muss hier die Angabe eines Ranges nicht explizit gemacht werden. Per Reasoning wird dies aus der Angabe der Domain für die inverse Property geschlossen.

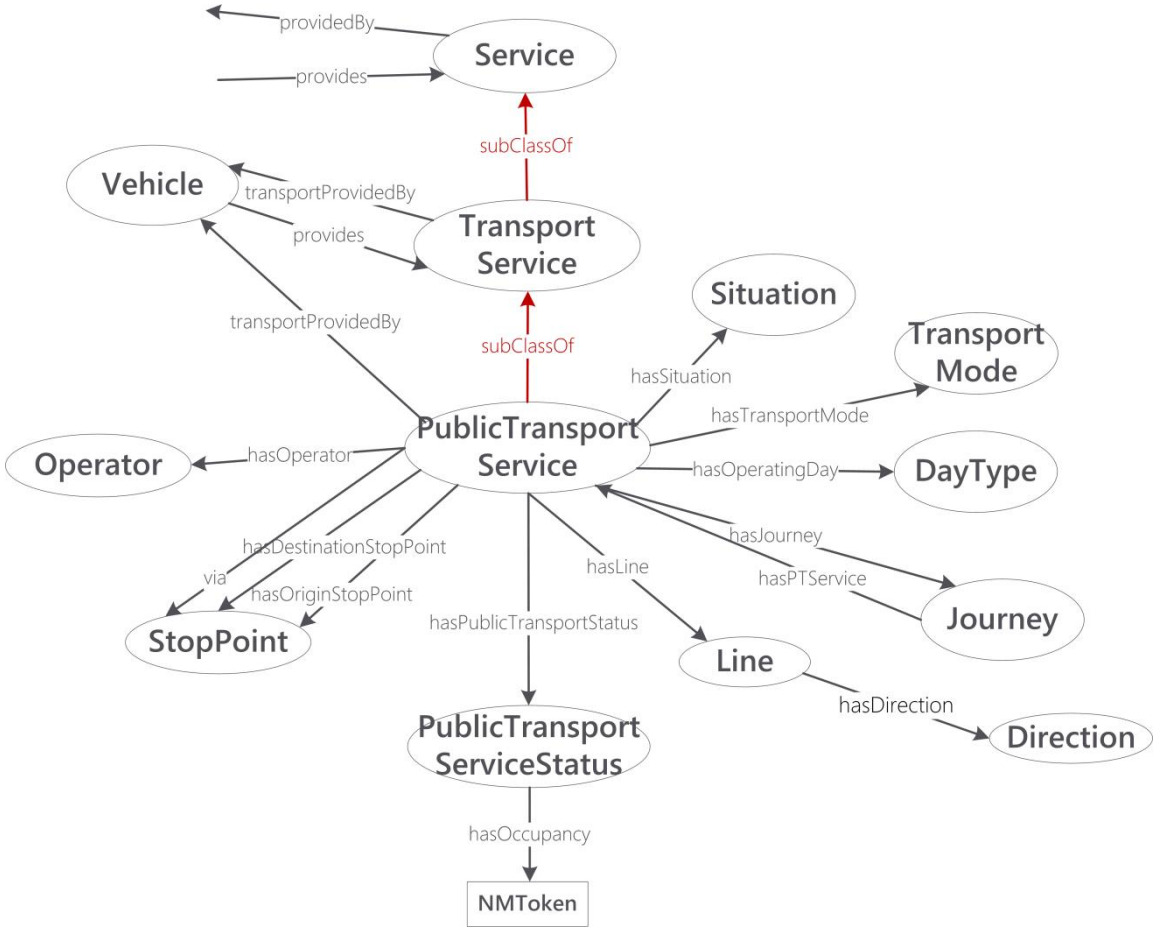

Abbildung 26: Modellierung von Services

Im Klassifikationsmodell wird die Klasse TransportService als Subklasse der Service Klasse definiert. Hier kann nun die Entität, die einen TransportService erbringt, genauer spezifiziert werden. Es handelt sich dabei um ein Fahrzeug, d.h. um die Klasse Vehicle. Daher wird für die Property providedBy eine Subproperty transportProvidedBy definiert, deren Domain und Range genauer spezifiziert sind als die der übergeordneten Property. Die Klasse PublicTransportService

ist eine weitere Spezialisierung. Die Properties hasOperatingDay und transportProvidedBy, bilden den OperatingDay und das Fahrzeug ab. Die Referenz auf eine Reise wird mit hasJourney ebenso abgebildet, wie die Referenz auf die Linie (hasLine), und darüber die Richtung (hasDirection). Mit hasTransportMode und hasOperator werden der Transportmodus und der Betreiber referenziert. Start und Ziel werden als StopPoints mit den Properties hasOriginStopPoint und hasDestinationStopPoint referenziert. Der Status eines PublicTransportService wird als PublicTransportServiceStatus abgebildet und die Property hasOccupancy modelliert dabei die Auslastung. Die Auslastung selbst wird in einer Klasse Occupancy definiert, die feste Instanzen hat. Störungen können über hasSituation referenziert werden.

6.6 Modellierungen von Transportmodi

Die Art der genutzten Verkehrsmittel wird als Transportmodus im Klassifikationsmodell abgebildet. Es existiert im Klassifikationsmodell eine Klasse TransportMode, die die Art der Fortbewegung darstellen kann. Diverse Arten der Fortbewegung werden als Subklassen modelliert und dabei klassifiziert in Arten der individuellen Fortbewegung und der Nutzung des öffentlichen Verkehrs durch Nutzung der Klassen IndividualTransportMode und PublicTransportMode.

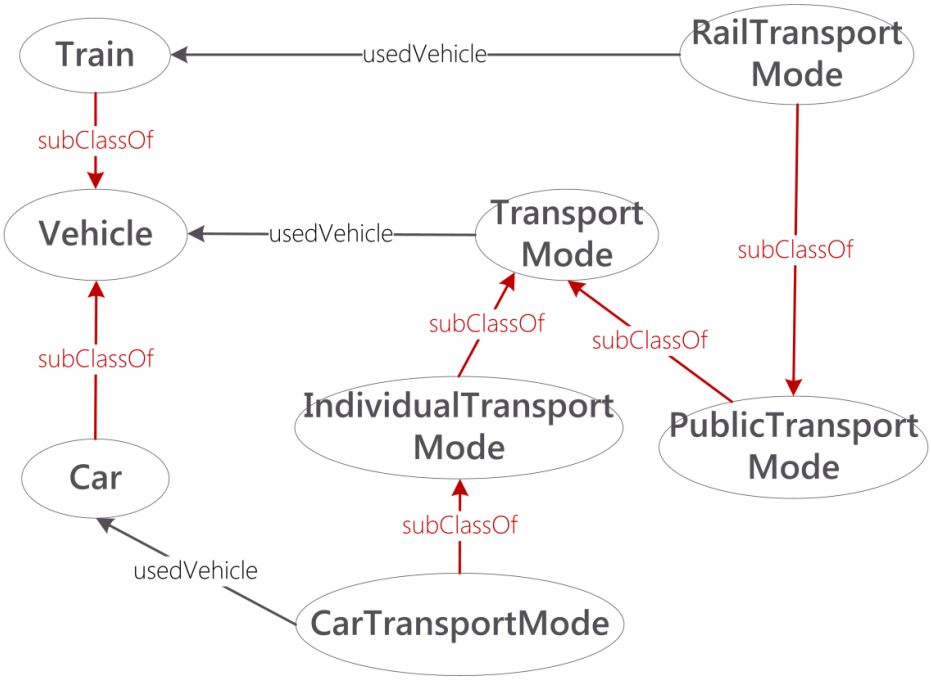

Abbildung 27: Modellierung von Transportmodi, Zusammenhang zu Fahrzeugen

Das genutzte Fahrzeug wird hier über die Property usedVehicle angegeben. Dabei wird für speziellere TransportMode-Klassen direkt angegeben, welche Fahrzeuge genutzt werden. Hierzu wird das Konstrukt der owl:restriction genutzt, um die möglichen Instanzen der Klasse RailTransportMode im Beispiel auf solche zu begrenzen, die einen Zug als Fahrzeug nutzen. Die Typen von Fahrzeugen - d.h. die Subklassen der Vehicle-Klasse - wurden dabei von den Enumerations aus Trias_ModesSupport abgeleitet, so dass für einen TransportMode ein entsprechender Fahrzeugtyp vorhanden ist.

6.7 Modellierung von Störungen

Eine Störung ist im Klassifikationsmodell als Klasse Situation modelliert. Eine Störung hat einen Grund - ein Ereignis, das die Störung hervorruft. Dies wird durch die Klasse Event dargestellt. Es wird eine Klassenhierarchie von Events definiert, die den Definitionen von Events in den TPEG Tabellen 18 bis 22 folgen.

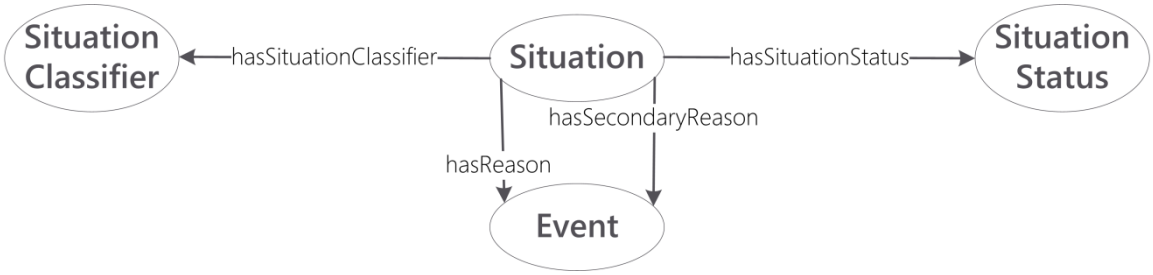

Abbildung 28: Überblick: Modellierung einer Störung

Die Gruppierung der Klassen SituationClassifier und SituationStatus werden, bis auf den Grund einer Störung, durch entsprechende gruppierende Klassen dargestellt. Die Gruppierung in Klassifikator- und Status-Gruppe wird hierbei direkt von der TRIAS-Schnittstelle übernommen. Ein SituationStatus ist dabei abgeleitet von der allgemeinen Status Klasse, siehe auch Abbildung 29. Die Elemente, die den Status einer Störung beschreiben, können an einen SituationStatus aggregiert werden. Die einzelnen Statuselemente stellen die Zustände verschiedener Eigenschaften einer Störung dar, beispielsweise ihren Verlauf (SituationProgress). In Siri werden hier Enumerations der einzelnen möglichen Zustände gebildet. Im Klassifikationsmodell werden die einzelnen Zustände durch Instanzen der jeweiligen Klassen dargestellt. Es wird außerdem festgehalten, dass die Klassen nur genau diese Instanzen enthalten, so dass nicht frei Zustände definiert werden können.

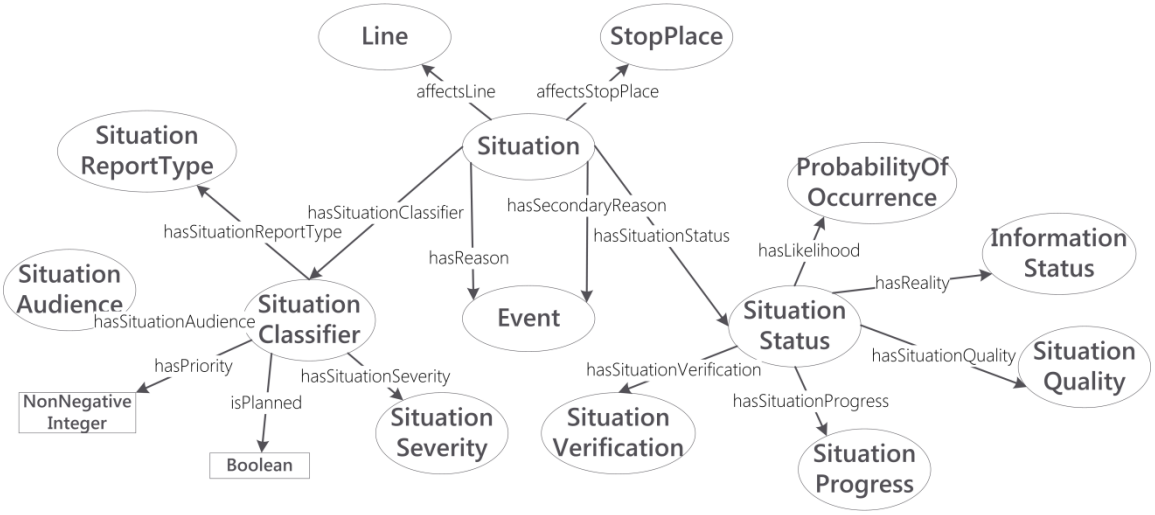

Abbildung 29: Modellierung einer Störung

Im Fall des SituationClassifier verhält es sich ebenso, dargestellt in Abbildung 29. Eine Störung kann verschiedene Dinge beeinflussen – ganze Linien, Haltestellen o.ä. Dies wird durch die Property affects abgebildet, die für die entsprechenden, möglicherweise beeinträchtigten Entitäten spezialisiert wird, zum Beispiel als affectsLine und affectsStopPlace, wie Abbildung 29 zeigt.

6.8 Modellierung einer Linie

Im Klassifikationsmodell wird eine Klasse Line erstellt, die eine Linie abbildet.

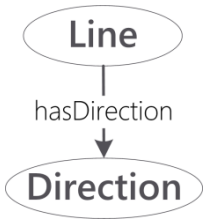

Abbildung 30: Eine Linie mit Richtung.

