Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Deutschland

Diplomarbeit Nr. 3588

Analyse, Konzept und Realisierung eines multimodalen B2B-Verkehrsplanungsdienstes auf Basis von Webserviceaggregation

Stanislaus Biesinger

Studiengang:	Softwaretechnik
Prüfer:	Jun.-Prof. Dr. Dimka Karastoyanova
Betreuer:	M.Sc. Wirt.-Inf. Andreas Weiß
Betreuer Industrie:	Dipl.-Ing. Volker W. Fricke
begonnen am:	01.11.2013
beendet am:	02.05.2014
CR-Klassifikation:	H.3.5, H.5.3

Kurzfassung

Die vorliegende Arbeit befasst sich mit der Konzeption und Entwicklung eines multimodalen B2B-Verkehrsplanungsdienstes auf der Basis von Webserviceaggregation. Es wird ein kurzer Überblick über das Thema „multimodale Verkehrsplanung“ und die bisher entstandenen Arbeiten auf Basis von gewichteten Graphen geboten. Die Nachteile bestehender Lösungen werden diskutiert und ein neuer Ansatz vorgestellt. Im Folgenden wird eine Beispielimplementierung dieses Ansatzes mithilfe von WS-BPEL beschrieben. Daraufhin werden die Ergebnisse, welche von dieser Beispielimplementierung geliefert werden, mit den Ergebnissen bestehender Planungsdienste verglichen. Schlussendlich werden Vor- und Nachteile des neuen webservicebasierten Ansatzes erörtert und denkbare Erweiterungsmöglichkeiten besprochen.

Abstract

This thesis deals with the conception and development of a multimodal B2B journey planner, based on web service aggregation. For this purpose, a short summary of the topic of multimodal journey planning is given. Prior work in this field based on shortest path algorithms is also discussed. It discusses the various shortcomings of existing solutions and present a novel approach based on the aggregation of existing web services. This is followed by the description of an implementation of this approach using WS-BPEL. The output of this implementation is then compared to the output of existing solutions. Finally, the various advantages and shortcomings of the concept are discussed, followed by possible solutions for these problems and possible expansions of the concept in general.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation und Zielsetzung

In der „Einführung in die Verkehrswirtschaft“ [Kum06] wird multimodaler Verkehr als „mehrgliedrige Transportkette [...], bei der die Beförderung von Personen oder der Transport eines Gutes mit zwei oder mehr unterschiedlichen Verkehrsträgern vollzogen wird“ definiert. Die multimodale Verkehrsplanung beschäftigt sich mit der Kreation dieser Transportketten. Es gibt bereits zahlreiche Arbeiten zu diesem Thema (siehe 3.2), auch gibt es bereits marktreife Implementierungen von computergestützten multimodalen Verkehrsplanungsdiensten. Jedoch besitzen diese Implementierungen durchweg zwei Limitierungen: Zum einen bedienen sie zumeist nur die Transportmittel eines einzelnen Mobilitätsanbieters, beispielsweise die Komponenten eines Verkehrsverbundes. Bei den wenigen Sonderfällen, in denen mehrere Mobilitätsanbieter berücksichtigt werden (moovel.com [Dai14]), handelt es sich mehr um eine Auflistung von alternativen Transportmitteln als eine wahre multimodale Transportplanung. Zum anderen arbeiten diese Systeme auf einem einzigen vereinheitlichten Datensatz. Dies hat zur Folge, dass neue Transportmittel/Mobilitätsanbieter, welche am multimodalen Verkehrsplanungssystem teilnehmen wollen, ihre Daten komplett offen legen und in das vom bereits bestehenden System verwendete Format konvertieren müssen. Zwar gibt es bereits Versuche, gerade für linienbasierte Mobilitätsanbieter einen einheitlichen Standard zu etablieren [Goo12], jedoch bleibt bei vielen Unternehmen der Widerwille [Lor12], ihre Daten Dritten zugänglich zu machen. Zusätzlich dazu haben sich in der Praxis bereits kommerzielle Verkehrsplanungssysteme etabliert [HAF14], deren Hersteller vermutlich wenig Interesse an einer Änderung des Status Quo besitzen. Dies ist vermutlich auch einer der Gründe, warum wahre multimodale Verkehrsplanung bisher nur innerhalb von Verbünden existiert.