Die Property hasLine kann dazu genutzt werden, einer Entität eine Linie zuzuordnen. Die Domain von hasLine ist nicht festgelegt, da verschiedene Entitäten in Frage kommen. Der Klasse Line kann über die Property hasDirection eine Direction - eine Richtung zugeordnet werden.

6.9 Modellierung einer Fahrt

Die Klasse Journey modelliert eine Fahrplanfahrt. Sie referenziert einen PublicTransportService durch die Property hasService, siehe Abbildung 31.

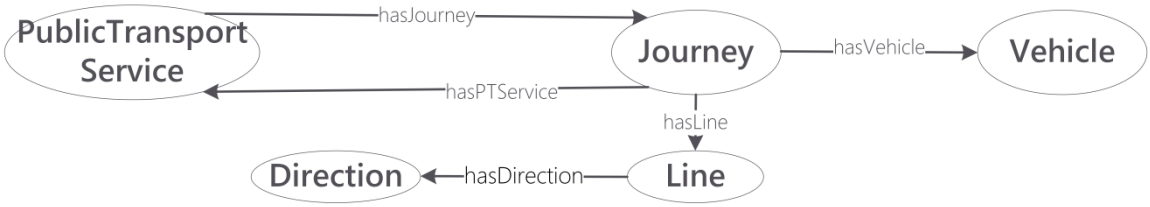

Abbildung 31: Modellierung einer Fahrt als Klasse Journey

Die Unterstrukturen des Service, so z. B. ServiceOrigin oder OperatorRef sind für die Service-Klasse bereits vorgesehen, siehe Kapitel 6.5. Eine Fahrt kann allerdings auch einer Line mit Richtung (Direction) direkt zugeordnet bekommen (über die Property hasLine), ebenso wie ein Fahrzeug über die Property hasVehicleVehicle.

7 Interaktionsmodell

Das Interaktionsmodell dient der einfachen Erweiterung des Klassifikationsmodells und des Kontextmodells um weitere Ontologien. Hier können zusätzliche Relationen und Klassen abgebildet werden, die für den jeweiligen Anwendungsfall benötigt werden. Das Interaktionsmodell ist dabei eine Art Klammer um alle genutzten Ontologien. Technisch gesehen werden im Interaktionsmodell die einzelnen Ontologien zum gemeinsamen Gebrauch per owl:import Statement importiert (siehe Abbildung 32).

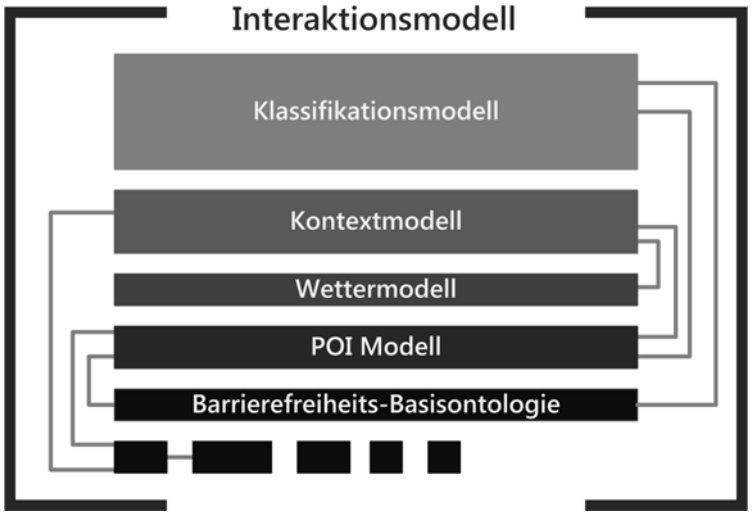

Abbildung 32: Schema des Interaktionsmodells, das die weiteren Ontologien enthält, die sich evtl. referenzieren

Das Interaktionsmodell kann eigens erstellte Ontologien ebenso zur Verfügung stellen wie externe Ontologien, die beispielsweise bestimmten Datenquellen zu Grunde liegen, die in einem Anwendungsfall gemeinsam mit den VDV-Ontologien für Fahrgastinformation genutzt werden sollen. Es wurden bereits drei solche weiterführenden Ontologien erstellt, die neben Kontext- und Klassifikationsmodell vom Interaktionsmodell importiert werden. Eine prototypische Umsetzung eines semantischen Portalsystems zeigt die Anwendung der Ontologien, inklusive der drei weiterführenden Ontologien und wird in Anhang I beschrieben. Es handelt sich dabei um eine Basisontologie für Barrierefreiheit, eine Point of Interest-Ontologie zum Einsatz in Tourismus-Szenarien und eine Wetterontologie, die im Folgenden beschrieben werden.

7.1 Point of Interest-Ontologie

Die „Point of Interest“-Ontologie dient der genaueren Beschreibung von Points of Interest (POIs), also interessanten Orten, wie zum Beispiel touristischen Attraktionen oder auch Restaurants, Cafés usw. Genutzt werden kann die Ontologie für POIs in beispielsweise touristischen Anwendungsfällen, wie in Kapitel 2.1 und Anhang I beschrieben. Die Ontologie nutzt dafür Strukturen aus der *LinkedGeoData*-Ontologie und dem „schema.org“-Schema, da Daten aus LinkedGeoData und Daten, die nach schema.org beschrieben wurden dann einfach mit der POI-Ontologie genutzt werden können. Wie in Kapitel 4.1 dargestellt, wird das jeweils per rdfs:seeAlso in der Ontologie referenziert. Die Ontologie für Points of Interest hat die Basis-URI http://vdv.de/ofi/poi#. Kern der Ontologie ist die Klasse PointOfInterest, die im Klassifikationsmodell definiert wird. Im POI-Modell werden zusätzliche Definitionen hinzugefügt, die Klasse selbst wird dabei nicht neu definiert. Das POI-Modell bezieht hier das Klassifikationsmodell mit ein. Weitere Klassen und Properties aus dem Klassifikationsmodell, aber auch dem Basismodell für Barrierefreiheit, werden im POI-Modell genutzt und neu verknüpft. In den folgenden Abbildungen werden diese gekennzeichnet, indem ihren Namen der entsprechende *Namespace* vorangestellt wird. Im Falle des Klassifikationsmodells ist der

entsprechende Namespace ipkom. Der Namespace accessibility: bezeichnet die Barrierefreiheits-Basisontologie. Im Fließtext wird diese Formulierung aus Gründen der Lesbarkeit nur genutzt, wenn aus dem Zusammenhang die Zugehörigkeit zu einer Ontologie nicht direkt ablesbar ist. In Abbildung 33 wird die Klasse PointOfInterest mit ihren Erweiterungen dargestellt.

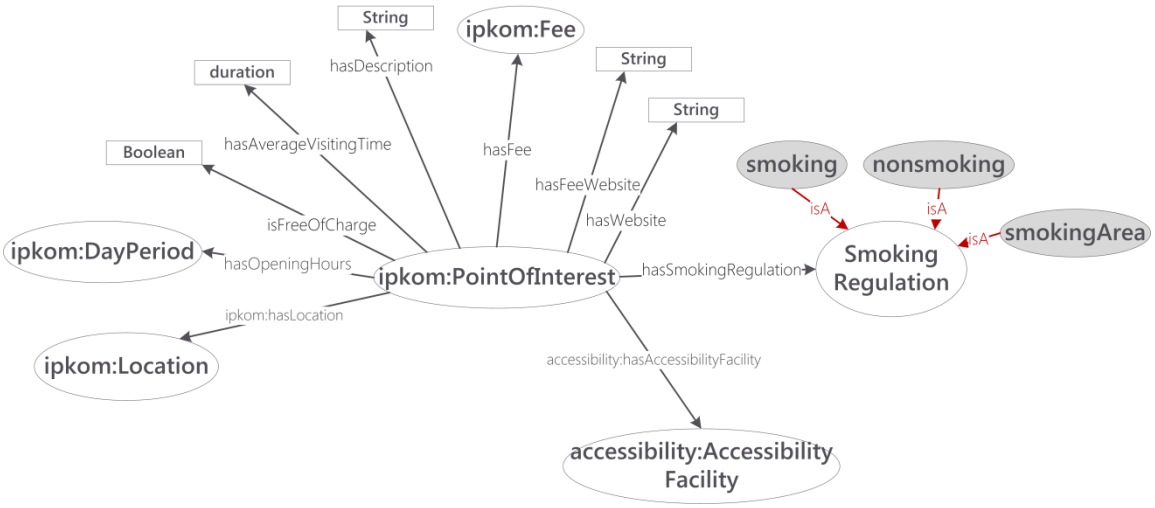

Abbildung 33: Point of Interest in der POI-Ontologie

Bereits im Klassifikationsmodell ist ein Point of Interest als Subklasse der Klasse Site definiert und es ist festgelegt, dass per hasLocation der Ort eines Point of Interest angegeben werden kann. Weiterhin wird im Basismodell für Barrierefreiheit die Property hasAccessibilityFacility definiert, mit der angegeben werden kann, welche Einrichtungen zur Barrierefreiheit vorhanden sind. Die Klasse Fee wurde ebenfalls im Klassifikationsmodell definiert. In der POI-Ontologie wird nun die Property hasFee hinzugefügt, die für einen Point of Interest ausdrücken kann, welche Gebühr bzw. welcher Preis verlangt wird. Falls die Eintrittspreise oder Gebühren nicht direkt in den Daten abgebildet werden können bzw. sollen, beispielsweise wenn diese abhängig von Uhrzeiten sind o.ä., kann mit der DatatypeProperty hasFeeWebsite die URL einer Website angegeben werden, die die Gebühren des Point of Interest auflistet, bzw. erläutert. Um direkt eine Aussage darüber treffen zu können, ob für einen Point of Interest überhaupt Eintrittspreise verlangt werden, dient die DatatypeProperty isFreeOfCharge.

Im POI-Modell kann die Klasse SmokingRegulation die Regeln für das Rauchen an einem Point of Interest definieren. Als Instanzen der Klasse sind bereits smoking, nonsmoking und smokingArea definiert, d.h. ein Point of Interest kann als Nichtraucherort oder als Raucherort ausgewiesen werden oder es wird angegeben, dass Raucherbereiche (smokingArea) vorhanden sind. Mit der DatatypeProperty hasAverageVisitingTime kann eine Einschätzung angegeben werden, wie lange die durchschnittliche Verweildauer an einem Point of Interest ist, was beispielsweise zur Berechnung einer Tour genutzt werden kann. Die weiteren DatatypeProperties hasDescription und hasWebsite, dienen der Beschreibung des Point of Interest für den Nutzer einer entsprechenden Applikation.

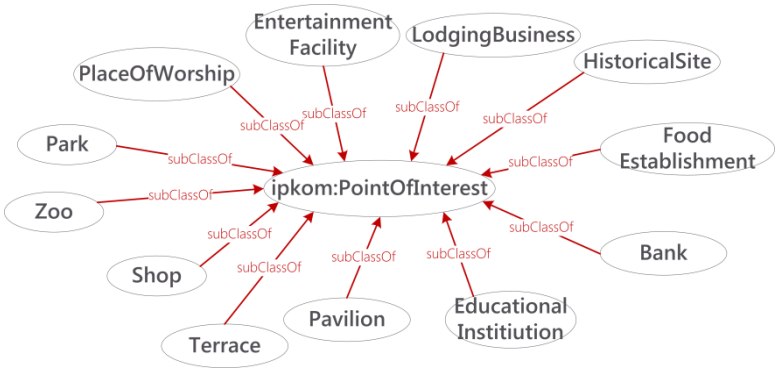

Abbildung 34: Übersicht über die Kategorien von Points of Interest

Die POI-Ontologie beschreibt verschiedene Kategorien von Points of Interest, die in Abbildung 34 gezeigt werden. Diese Kategorien entsprechen häufig Klassen, die in schema.org bzw. der LinkedGeoData-Ontologie ebenso definiert werden, teilweise sogar in beiden Schemata. Die Klassen referenzieren auf ihre Entsprechungen in anderen Schemata wie beschrieben. Die Modellierung der Point of Interest-Kategorien erhebt keinen Anspruch auf Vollständigkeit, sondern beschreibt nur den Ausschnitt aus möglichen Points of Interest, der für die beispielhafte Anwendung in einer Smart App für Touristen benötigt wurde. Die POI-Ontologie ist hier ohne weiteres um zusätzliche Kategorien erweiterbar, falls diese für andere Anwendungsfälle benötigt werden.

Einige der Subklassen von PointOfInterest werden in einer weiterführenden Taxonomie ausdefiniert und im Folgenden beschrieben. Die Klassen Zoo, Park, Terrace, Pavilion, und Bank werden allerdings nicht noch näher spezifiziert, da ihre Eigenschaften bereits durch die Properties ausreichend beschrieben werden, die für die POI-Klasse definiert wurden.