Ziel dieser Diplomarbeit ist es, einen multimodalen Verkehrsplanungsdienst zu entwickeln, welcher die beiden oben genannten Limitationen umgeht. Es soll eine Integrationsplattform entstehen, in welcher bereits bestehende Planungsdienste von Mobilitätsanbietern in Form von Webservices aggregiert werden. Folgende Formen der Mobilität sollen in die multimodale Planung aufgenommen werden:

- Linienbasierter Verkehr (Bus, Bahn etc.)
- Carsharing
- Taxi
- Fußweg

Als Namen für den zu implementierenden Dienst wurde MoBee (Multimodal Mobility) gewählt.

Dieser Dienst soll im Rahmen der Green eMotion B2B Integrationsplattform als B2B Webservice (siehe 2.3) bereitgestellt werden. Das Green eMotion Projekt ist ein Teil der European Green Cars Initiative [Sie14]. Es besteht aus einem Konsortium aus 43 Partnern, sowohl aus

der Industrie als auch aus dem Universitäts- und Forschungsbereich. Ziel von Green eMotion ist die Entwicklung europaweiter Standards und Infrastruktur für Elektromobilität im Zuge der EU-Klimaziele einer CO2-Reduzierung von 60 Prozent bis zum Jahre 2050. Eine zentrale Komponente dieser Infrastruktur ist der Green eMotion E-Mobility B2B-Marketplace. Hersteller und Anbieter von Dienstleistungen rund um Elektromobilität können dort in einer auf offenen Standards basierenden Cloud Computing-Umgebung ihre Angebote integrieren und miteinander verknüpfen.

1.2 Struktur des Dokumentes

Diese Diplomarbeit gliedert sich in 7 Kapitel. In *Kapitel 2 - Grundlagen* werden Grundlagen besprochen, welche für das Verständnis der Arbeit vonnöten sind. *Kapitel 3 - Verwandte Arbeiten* behandelt verwandte Arbeiten zum Thema multimodale Verkehrsplanung und Pathing. Diese werden kurz beschrieben. Aus den ihnen inhärenten Nachteilen werden Anforderungen an das MoBee Konzept abgeleitet und dessen darauf basierende Designprinzipien erläutert. *Kapitel 4 - Implementierung* beschreibt die Implementierung des MoBee Konzeptes im Rahmen dieser Arbeit. Es behandelt die verwendete Middleware, den dem Konzept zugrunde liegenden Algorithmus, die Systemarchitektur und die Implementierung der zuvor beschriebenen Komponenten. In *Kapitel 5 - Webservice Ports* werden diejenigen Webservice Ports, welche zur Kommunikation zwischen den einzelnen Subsystemen und zur Kommunikation mit externen Webservices benötigt werden, beschrieben. *Kapitel 6 - Evaluierung* bietet einen Überblick über die nach Fertigstellung der Implementierung erkannten Stärken und Schwächen des MoBee Konzeptes. Zusätzlich werden für die weitere Arbeit am Thema Lösungsansätze für die erkannten Probleme des Konzeptes vorgeschlagen. In *Kapitel 7 - Ausblick* werden schließlich mögliche Erweiterungen für das MoBee Konzept vorgestellt.

2 Grundlagen

Im folgenden Kapitel werden einige grundlegenden Konzepte und Technologien aus dem Umfeld von Routenplanung und Webservices vorgestellt, welche innerhalb dieser Arbeit verwendet werden. Der Abschnitt *multimodale Verkehrsplanung* beschreibt die Softwaredomäne der Arbeit. *Webserviceaggregation* definiert die Technologie Webservices als ganzes und beschreibt das Konzept der Webserviceaggregation. Der Abschnitt *WS-BPEL* stellt die in der Arbeit verwendete Ausführungssprache vor. *WSDL* beschreibt die Web Service Description Language welche verwendet wird, um die internen und externen Schnittstelle des MoBee Systems zu definieren. Der Abschnitt *WGS84* beschreibt schließlich das in dieser Arbeit verwendete räumliche Bezugssystem.