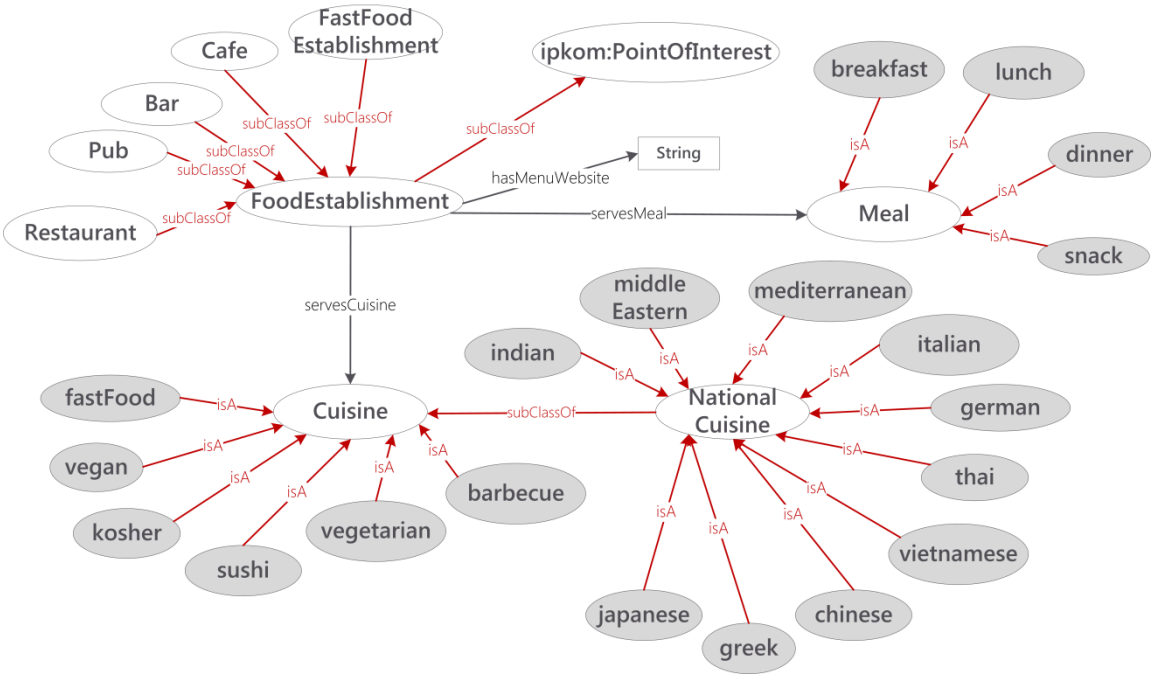

Abbildung 35: Modellierung der Klasse FoodEstablishment

Die Klasse FoodEstablishment umfasst Bars, Pubs, Restaurants, Cafes und Fast Food-Restaurants, die in den entsprechenden Subklassen definiert werden. Die DatatypeProperty hasMenuWebsite kann dazu genutzt werden, die Speisekarte zu verlinken, falls sie als Website verfügbar ist. Weiterhin kann genauer spezifiziert werden, welche Mahlzeiten serviert werden. Die Klasse Meal, mit den Instanzen breakfast, lunch, dinner und snack, dient der Einordnung, welche Mahlzeit im entsprechenden Restaurant, Café usw. eingenommen werden kann. Die Klasse Cuisine definiert die Art der Küche genauer. NationalCuisine als Subklasse definiert die verschiedenen nationalen Spezialitäten-Küchen, wobei die in Abbildung 35 dargestellten Instanzen vordefiniert sind. Hier kann die POI-Ontologie selbstverständlich erweitert werden. Instanzen der Klasse Cuisine beschreiben eine nicht länderspezifische Küche, beispielsweise wenn Fast Food oder vegane Gerichte serviert werden (vgl. Abbildung 35).

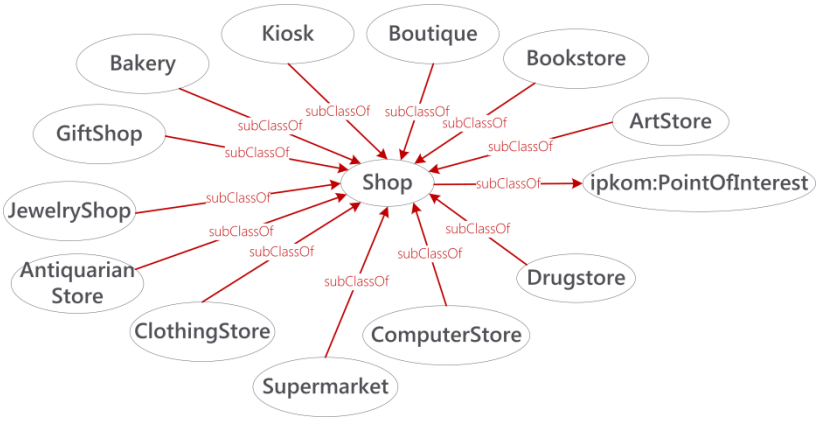

Abbildung 36: Unterkategorien von Geschäften

Die Kategorie Shop definiert mögliche Geschäfte. Verschiedene Subklassen modellieren unterschiedliche Einkaufsmöglichkeiten, wie in Abbildung 36 gezeigt. Auch für die Klasse LodgingBusiness definieren die Subklassen BedAndBreakfast, Hotel, Hostel und Motel verschiedene Unterbringungsarten, wie in Abbildung 37 dargestellt.

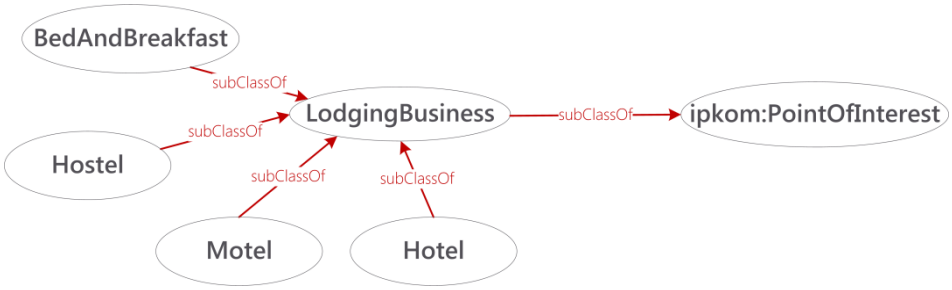

Abbildung 37: Unterbringungsarten

Historische Stätten werden durch die Klasse HistoricalSite und ihre Subklassen definiert, wie Abbildung 38 zeigt. Hier können für die entsprechenden Anwendungsfälle, z. B. für weitere Städte oder spezielle historische Orte weitere Subklassen von HistoricalSite hinzudefiniert werden.

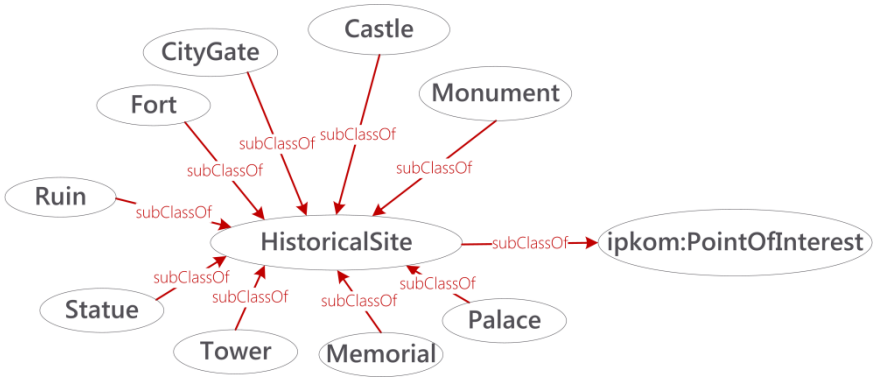

Abbildung 38: Historische Stätten als Points of Interest

Die Klasse EducationalInstitution umfasst Bildungseinrichtungen aller Art. In der Ontologie sind die bereits die Subklassen University, School, Library und Museum definiert. Für Museen kann mit der Property hasExhibitionTopic angegeben werden, um welche Art Museum es sich handelt. Vordefinierte Instanzen der Klasse ExhibitionTopic sind art, modernArt, history, localHistory, für Heimatmuseen, naturalHistory für Naturkundemuseen und physics, wie Abbildung 39 zeigt. Auch hier kann die Klasse ExhibitionTopic ohne weiteres um zusätzliche Instanzen erweitert werden.

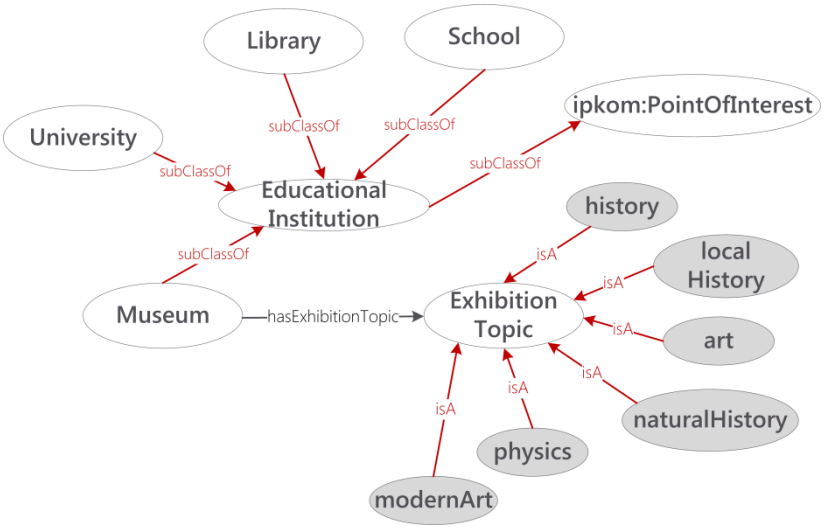

Abbildung 39: Bildungseinrichtungen in der POI-Ontologie

Gebetsstätten, Kirchen usw. werden in der Klasse PlaceOfWorship zusammengefasst, wie Abbildung 40 darstellt. Als Subklassen wurden die Klassen Synagogue, Chapel und Church modelliert, ohne Anspruch auf Vollständigkeit. Hier kann die Ontologie je nach Anwendungsfall erweitert werden.

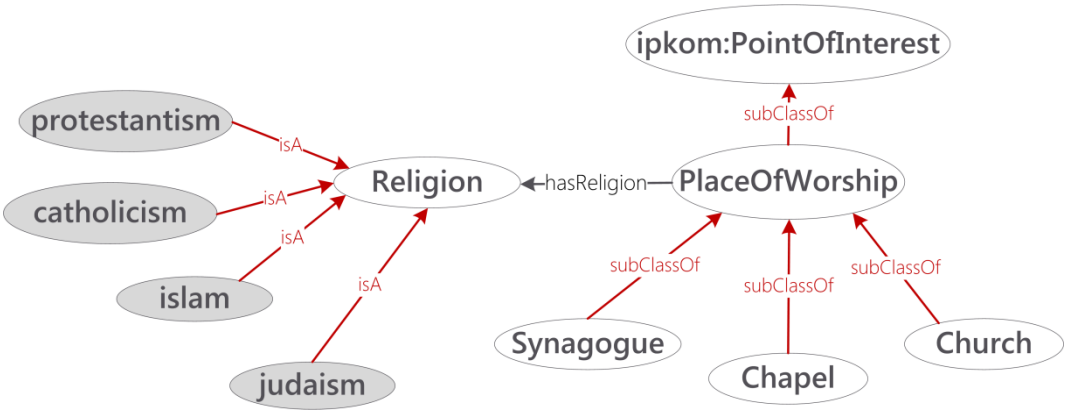

Abbildung 40: Die Klasse PlaceOfWorship für Gebetsstätten

Für eine Gebetsstätte kann mit der Property hasReligion angegeben werden, welcher Religion sie angehört. Die in Abbildung 40 dargestellten Instanzen der Klasse Religion dienen als Beispiel, wie verschiedene Glaubensbekenntnisse abgebildet werden können, falls nötig.

Die Klasse EntertainmentFacility umfasst Einrichtungen, die der Unterhaltung dienen. Im Modell sind die Subklassen OperaHouse, Cinema, Theatre, DanceClub, ConcertHall und Casino abgebildet, wie Abbildung 41 zeigt. Ein Unterhaltungsprogramm für eine solche Einrichtung wird von der Klasse Program beschrieben. Ein solches Programm umfasst meist verschiedene Programmpunkte, die als ProgramElement modelliert werden. Je nach Programm umfasst ein Programmpunkt verschiedene Unterpunkte, ProgramItems. Ein ProgramItem kann ein Kinofilm, eine Oper, ein Theaterstück oder ein Musical sein, wobei diese Klassen je nach Bedarf auch erweitert werden kann. In der POI-Ontologie ist dazu definiert, dass jedes ProgramItem einem Terminplan folgt. Dies wird durch die Property hasSchedule unter Nutzung der im Klassifikationsmodell definierten Klasse OperatingDays modelliert (so wird der Kinofilm „Der Hobbit“ beispielsweise Donnerstag bis Samstag um 19 Uhr, 20 Uhr und 21 Uhr gezeigt, Sonntags jedoch nur um 20 Uhr). Hier zeigt sich, wozu die Klasse ProgramElement verschiedene Items enthalten kann, da für das genannte Beispiel zwei ProgramItems mit verschiedenen Spielzeiten festgelegt werden müssen.

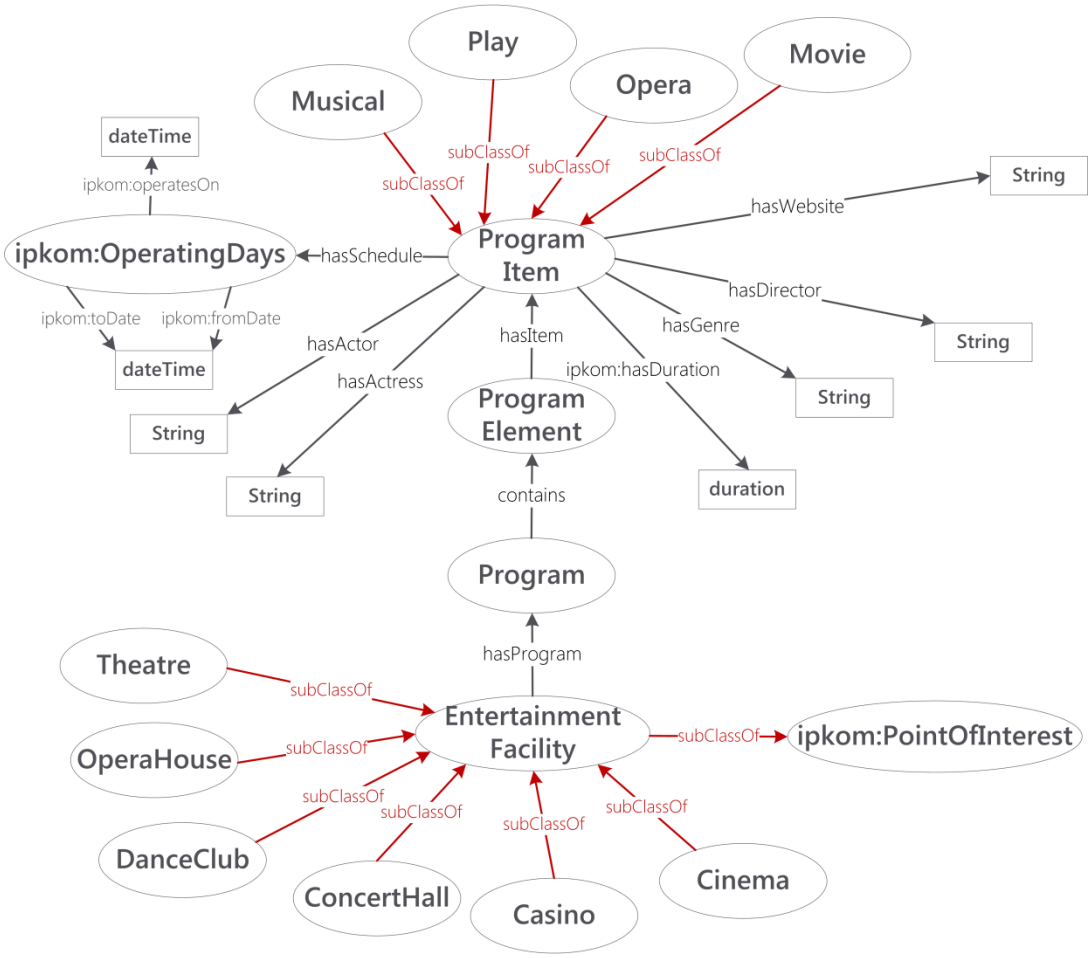

Abbildung 41 Die Klasse EntertainmentFacility und dazu gehörende Modellierung

Die Properties hasWebsite und hasDuration können hier wiederverwendet werden, um den Programmpunkt genauer zu beschreiben. Weiterhin wurden die spezialisierten DatatypeProperties hasDirector, hasActor, hasActress und hasGenre eingeführt.

7.2 Wetterontologie

Die Wetterontologie kann dazu genutzt werden, aktuelles Wetter zu beschreiben. Abbildung 42 gibt einen Überblick über die Klassen, die sie abbildet. Die Wetterontologie hat die Basis-URI http://vdv.de/ofi/weather. Zentrale Klasse ist die Klasse Weather, die eine Wetterlage an einem bestimmten Ort zu einer bestimmten Zeit beschreibt. Eine Beschreibung des aktuellen Wetters in natürlicher Sprache kann über die Property hasWeatherDescription angegeben werden. Zum aktuellen Wetter gehört weiterhin eine Beschreibung des Windes, des Niederschlags (Precipitation), der Temperatur, der Luftfeuchtigkeit, der Bewölkung und der Sichtweite (z. B. bei Nebel). Die entsprechenden Klassen und Properties sind in Abbildung 42 dargestellt.

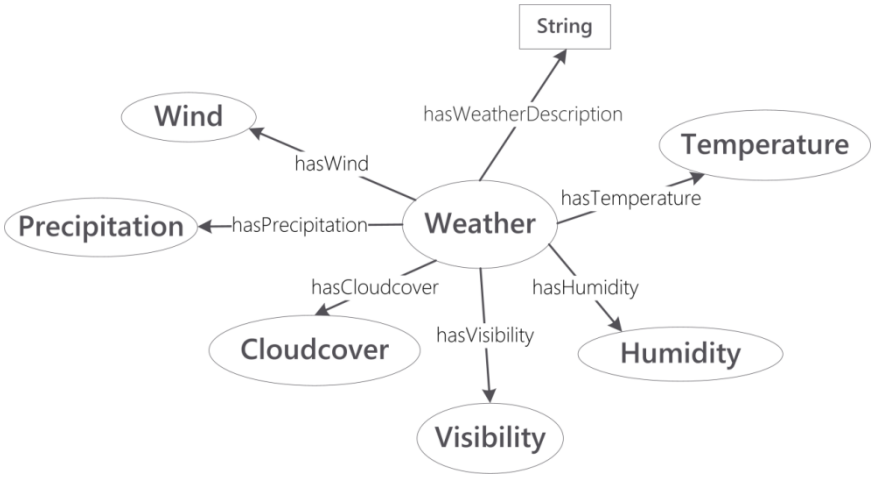

Abbildung 42: Wetterontologie, Überblick.

Für Niederschläge wird die Klasse Precipitation noch weiter ausdefiniert, wie Abbildung 43 zeigt. Es kann durch die Subklassen DayPrecipitation und CurrentPrecipitation zwischen dem Niederschlag für einen gesamten Tag und für den aktuellen Moment unterschieden werden. Der Niederschlag selbst wird durch einen Wert angegeben, der durch die DatatypeProperty hasPrecipitationValue angegeben werden kann. Für Menschen verständlicher ist eine Einordnung in Kategorien wie „viel Niederschlag“ oder „wenig Niederschlag“, was durch die Klasse PrecipitationCategory modelliert wird.

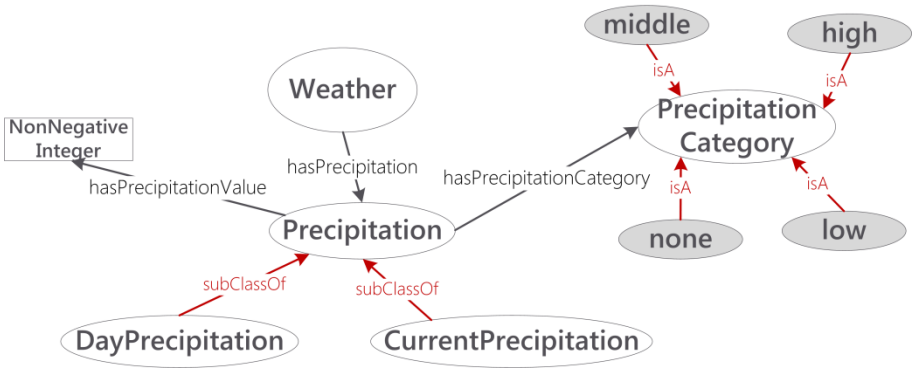

Abbildung 43: Wetterontologie, Klasse Precipitation

Die Windverhältnisse sind mit ähnlicher Struktur modelliert, wie Abbildung 44 zeigt. Hier kann die Windgeschwindigkeit als Wert in km/h oder mph angegeben werden, durch die jeweiligen DatatypeProperties.

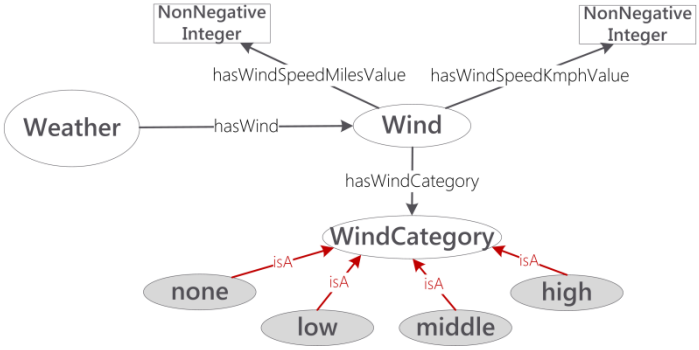

Abbildung 44: Wetterontologie, Klasse Wind

Die Klasse WindCategory erlaubt die Angabe der Windgeschwindigkeit in groben Kategorien wie „kein Wind“ bis zu „hohe Windgeschwindigkeit“. Für die Klassen Visibility und Cloudcover werden jeweils nur die entsprechenden Werte angegeben, wie Abbildung 45 und Abbildung 46 zeigen.

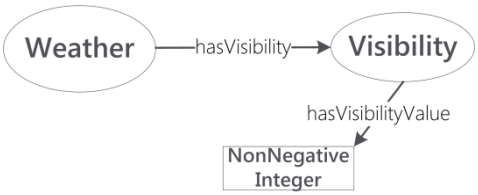

Abbildung 45: Die Klasse Visibility

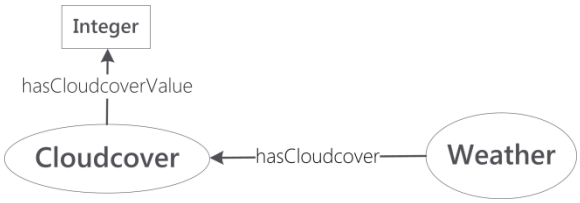

Abbildung 46: Die Klasse Cloudcover

Die Luftfeuchtigkeit, abgebildet in der Klasse Humidity, wird ebenso modelliert. Der Wert kann über die DatatypeProperty hasHumdityValue direkt angegeben werden.

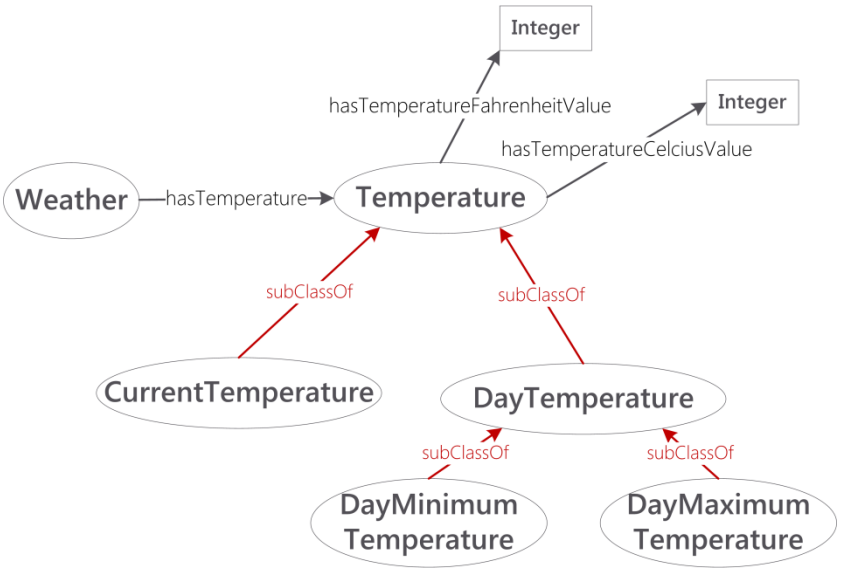

Abbildung 47: Die Modellierung von Temperatur in der Wetterontologie

Die Temperatur kann, ebenso wie der Niederschlag, als aktuelle Temperatur in der Klasse CurrentTemperature oder als Tagestemperatur in der Klasse DayTemperature angegeben werden. Diese wiederum kann in Tageshöchst- und Tagestiefsttemperatur unterschieden werden. Die Temperaturwerte können in Fahrenheit oder Celcius angegeben werden, wie Abbildung 47 zeigt.

7.3 Barrierefreiheit – Basisontologie

Die Basisontologie für Barrierefreiheit ermöglicht es, grundlegende Einrichtungen, die der Barrierefreiheit von Orten dienen, zu beschreiben. Sie dient ebenso dazu, die Anforderungen von Personen an Barrierefreiheit abzubilden. Die Ontologie ist für die Illustration der Möglichkeiten einer Modellierung von Barrierefreiheit gedacht, nicht jedoch als vollständiges Modell zu sehen. Für eine umfassende Modellierung sollte auf die Ergebnisse vorangegangener Projekte, wie beispielsweise BAIM und BAIMplus, genauer eingegangen werden. Für die Demonstration der Anwendung solcher Modelle in den in Anhang I beschriebenen Kommunikationsdiensten genügt die Ontologie wie hier beschrieben. Die Ontologie hat die Basis-URI http://vdv.de/ofi/accessibility. Abbildung 48 zeigt die Kernkonzepte der Ontologie. Die Klasse AccessibilityFacility bildet Einrichtungen ab, die die Barrierefreiheit betreffen. Dies sind nicht nur Einrichtungen, die Barrierefreiheit gewährleisten, sondern auch solche, die sie eventuell behindern, beispielsweise Treppen oder Fußwege. Die Properties hasLength und hasSlope beschreiben die Eigenschaften von Rampen, Treppen oder Fußwegen.