2.1 Multimodale Verkehrsplanung

Die Vorteile öffentlich verfügbarer Transportmöglichkeiten sind hinreichend bekannt: Nicht nur sind sie umweltfreundlicher als private Automobile, sie stellen auch einen Zugewinn an Mobilität für diejenigen Personen dar, welche sich kein eigenes Automobil leisten können oder führen dürfen und tragen dadurch zu deren persönlicher Freiheit bei. Trotz dieser bekannten Vorteile ist der Anteil öffentlich verfügbarer Transportmöglichkeiten an der Gesamtmobilität der Bevölkerung vergleichsweise niedrig [New]. Obwohl die Gründe für die relative Inakzeptanz öffentlich verfügbarer Transportmöglichkeiten sicherlich vielfältig sind, stechen doch zwei Gründe besonders hervor [Kry04]: Zum einen wäre hier das Problem der Fragmentierung zu erwähnen. Öffentlich verfügbare Transportmöglichkeiten beschränken sich meist auf einen spezifischen Raum (Regionales Verkehrsnetz, Rückgabestationen von Carsharingdiensten). Zusätzlich dazu ist es oft umständlich und zeitraubend, einen Startpunkt für die Transportmöglichkeit (Haltestelle) zu erreichen. Zum anderen besteht, gerade bei liniengebundenen Transportmitteln, das Problem der Ineffizienz, welches zum Teil aus der zuvor erwähnten Fragmentierung entsteht: Nur ein relativ geringer Teil des Zeitaufwandes, welcher für die Benutzung liniengebundener Transportmittel benötigt wird, wird tatsächlich darauf aufgewendet, sich dem Ziel physisch zu nähern. Einen weitaus größeren Teil des Zeitaufwandes stellen das Warten auf das Eintreffen des gewählten Verkehrsmittels und das Anfahren von Verkehrsknoten (zum Beispiel Umstieg am Hauptbahnhof) dar.

Die multimodale Verkehrsplanung, also die Streckenplanung unter Zuhilfenahme mehrerer verschiedener öffentlicher Transportmöglichkeiten, versucht diese Probleme zu lindern und somit die Akzeptanz der öffentlichen Transportmöglichkeiten zu erhöhen. Durch die Kombination mehrerer Mobilitätsanbieter können Fahrtstrecken über die Verfügbarkeitsgrenzen einzelner Mobilitätsanbieter hinaus effizient geplant werden. Dies behebt das Problem der Fragmentierung fast vollständig. Auch das Problem der Ineffizienz wird abgemildert, da von einzelnen Mobilitätsanbietern nur unzureichend abgedeckte Gebiete durch flexiblere, in diesem Gebiet gut vertretene Mobilitätsanbieter überbrückt werden können. Hierbei muss beachtet werden, dass multimodale Verkehrsplanung sich nicht nur auf das Aufzeigen von Alternativen für das zurücklegen einer bestimmten Fahrtstrecke beschränkt. Plattformen wie Moovel [Dai14], welche verschiedene Transportmöglichkeiten berücksichtigen, diese jedoch nur eingeschränkt kombinieren (Fußweg bis zur Haltestelle), sind zwar sicherlich nützlich, adressie-

ren jedoch nicht die im vorigen Abschnitt erläuterten Einschränkungen öffentlich verfügbarer Transportmittel. Echte multimodale Verkehrsplanung zeigt kombinierte Fahrtstrecken unter Zuhilfenahme aller zur Verfügung stehenden Transportmöglichkeiten auf.

2.2 Webserviceaggregation

Das World Wide Web Consortium (W3C) definiert den Begriff „Webservice“ folgendermaßen:

„A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.“[W3C04]

Ein Webservice stellt also Softwarefunktionalität als Dienstleistung dar. Auf diese Funktionalität kann von externen Softwaresystemen über standardisierte Kommunikationsprotokolle (SOAP, JSON-RPC etc.) unabhängig von der Plattform und Implementierung des Webservices zugegriffen werden. Der Nutzen einzelner Webservices in Isolation, also ohne Weiterverarbeitung der Ein- oder Ausgabedaten durch weitere Software, ist jedoch stark begrenzt: Um für eine Vielzahl von Anwendungen nützlich und damit praktikabel zu sein, muss die vom Webservice angebotene Funktionalität möglichst generisch und nicht auf einen spezifischen Anwendungsbereich beschränkt sein. Eine einzelne Softwareoperation auf ein externes System auszulagern ist jedoch für moderne Softwaresysteme nicht besonders nützlich, da diese sich nur selten mit elementaren Operationen befassen. Aus diesem Grund entstand das Konzept der Webserviceaggregation [Kha03]. Durch ein System von lose gekoppelten Webservices, welche jeweils einen Teilbereich der Softwarefunktionalität übernehmen, lässt sich ein großer Teil bereits bestehender Software wiederverwenden und somit der Aufwand für neue Software erheblich reduzieren.