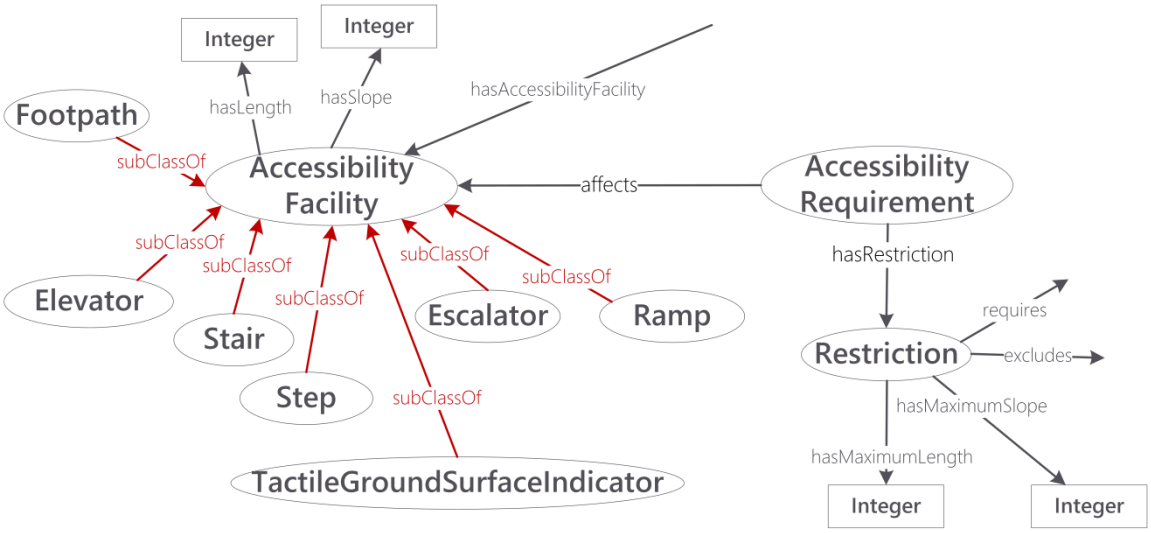

Abbildung 48: Basisontologie für Barrierefreiheit

Die Klasse AccessibilityRequirement wiederum beschreibt Anforderungen an Barrierefreiheit, wie sie beispielsweise Personen haben können. Eine Anforderung betrifft (affects) eine AccessibilityFacility. Eine solche Anforderung wird durch eine Beschränkung formuliert, durch die Klasse Restriction. So kann modelliert werden, dass eine bestimmte AccessibilityFacility benötigt (beispielsweise ein Aufzug) oder ausgeschlossen (beispielsweise eine Treppe) wird. Durch hasMaximumLength und hasMaximumSlope können direkte Einschränkungen auf Länge oder Neigung von Rampen, Fußwegen etc. formuliert werden. Die Basisontologie für Barrierefreiheit wird im Kontextmodell referenziert um den Kontext eines Fahrgastes zu beschreiben, siehe Kapitel 8.2.2.

8 Kontextmodell

Ein Kontextmodell wird dazu genutzt, den möglichen Kontext eines Nutzers abzubilden und Kontextdaten zu strukturieren, um dann darauf aufbauend auf die aktuelle Situation zu schließen. Ein kontextadaptives System versucht, auf aktuelle Situationen zu reagieren. Das hier vorgestellte Kontextmodell soll einen Rahmen bieten, der es ermöglicht, im öffentlichen Personenverkehr kontextadaptive Anwendungen für Fahrgäste umzusetzen.

8.1 Kontextmodellierung mit Ontologien

Der Kontext für kontextadaptive Anwendungen kann auf viele verschiedene Arten modelliert werden. Einen Überblick geben interessierten Lesern Strang & Popien (Strang & Popien, 2004) und Bettini et al. (Bettini, et al., 2010). Das Kontextmodell für den öffentlichen Verkehr, das im Folgenden beschrieben wird, wurde als Ontologie modelliert. Durch die Modellierung von Kontext als Ontologie wird die gemeinsame Nutzung mit dem Klassifikationsmodell und dem Interaktionsmodell unterstützt.

8.2 Eine Kontextontologie für den öffentlichen Verkehr

Kontextdefinitionen sind meist sehr umfangreich, so definieren Dey et al. beispielsweise Kontext als: *„…any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves“* (Dey, Abowd, & Salber, 2001). Um solch allgemein gehaltene Kontextdefinitionen für die Modellierung einer Kontextontologie zu verdichten, ist es nötig, auf den Einsatzzweck des Modells zu fokussieren und die möglichen Kontextinformationen einzugrenzen, die die Ontologie abbilden können soll. Abbildung 49 zeigt die Dimensionen von Kontext für kontextadaptive Anwendungen im öffentlichen Verkehr (Kühn, Keller, & Schlegel, 2011). Das hier vorgestellte Kontextmodell fokussiert, wie auch in der Abbildung 49 zu sehen ist, auf Fahrgast-bezogenen Kontext. Es ist allerdings ohne weiteres möglich, die in diesem Dokument beschriebene Kontextontologie um weitere Kontexte zu erweitern, beispielsweise um den Kontext von Einrichtungen (Haltestellen, Gebäude), oder Fahrzeugen zu erfassen.

Abbildung 49: Kontextdimensionen im öffentlichen Verkehr

Für die in diesem Dokument beschriebene Kontextontologie wird der allgemeine Kontext auf die in Abbildung 49 dargestellten Kontextdimensionen konzentriert, da diese die für kontextadaptive Anwendungen für Fahrgäste im öffentlichen Verkehr relevanten Kontextinformationen umfassen:

- Der *physikalische Kontext* umfasst dabei Informationen, die die physische Umgebung des Fahrgastest betreffen, so beispielsweise Temperatur oder Umgebungslautstärke.
- Als *Aufgabenkontext* wird der zielbezogene Kontext (die Aufgaben des Fahrgastes) verstanden. Hiermit kann abgedeckt werden, welche Ziele ein Fahrgast verfolgt und welche Aufgabenschritte noch vor ihm liegen (zu Aufgabenkontext siehe auch (Bardram, Bunde-Pedersen, & Soegaard, 2006), (Ni, Zhou, Zhang, & Heng, 2006) oder (Prekop & Burnett, 2003)).
- Die Beschreibung der Lokation eines Fahrgastes erfolgt wiederum in der Kategorie des *Ortskontexts.* Hierbei kann es sich sowohl um die Einordnung des physischen Aufenthaltsortes eines Nutzers als auch die technische Lokalisation (z. B. die Position im Netzwerk) handeln. Auch die Information, ob und wie sich der Fahrgast räumlich bewegt, fällt in diese Kategorie.
- In der Kategorie des *zeitlichen Kontexts* fallen verschiedene Zeitinformationen. Dies beinhaltet neben der Erfassung der aktuellen Uhrzeit bspw. auch eine saisonale Einteilung, Sommer- und Winterzeit oder Zeitzonen auf internationalen Reisen.
- Die jeweilig verfügbaren Interaktionstechniken können unter *Interaktionskontext* zusammengefasst werden, um in Abhängigkeit von Gerätespezifika oder Nutzerpräferenzen Interaktionsmöglichkeiten anzupassen.
- Letztendlich beschreibt der *soziotechnische Kontext* diverse Nutzereigenschaften und persönliche Präferenzen für die Domäne des öffentlichen Verkehrs. Ebenso fallen operationale und betriebliche Kontextinformationen in diese Kategorie.

Die Gewichtung und Auswahl der zu nutzenden Kontextinformationen unterscheidet sich in der Regel in Abhängigkeit des jeweiligen Einsatzszenarios. So sind bspw. Ortsinformationen besonders bei mobilen Anwendungen von Interesse. Da die Menge aller Kontextzustände prinzipiell unendlich ist, wird ein Ansatz zur Kontextmodellierung stets nur einen Ausschnitt des Kontextes erfassen können (Brezillon, 2003). Die hier vorgestellte Kontextontologie wurde daher so flexibel wie möglich gestaltet, so dass für jedes Einsatzszenario die jeweils benötigten Kontextdimensionen und spezifischeren Kontextelemente genutzt werden können. Die Erweiterbarkeit der in diesem Dokument beschriebenen Ontologie erlaubt es, Kontextinformationen, die die Ontologie bisher nicht abbildet, ohne weiteres zu ergänzen.

8.2.1 Fahrgastkontext

Die zentrale Klasse der in diesem Dokument beschriebenen Kontextontologie ist die Klasse UserContext, die in Abbildung 50 dargestellt ist. Ein Fahrgast wird in der Klasse Person modelliert und über die Property hasUserContext wird dem Fahrgast ein UserContext zugeordnet.

Die Klasse UserContext referenziert über die Property hasContextElement Instanzen der Klasse ContextElement. Ein Kontextelement stellt dabei eine Kontextinformation dar. Die Quelle der Kontextinformation kann über hasSource angegeben werden. Einem Kontextelement wird außerdem über die DatatypeProperty lastModified ein Zeitstempel zugewiesen, der markiert, wann dieses Kontextelement zuletzt geändert wurde. Dem liegt eine wichtige Designentscheidung zu Grunde: Der zeitliche Verlauf des Kontext wird hier nicht abgebildet, d.h. Kontextinformationen werden überschrieben, wenn sie sich ändern und lastModified wird dann entsprechend aktualisiert. Der UserContext bildet damit immer nur die aktuelle Situation des Fahrgastes ab. Für die identifizierten Anwendungsfälle ist diese Modellierung ausreichend. Eine Modellierung des zeitlichen Verlaufes und der Veränderung von Kontextinformationen ist deutlich komplizierter und resultiert auch in höherer Implementierungs- und Speicherverwaltungskomplexität. Nichtdestotrotz bietet die hier beschriebene Kontextontologie

die Möglichkeit, über Erweiterungen auch den zeitlichen Verlauf von Kontextinformation abzubilden, falls dies für sehr komplexe Anwendungsfälle nötig sein sollte.

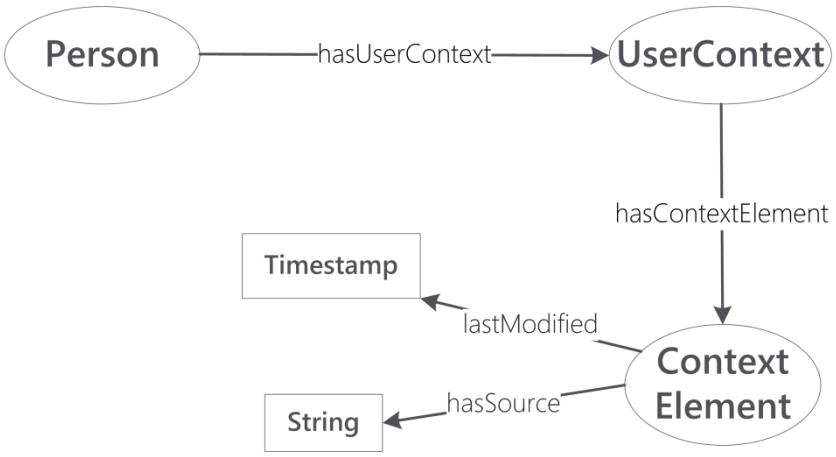

Abbildung 50: Fahrgastkontext mit Kontextelementen

8.2.2 Kontextelemente

Abgeleitet von der Klasse ContextElement werden verschiedene Spezialisierungen für Kontextelemente, so dass jede Art von Kontext, der für eine Anwendung relevant ist, modelliert werden kann. Abbildung 51 zeigt die im Kontextmodell vordefinierten Kontextelemente. Hier bietet sich die Möglichkeit, die in diesem Dokument vorgestellte Kontextontologie beliebig zu erweitern, je nach Anwendungsfall, indem weitere Kontextelemente als Spezialisierungen der Klasse ContextElement definiert werden.

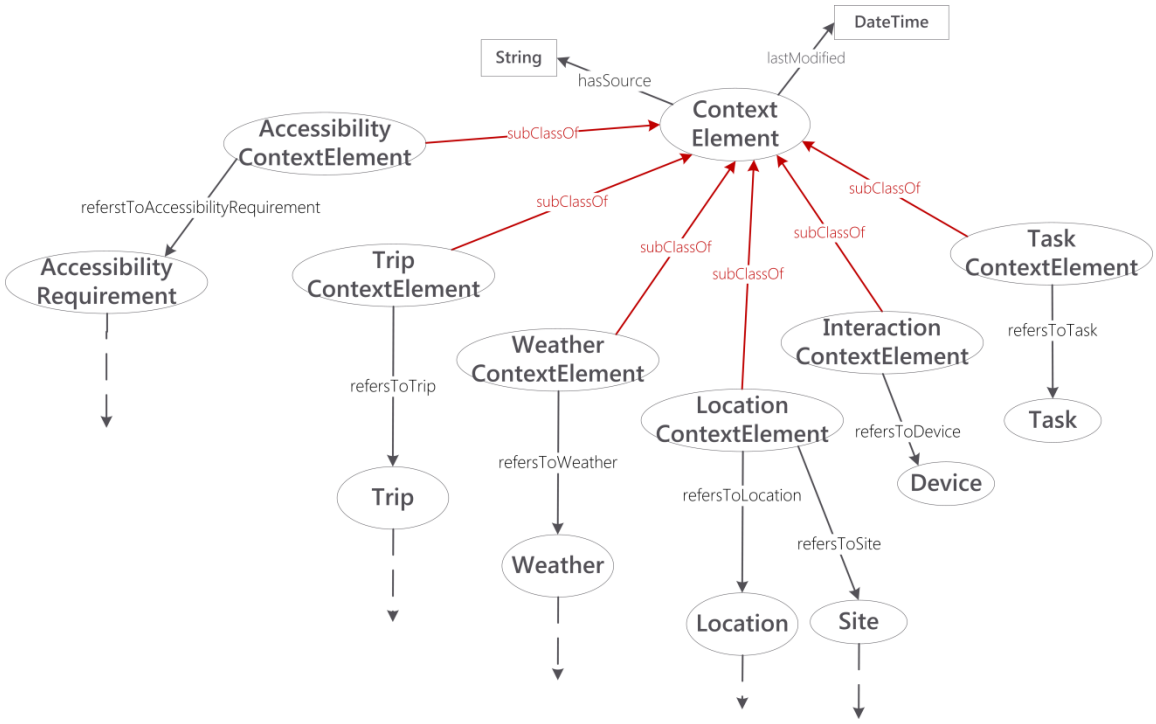

Abbildung 51: Vordefinierte Kontextelemente

Ein Kontextelement bezieht sich immer auf eine Information, die in Bezug auf den Fahrgast als Kontext des Fahrgastes zu sehen ist. Ein Beispiel hierfür ist eine Reise im ÖPV, die der Fahrgast angetreten hat. Als Trip ist die Reise oder Verbindung bereits im Klassifikationsmodell

beschrieben und da auch andere Fahrgäste diese Reise antreten können, ist sie nicht spezifisch dem Fahrgast zugeordnet. Für die Beschreibung des Fahrgastkontextes ist allerdings die angetretene oder auch geplante Reise relevant. Daher enthält der Fahrgastkontext ein Kontextelement der Klasse TripContextElement, das über die Property refersToTrip auf die entsprechende Instanz der Klasse Trip verweist. So wird der Bezug zum Fahrgastkontext hergestellt.

Abbildung 52: Erweiterung der Kontextontologie um eine weitere Kontextinformation „XYZ“

Dieses Muster, dargestellt in Abbildung 52, wird für jedes Kontextelement angewandt. Es nutzt dabei die Struktur der in diesem Dokument definierten Ontologien, nach denen das Interaktionsmodell sowohl Klassifikations- als auch Kontextmodell und weitere, speziellere Ontologien kapselt. Die ursprüngliche Information, die als Kontextinformation modelliert werden soll, wird in einer der Ontologien modelliert, die das Interaktionsmodell zusammenfügt. Im hier vorgestellten Kontextmodell wird nur noch eine entsprechende Spezialisierung der Klasse ContextElement und die spezialisierte „refersTo“ Property ergänzt, die auf die Klasse aus einer anderen Ontologie verweist. Auf diese Art ist die in diesem Dokument beschriebene Kontextontologie jederzeit einfach erweiterbar.

Folgende Kontextelemente sind in der in diesem Dokument beschriebenen Kontextontologie bereits definiert:

- AccessibilityContextElement: Anforderungen des Fahrgastes an Barrierefreiheit können hier modelliert werden, in Bezug auf die in Kapitel 7.3 beschriebene Basisontologie für Barrierefreiheit. Das Kontextelement bezieht sich auf die Klasse AccessibilityRequirement, die solche Anforderungen modelliert.
- InteractionContextElement: Der Interaktionskontext des Fahrgastes kann hier modelliert werden. Es wird auf Interaktionsgeräte Bezug genommen durch die Klasse Device. Bisher ist diese Beziehung nicht weiter ausmodelliert und kann je nach Anwendungsfall erweitert werden.
- LocationContextElement: Das Kontextelement beschreibt den Standort des Fahrgastes. Dieser kann auf verschiedene Arten angegeben werden, daher kann das Element per refersToLocation eine Instanz der Klasse Location aus dem Klassifikationsmodell referenzieren oder per refersToSite eine Instanz der Klasse Site oder von Spezialisierungen der Klasse Site – wie zum Beispiel StopPlace (siehe Kapitel 6.2). Der Standort eines Fahrgastes kann damit durch eine Adresse, GPS Koordinate oder auch durch eine Haltestelle angegeben werden.
- TaskContextElement: Das Kontextelement modelliert den Aufgabenkontext eines Fahrgastes und referenziert eine Modellierung einer Aufgabe in der Klasse Task. Auch diese Klasse kann je nach Anwendungsfall erweitert werden.
- TripContextElement: Das Kontextelement bezieht sich auf eine angetretene oder geplante Reise des Fahrgastes und referenziert die Klasse Trip aus dem Klassifikationsmodell (Kapitel 6.4).

– WeatherContextElement: Das Wetter wird durch dieses Kontextelement abgebildet. Es wird die Wetterontologie referenziert, die in Kapitel 7.2 beschrieben wurde und vom Interaktionsmodell importiert wird.

In Anhang I dieses Dokuments befindet sich die Beschreibung eines prototypischen semantischen Portalsystems, das im Rahmen des IP-KOM-ÖV Projektes implementiert wurde. Die Beschreibung enthält eine Darstellung der Implementierung der Infrastruktur für die entwickelten kontextadaptiven Dienste und dient als Beispiel für die Umsetzung eines kontextadaptiven Fahrgastinformationssystems auf Basis der vorgestellten Kontextontologie.

Anhang I (informativ)

Im Rahmen des Projektes IP-KOM-ÖV wurde ein Prototyp eines semantischen Portalsystems umgesetzt, das intelligente Kommunikationsdienste zur Verfügung stellt, die auf den Ontologien basieren, die in dieser Mitteilung dokumentiert sind. Die Architektur des Prototyps und die gesammelten Erfahrungen der Umsetzung sind nachfolgend beschrieben, um Entwicklern von intelligenten, Ontologie-basierten Kommunikationsdiensten Hinweise für Entwurf und Umsetzung solcher Dienste zu geben. Zudem werden die für den Prototyp evaluierten und eingesetzten Technologien beschrieben, zur Einordnung für zukünftige Systeme (Stand Mitte 2013).

Ein prototypisches semantisches Portalsystem

Ein semantisches Portalsystem ist in zahlreichen Varianten denkbar. Basis des hier vorgestellten Prototypen ist die TRIAS-Architektur aus (VDV-Schrift 430, 2013) und (VDV-Schrift 431, Teil 1, 2013). Es stehen daher als Komponenten die standardisierte Echtzeit-Kommunikations- und Auskunftsplattform (EKAP) und ein Portalsystem zur Verfügung. Letzteres umfasst einen Push-Dienst sowie eine Anfragesteuerung zur Steuerung des asynchronen (Push-Dienst) und Request/Response (Anfragesteuerung) Kommunikationsflusses. Ontologie-basierte Kommunikationsdienste werden in der TRIAS Architektur im Portalsystem als semantische Dienste vorgesehen, ebenso wie ein Modellserver, der die Modelle und semantischen Daten verwaltet.

Die konkrete Ausgestaltung der Architektur eines semantischen Portalsystems hängt von der Art der intelligenten Kommunikationsdienste ab, die das System zur Verfügung stellen wird. Ausgangsszenario für die hier vorgestellte prototypische Implementierung sind intelligente, Ontologie-basierte Kommunikationsdienste für Anwendungen im Tourismus. Diese Dienste sollen eine Smart App für Touristen ermöglichen, ähnlich der Smart App „Tourist Guide“ aus Kapitel 2.1 dieses Dokuments.

Die zentralen intelligenten Kommunikationsdienste des Prototypen sollen für eine gegebene Zeitspanne einem Fahrgast eine Liste von Points of Interest (POI) berechnen, die kontextadaptiv (auf seine aktuelle Situation bezogen) und innerhalb der Zeitspanne mit dem ÖPV erreichbar sind. Weiterhin sollen, ebenfalls für eine gegebene Zeitspanne, komplette Touren geplant werden, die den aktuellen Kontext des Fahrgastes, darunter seine Vorlieben, die Tageszeit, das aktuelle Wetter usw. miteinbeziehen. Abbildung 53 zeigt die Dienste, die hierfür notwendig sind. Neben den bereits erwähnten, nicht-semantischen Diensten (in grau) und einem Mehrwertdienst (ebenfalls grau) werden die semantischen Dienste grün dargestellt. Dazu gehört die Komponente **PointOfInterest und Tourenplanung**, die die zentrale Logik für die beschriebene Funktionalität umsetzt.

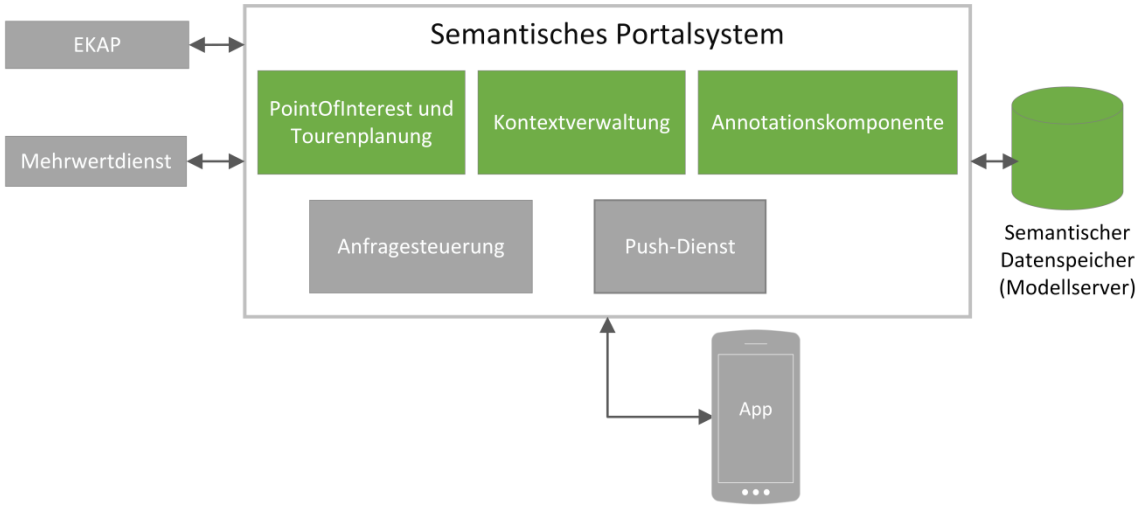

Abbildung 53: Grobe Architektur des prototypischen semantischen Portalsystems

Weiterhin wurde eine **Annotationskomponente** umgesetzt. Sie sorgt dafür, dass benötigte Daten von nicht-semantischen Diensten, wie beispielsweise EKAP Diensten oder auch Mehrwertdiensten in semantische Daten, konform mit den zu Grunde liegenden Ontologien transformiert werden.

Die **Kontextverwaltung** verwaltet Kontextquellen und stellt Kontextdaten für andere intelligente Kommunikationsdienste zur Verfügung, die mit entsprechender Kontextinformation auf die Situation des Fahrgastes reagieren. Die Kontextdaten werden nach der Kontextontologie strukturiert und als semantische Daten gespeichert. Wo Kontextdaten als nicht-semantische Daten erhoben werden, kann die **Annotationskomponente** genutzt werden, um sie in semantische Daten zu transformieren.

Nicht zuletzt wird ein **semantischer Datenspeicher** benötigt, der die semantischen Daten speichert und für die Dienste im Portalsystem per SPARQL eine Schnittstelle zur Verfügung stellt. Die Umsetzung der einzelnen Komponenten wird im Folgenden genauer beschrieben und in einer Feinarchitektur des semantischen Portalsystems verortet.

Semantischer Datenspeicher

Ein semantischer Datenspeicher stellt eine Softwarekomponente dar, die der persistenten Speicherung, Verarbeitung und dem Abruf von semantischen Daten in RDF dient. Da RDF-Daten als Aussagen, sogenannte Triple vorliegen, werden diese Komponenten als *Triple Store* bezeichnet. Ein Triple Store ist vergleichbar mit einem relationalen Datenbankmanagementsystem (RDBMS) für Datenbanken, ist allerdings optimiert für die Graphstruktur von RDF. Manche Triple Stores setzen auf relationale Datenbanken für die interne Speicherung der RDF Tripel auf, andere dagegen verwenden ein eigenes, für RDF optimiertes Speicherformat. Die meisten Triple Stores besitzen proprietäre Schnittstellen für die Verwaltung von RDF Graphen und Ontologien und das Einfügen und Abfragen von Daten einschließlich Programmierschnittstellen (Application Programming Interface – API) für gängige Programmiersprachen. Weiterhin wird im Normalfall die beschriebene Anfragesprache SPARQL für den Lesezugriff unterstützt. Manche Systeme unterstützen auch den Schreibzugriff per SPARQL Update sowie den in der Version 1.1 von SPARQL standardisierten Zugriff per Graph Store HTTP Protocol.

Einige Triple Stores integrieren zudem Reasoner, um das automatische Schlussfolgern auf den gespeicherten Daten zu ermöglichen. Meist ist die Inferenz jedoch auf RDFS und eine kleine Teilmenge von OWL begrenzt. In Tabelle 1 werden einige aktuell verfügbare Triple Stores

einander gegenüber gestellt und verglichen. Wie die Tabelle 1 zeigt, ist die Wahl des semantischen Datenspeichers anwendungsabhängig.

	Skalierbarkeit	**Datenbank**	**API**	**Reasoner**	**Lizenz**
OpenLink Virtuoso v6.1	Hoch	– Nativ	– CLI – Java (Sesame und Jena Interface) – SPARQL – REST	RDFS OWL (eingeschränkt),	Kommerziell / GPL
OWLIM	Hoch	– Nativ	– Java (Sesame und Jena Interface) – SPARQL 1.1 – REST	RDFS, OWL 2 RL und OWL 2 QL	Kommerziell
Systap Bigdata	Hoch	– Nativ	– Java (Sesame Interface) – SPARQL	RDFS+	Kommerziell / GPL
Oracle Spatial and Graph RDF Semantic Graph	Hoch	– Relational	– SPARQL 1.1 – Java (Sesame und Jena Interface)	RDFS, OWL, Regelbasiert	Kommerziell
Mulgara	Hoch	– Nativ	– Java (Jena Interface) – SPARQL – REST	Regelbasiert	GNU Free Documentation License Version 1.2
Jena Fuseki	Mittel	– Nativ – PostgreSQL – MySQL	– Java – SPARQL 1.1 – REST	OWL, RDFS, Regelbasiert	Apache License, Version 2.0
OpenRDF Sesame	Mittel	– Nativ – PostgreSQL – MySQL	– Java – SPARQL 1.1 – REST	RDFS	BSD-style license

Tabelle 1: Verschiedene semantische Datenspeicher[2]

Für eine hohe Anzahl an Tripeln (über 10 Mio.) eignen sich kommerzielle Lösungen wie *Virtuoso*, OWLIM oder Bigdata, welche native RDF Datenspeicherung, Reasoner und vielfältige Schnittstellen mitbringen. Der Open Source Triple Store *Mulgara* skaliert ähnlich gut, bringt aber nur einen Regel basierten Reasoner mit. *Jena* und *Sesame* können sehr gut für kleine bis mittlere

[2] http://www.bioontology.org/wiki/images/6/6a/Triple_Stores.pdf

Datenmengen (unter 10 Mio. Tripeln) eingesetzt werden, wobei *Jena* über einen vollständigen Reasoner und die Möglichkeit der Anbindung einer vorhandenen Datenbank verfügt. Sowohl Jena als auch Sesame sind keine reinen Triple Stores, sondern ebenfalls Java Frameworks für die Verarbeitung von RDF Daten und bieten entsprechende APIs an. Das ist vorteilhaft, wenn das semantische Portalsystem ebenfalls in Java umgesetzt wird. Die anderen Triple Stores besitzen in den meisten Fällen Schnittstellen für die Verwendung in Kombination mit Sesame bzw. Jena.

Für die Speicherung semantischer Daten im prototypischen semantischen Portalsystem wurden verschiedene der Triple Store-Implementierungen getestet. Dazu gehört **Fuseki**[3], eine Open-Source Entwicklung eines Triple Store als Apache Projekt. Fuseki wird im Rahmen von Apache Jena[4] entwickelt, einem Projekt zur Umsetzung eines Semantic Web Frameworks. Weiterhin wurde **Sesame**[5] getestet, eine Triple Store Implementierung durch Aduna[6], einer Firma mit Sitz in den Niederlanden. Sesame steht unter Open Source Lizenz zur Verfügung. Der Sesame Triple Store wurde in zwei Konfigurationen getestet: einerseits als alleinstehender Triple Store, anderseits mit einer Erweiterung um die Geo-Information-System-Komponente aus dem Open Sahara[7] Projekt.

Diese Erweiterung ergänzt den Triple Store um einen Geo-Index, so dass direkt in SPARQL-Anfragen Geo-Informations-Anfragen integriert und effizient im Triple Store ausgeführt werden können. Da die Ontologien konform mit dem OWL Standard spezifiziert vorliegen und sowohl Fuseki als auch Sesame diesen Standard unterstützen, sind beide Triple Stores ohne weiteres für ein semantisches Portalsystem auf Basis der VDV-Ontologien für Fahrgastinformation einsetzbar. Unterschiede gibt es auf Seiten des Einrichtungs- und Wartungsaufwandes der Software-Komponenten und bei der Effizienz der Anfragen. Die Anfrageeffizienz hängt allerdings stärker noch von der Hardware ab, auf der die Software läuft. Dabei ist der verfügbare Hauptspeicher entscheidend. Der Einrichtungs- und Wartungsaufwand ist bei Sesame höher als bei Fuseki. Die Erweiterbarkeit (z. B. durch die GIS-Komponente) und auch die Skalierbarkeit sind nach unseren Erfahrungen für Sesame besser zu bewerten als für Fuseki. Es ist hervorzuheben, dass durch die Verwendung von Standards wie RDF/S und SPARQL die Triple-Store Implementierungen grundsätzlich austauschbar sind.

Annotationskomponente

Die XML-Daten, die das Portalsystem und die EKAP versenden, sollen auch den semantischen Komponenten für weitere Operationen zur Verfügung stehen. Dabei soll es nicht die Aufgabe einer jeden semantischen Komponente sein, XML-Dateien neu aufzuarbeiten und gegebenenfalls die gleichen Anfragen an die anderen Komponenten wiederholt zu stellen. Daher kapselt die Annotationskomponente die Funktionalität, die nicht semantischen Daten, konform mit den zu Grunde liegenden Ontologien in semantische Daten, d.h. RDF zu transformieren. Die RDF Daten werden dann im Triple Store gespeichert und können von den semantisch arbeitenden Komponenten verwendet werden. Die Funktion der Annotationskomponente wird in Abbildung 54 schematisch dargestellt. Die Komponente arbeitet auf strukturierten XML-Daten, die durch eine XSD-Definition beschrieben sind. Weiterhin wird die Ontologie herangezogen, die die Struktur der semantischen Daten vorgibt, die entstehen sollen. Die Komponenten ermöglicht es einem Entwickler, Transformationsregeln zu definieren, die die Transformation von XML-Daten in RDF-Daten beschreiben. Es wird ebenso ermöglicht, bereits erstellte Regeln zu verwalten, um sie beispielsweise an Änderungen in der XSD-Definition oder der Ontologie anzupassen. Wird der

[3] http://jena.apache.org/documentation/serving_data/

[4] http://jena.apache.org/index.html

[5] http://www.openrdf.org/

[6] http://www.aduna-software.com/

[7] https://opensahara.com

Annotationskomponente nun eine XML-Datei übergeben, wird diese anhand der Regeln in RDF-Daten überführt und im Ziel-Triple Store gespeichert. Die Implementierung der einzelnen Schritte soll im Folgenden erläutert werden.

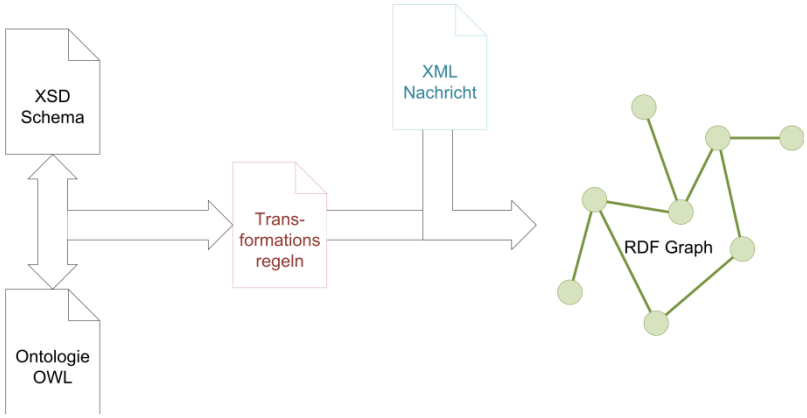

Abbildung 54: Funktion der Annotationskomponente

Die Umsetzung der Annotationsregeln direkt im Programmcode ist nicht empfehlenswert, da die Anpassung einzelner Regeln durch Änderungen am XML Schema oder an den Ontologien Änderungen am Programmcode und Neukompilieren des Gesamtprojektes nach sich ziehen würde. Die Annotationsregeln werden daher in XML-Form verwaltet. Die Verwendung des XML-Formates für Annotationsregeln stellt einen Kompromiss zwischen Lesbarkeit und maschineller Verarbeitbarkeit dar. Insgesamt ist dadurch die Möglichkeit geschaffen, eine lesbare Form von Annotationsregeln auch während der Programmlaufzeit auszutauschen, ohne dass die Komponente umprogrammiert oder neu kompiliert werden muss. Der Regelsatz wird vom Programm eingelesen und interpretiert. Jede Regel besteht dabei aus einem Header und einem Statementteil. Im Header wird definiert, an welcher Stelle eine Regel anwendbar ist. Er besteht zunächst aus dem Regelnamen, der den Namen eines XML-Knotens trägt, für welchen die Regel gilt. Weiterhin ist der Kontext im Header definiert, da bestimmte XML-Knoten an unterschiedlichen Stellen auftauchen können und unter Umständen auch anders behandelt werden sollen. Der Header enthält außerdem Strukturinformationen. Es ist vorstellbar, dass eine XML-Datei, die in einem Knoten einen zusätzlichen Unterknoten enthält, andere relevanten Daten liefert, als anders herum, daher kann auf Grund von Strukturinformationen entschieden werden, welche Regel angewandt werden soll. XML-Knoten können zudem unterschiedlich ausgewertet werden, wenn die Auswertung für einen anderen semantischen Kontext geschieht. Daher enthält der Header auch Hinweise auf den semantischen Kontext, in dem die Regel anzuwenden ist. Die auf den Header folgenden Regelstatements entsprechen einer einfachen XML-Struktur, die an die gängigen Konstrukte von Programmiersprachen angelehnt ist, wie Tabelle 2 zeigt. Sie werden zum Applikationsstart eingelesen und mit den zu transformierenden XML-Dokumenten in der Durchführung der Annotation interpretiert.

Statement	**Bedeutung**
<block>	Bildet den Rahmen für die Aneinanderreihung mehrerer einfacher und komplexer Statements. Die enthaltenen Statements werden während der Ausführung schrittweise nacheinander ausgewertet
<goup>, <godown>, <gotoroot>	Dienen der Navigation zwischen XML-Knoten innerhalb eines XML-Dokumentes. Solche Navigationsstatements können geschachtelt, oder innerhalb eines Blocks auch aneinandergereiht, angegeben werden.

Statement	Bedeutung
<foreach-child>, <foreach-named-child>	Iteriert über alle Kindknoten eines XML-Knotens.
<cond>, <then>, <else>	Bestandteile einer If-Then-Else Konstruktion.
<isempty>	Prüft, ob der aktuelle Knoten leer ist.
<hasChild>	Überprüft auf Existenz eines bestimmten Kindknotens.
<evaluate-content>	Wertet den textuell angegeben Inhalt eines XML-Knoten aus.
<ignore>	Beschreibt eine Anweisung, die den aktuellen XML-Knoten ignoriert. Dies ist sinnvoll um zum Beispiel bestimmte Knoten in Abhängigkeit von deren Inhalt von der Annotation auszuschließen.
<cancel>	Bricht den Annotationsvorgang ab, falls frühzeitig erkannt werden kann, dass die XML-Datei nur irrelevante oder sogar fehlerbehaftete Daten enthält.
<apply_rule_by_element>	Löst den Aufruf einer anderen Annotationsregel für den berechneten XML-Knoten aus. Damit kann ein separater, paralleler Annotationsvorgang gestartet werden, der einen anderen aktuellen XML-Knoten betrachtet.
<apply_rule_by_elementstructure>	Ruft für ein XML-Element eine Regel nur dann auf, wenn das Element auch der im Header der Regel deklarierten Struktur entspricht.
< apply_rule_by_rulename>	Ruft eine Regel explizit per Namen auf. Dabei muss der Regelentwickler dafür Sorge tragen, dass der Name der Regel eindeutig definiert ist.
<create-individual>, <create-literal>	Lösen ein Mapping aus. Die Ausdrücke, die innerhalb dieses Statements auftauchen, dienen dem Aufbau eines RDF-Graphen.
<split>, <patternmatch>, <string-build>	String-Operationen, die auf entsprechende Java-Funktionen verweisen.
<property>	Ein Property-Statement ist beschrieben durch die Angabe der repräsentierten Property und der Angabe einer Regel, wie das Objekt des jeweils zu erstellenden Tripels erstellt werden soll.
<coded>	Definiert den Aufruf einer in Java-Code gegossener Regel. Per URL-Klassenlader wird dann zur Auswertung die in <coded> deklarierte Klasse geladen und instanziiert. Der Regelentwickler kann damit Sonderfälle berücksichtigen, die in dem XML-Regelformat nicht so ohne weiteres möglich sind.

Tabelle 2: Mögliche Statements in Annotationsregeln

Sollen die definierten Regeln angewandt werden, sind drei wesentliche Arbeitsschritte für die Ausführungsphase der Annotationskomponenten umzusetzen:

- Datenextraktion / -selektion: Ein Teil der Daten, die in den XML-Dateien enthalten sind, sind für die Wissensbasis unerheblich und können demnach vernachlässigt werden. Nur die XML-Knoten, die tatsächlich relevante Daten enthalten, sollen ausgewertet werden. Weiterhin können Daten fehlerbehaftet sein: beispielsweise die Antwort auf eine fehlerhafte Anfrage.

Fehlerhafte Daten, sollen nicht in die Wissensbasis aufgenommen werden. In diesem Schritt werden die zu bearbeitenden Daten selektiert.

- Transformation / Aufbereitung zu semantischen Daten: Schrittweise werden mit Hilfe der Daten einzelne oder mehrere Instanzgraphen aufgebaut. Sonderfälle sind zu beachten. Auch die innere und äußere Struktur eines XML-Knotens muss für die Typbestimmung ausgewertet werden. Eine eindeutige Zuordnung von primitiven Datentypen ist notwendig.
- Aktualisierung des Triple Store: Um doppelte Einträge in der Wissensbasis zu vermeiden, wird SPARQL Update verwendet und die Erstellung von URIs eindeutig umgesetzt, so dass Elemente, die bereits beschrieben wurden, immer denselben Namen erhalten. Bestehende Daten werden allerdings, wenn Zusatzinformationen vorhanden sind, um diese erweitert.

Kontextverwaltung

Die Kontextkomponente verwaltet Kontextquellen und Fahrgastkontext. Dazu wird für jedes mobile Endgerät, das die Smart App installiert, ein Fahrgastkontext angelegt. Als Kontextquellen stehen zur Verfügung:

- Benutzereingaben, d.h. Angaben von Präferenzen,
- Sensoren des mobilen Endgerätes, wie GPS und Beschleunigungssensoren,
- die EKAP,
- sowie externe Dienste, wie zum Beispiel im Prototyp ein Wetter Web-Service.

Die Kontextverwaltung sorgt für regelmäßige Updates der Kontextdaten. Das mobile Endgerät wird dazu mit dem Push-Dienst angefragt, externe Dienste oder die EKAP werden über die Anfragesteuerung angesprochen. Die Kontextkomponente ergänzt oder aktualisiert den Fahrgastkontext, wenn sie neue Kontextdaten erhält. Der Fahrgastkontext wird dabei im Triple Store gespeichert. Anderen Komponenten, so zum Beispiel der POI- und Tourenplanung, stellt die Kontextverwaltung Fahrgastkontext per API zur Verfügung.

PointOfInterest und Tourenplanung

Diese Komponente stellt zwei Dienste zur Verfügung. Zunächst können auf Basis eines Zeitraums und eines Standortes für einen Fahrgast interessante Points of Interest berechnet werden, die innerhalb des gegebenen Zeitraums, ausgehend vom aktuellen Standort, mit den öffentlichen Verkehrsmitteln erreichbar sind. Der Fahrgastkontext wird dabei zur Filterung der POIs herangezogen. Anhand des Fahrgastkontextes, der an der Kontextverwaltung abgefragt werden kann, werden mit Kontextregeln Points of Interest aus dem Triple Store abgefragt. Weiterhin wird ein Dienst umgesetzt, der für eine gegebene Zeitspanne komplette Touren zu verschiedenen POIs berechnet. Dabei werden zunächst die POIs gefiltert und extrahiert, woraufhin diese, nach Anwendung verschiedener Ratingfunktionen, in eine Reihenfolge gebracht werden. Dabei wird der aktuelle Kontext des Fahrgastes, darunter seine Vorlieben, die Tageszeit, das aktuelle Wetter usw. miteinbezogen. Die Verknüpfung der POIs zu einer Tour erfolgt unter Nutzung der EKAP für die Verbindungsauskünfte zwischen den POIs.

Weitere Komponenten des prototypischen semantischen Portalsystems

Für die Umsetzung des Prototyps wurden einige Hilfskomponenten erstellt, sowie eine Datenbasis erzeugt. Eine Feinarchitektur der prototypischen Umsetzung wird in Abbildung 55 gezeigt, wobei die nicht-semantischen Dienste erneut grau, die semantischen Dienste grün dargestellt werden. Im Prototyp des semantischen Portalsystems wurde der Sesame-Triple Store als Datenspeicher für die semantischen Daten eingesetzt. Dazu gehört die Abbildung von Daten aus der TRIAS-Schnittstelle in semantische Daten auf Basis des Klassifikationsmodells, Kontextdaten nach dem Kontextmodell und zusätzliche Daten nach dem Interaktionsmodell und darin integrierten (Teil-)Ontologien. Als zusätzliche Datenquelle wurde ein Teil der Daten aus dem LinkedGeoData Projekt genutzt. Für den Prototypen des semantischen Portalsystems wurden die Daten für den Raum Stuttgart extrahiert und in den Sesame Triple Store mit GIS-Erweiterung überspielt, so dass die Daten für das Prototyping genutzt und für den Feldtest des Projektes IP-KOM-ÖV zur Verfügung gestellt werden können. Die Daten können unter anderem über das Protokoll SPARQL abgefragt werden. Eine Hilfskomponente zur SPARQL-Anfrage-Abstraktion dient als Abstraktionsschicht zwischen den Triple Stores und allen weiteren Komponenten die per SPARQL die semantischen Daten anfragen. Die Komponente ermöglicht es, typsichere SPARQL-Anfragen im Programmcode zu erstellen und an einen gegebenen SPARQL-Endpunkt, d.h. Triple Store zu senden.

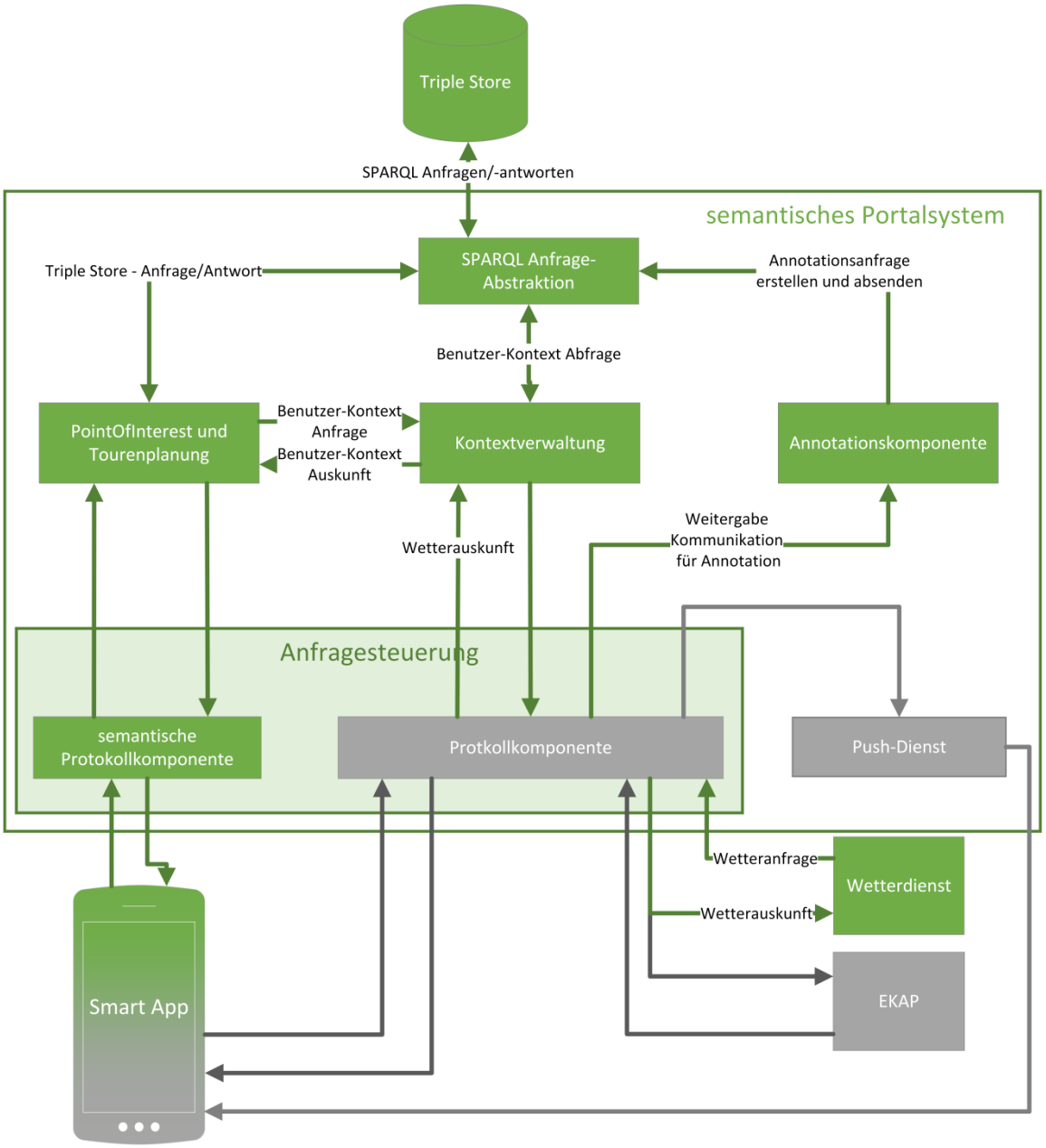

Abbildung 55: Feinarchitektur für ein semantisches Portalsystem

Um die Übertragung der Daten von den intelligenten Kommunikationsdiensten zum mobilen Endgerät zu ermöglichen, wurde die Anfragesteuerung um eine semantische Protokollkomponente erweitert, die ein Protokoll umsetzt, das die Kommunikation zwischen den Komponenten des semantischen Portalsystems und der Basisapplikation ermöglicht.

Regelwerke – Normen, Empfehlungen, Literatur

CEN, EN 12896:2006. (2006). *Reference Data Model for Public Transport.* CEN - Europäisches Komitee für Normung.

CEN, EN 28701:2012. (2012). Intelligent transport systems - Public transport - Identification of Fixed Objects in Public Transport (IFOPT, EN 28701:2012). CEN - Europäisches Komitee für Normung.

CEN, TS 15531 Part 1. *SIRI - Service Interface for Realtime Information, Part 1.* CEN - Europäisches Komitee für Normung.

CEN, TS 15531 Part 5. *SIRI - Service Interface for Realtime Information, Part 5.* CEN - Europäisches Komitee für Normung.

CEN, TS 15531, Part 2. *SIRI - Service Interface for Realtime Information, Part 2.* CEN - Europäisches Komitee für Normung.

VDV-Schrift 430. (2013). *Kundenschnittstelle - Architektur.* Köln: VDV - Die Verkehrsunternehmen.

VDV-Schrift 431 Teil 2. (2013). *EKAP-Schnittstellenbeschreibung.* Köln: VDV - Die Verkehrsunternehmen.

VDV-Schrift 431, Teil 1. (2013). *Systemarchitektur EKAP.* Köln: VDV - Die Verkehrsunternehmen.

VDV-Mitteilung 7023. (2012). Kommunikation im ÖV (IP-KOM-ÖV)- Szenarien & Personen sowie deren Anforderungen. Köln: VDV - Die Verkehrsunternehmen.

Open Geospatial Consortium. OGC GeoSPARQL - a Geographic Query Language for RDF data. 2012. http://www.opengeospatial.org/standards/geosparql.

World Wide Web Consortium. OWL Web Ontology Language. 2004. http://www.w3.org/TR/owl-features/.

World Wide Web Consortium. OWL 2 Web Ontology Language, 2012. http://www.w3.org/TR/owl2-overview/.

World Wide Web Consortium. Resource Description Framework. 2004. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

World Wide Web Consortium. RDF Vocabulary Description Language 1.0: RDF Schema. 2004. http://www.w3.org/TR/rdf-schema/.

World Wide Web Consortium. SPARQL Query Language for RDF. 2008. http://www.w3.org/TR/rdf-sparql-query/.

World Wide Web Consortium. SPARQL 1.1. 2012. http://www.w3.org/TR/sparql11-overview/.

(OGC), O. G. (2011). *GeoSPARQL - A Geographic Query Language for RDF Data.* Open Geospatial Consortium (OGC).

Bardram, J., Bunde-Pedersen, J., & Soegaard, M. (2006). Support for activity-based computing in a personal computing operating system. *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems* (S. 211-220). New York, NY, USA: ACM Press.

Beckett, D., & Berners-Lee, T. (2011). *Turtle - Terse RDF Triple Language.* Tech. rep., World Wide Web Consortium (W3C).

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American , 284* (5), 34-43.

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., et al. (2010). A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing , 6* (2), 161-180.

Brezillon, P. (2003). Using context for supporting users efficiently. System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on, (S. 9 pp.-).

Brickley, D., & Guha, R. (2004). *RDF Vocabulary Description Language 1.0: RDF Schema.* W3C Recommendation, World Wide Web Consortium (W3C).

Dey, A., Abowd, G., & Salber, D. (2001). A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction (HCI) Journal , 16* (2), 97-166.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition , 5* (2), 199-220.

Hitzler, P., Kr\"{o}tzsch, M., Rudolph, S., & Sure, Y. (2008). *Semantic Web Grundlagen* (1 Ausg.). Springer Verlag Berlin Heidelberg.

Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., & Rudolph., S. (2009). *OWL 2 Web Ontology Language Primer.* Tech. rep., World Wide Web Consortium (W3C).

Indulska, J., Robinson, R., Rakotonirainy, A., & Henricksen, K. (2003). Experiences in Using CC/PP in Context-Aware Systems. In M.-S. Chen, P. Chrysanthis, M. Sloman, & A. Zaslavsky (Hrsg.), *Mobile Data Management* (Bd. 2574, S. 247-261). Springer Berlin Heidelberg.

ISO 19125-1:2004 . (2004). ISO 19125-1:2004 Geographic information -- Simple feature access -- Part 1: Common architecture. ISO/TC 211 Geographic information/Geomatics.

Kühn, R. R., Keller, C., & Schlegel, T. (2011). A Context Taxonomy Supporting Public System Design. *Proceedings of the 1st International Workshop on Model-based Interactive Ubiquitous Systems.* Pisa, Italy.

Manola, F., & Miller, E. (2004). *RDF Primer.* Tech. rep., World Wide Web Consortium (W3C).

Ni, H., Zhou, X., Zhang, D., & Heng, N. (2006). Context-Dependent Task Computing in Pervasive Environment. In H. Youn, M. Kim, & H. Morikawa (Hrsg.), *Ubiquitous Computing Systems* (Bd. 4239, S. 119-128). Springer Berlin Heidelberg.

Ogbuji, C. (2012). *SPARQL 1.1 Graph Store HTTP Protocol.* Working Draft, World Wide Web Consortium (W3C).

Prekop, P., & Burnett, M. (2003). Activities, context and ubiquitous computing. *Computer Communications , 26* (11), 1168-1176.

Prud'hommeaux, E., & Seaborne, A. (2008). *SPARQL Query Language for RDF.* Tech. rep., World Wide Web Consortium (W3C).

Strang, T., & Popien, C. L. (2004). A Context Modeling Survey. UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management, (S. 31-41). Nottingham.

Ziegler, J. a. (2005). Kontextmodellierung für adaptive webbasierte Systeme. In C. Stary (Hrsg.), *Mensch und Computer 2005: Kunst und Wissenschaft - Grenzüberschreitung der interaktiven Art* (S. 181-189). München: Oldenbourg Verlag.

(1)

Bildverzeichnis

Tabellenverzeichnis

Impressum

Verband Deutscher Verkehrsunternehmen e. V. (VDV)

Kamekestraße 37-39 · 50672 Köln

T 0221 57979-0 · F 0221 57979-8000

info@vdv.de · www.vdv.de

Ansprechpartner

Berthold Radermacher

T 0221 57979-141

F 0221 57979-8141

radermacher@vdv.de

Abschnittswechsel – diese Seite nie löschen!

Verband Deutscher Verkehrsunternehmen e. V. (VDV)
Kamekestraße 37-39 · 50672 Köln
T 0221 57979-0 · F 0221 57979-8000
info@vdv.de · www.vdv.de