Bei der Aggregation von Webservices wird zwischen zwei Herangehensweisen unterschieden: Webservice Choreography und Webservice Orchestration. Webservice Choreography bezeichnet ein dezentrales Interaktionsprotokoll zwischen mehreren Webservices. Alle beteiligten Webservices werden dabei gleich behandelt, es besteht, ähnlich einem Peer-to-Peer System, keinerlei Hierarchie innerhalb des Systems. Der Fokus der Webservice Choreography liegt auf dem Nachrichtenaustausch. Jeder beteiligte Webservice weiß, zu welchem Zeitpunkt welche Operation ausgeführt werden soll und zu welcher Zeit mit welchem Webservice interagiert werden muss. Webservice Orchestration bezeichnet hingegen die Organisation des Softwareprozessflusses aus der Sicht eines einzelnen Teilnehmers, des Controller Service. Dieser enthält die Prozesslogik und koordiniert somit die ihm unterstellten Webservices. Abbildung 2-1 enthält eine grafische Gegenüberstellung dieser beiden Aggregationskonzepte

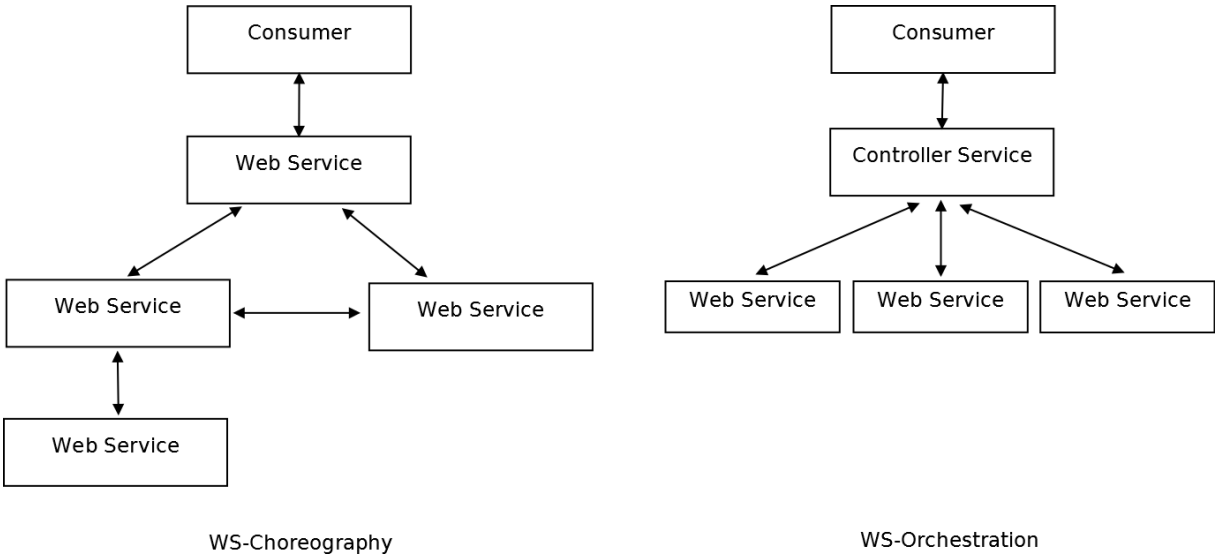


Abbildung 2-1 WS-Choreography vs. WS-Orchestration

Die Webservice Orchestration ist bereits wohlverstanden [Kar11] und es gibt bereits einige hierzu einsetzbare Technologien (WS-BPEL, WebSphere MQSeries), welche produktiv genutzt werden. Webservice Choreography ist jedoch noch Gegenstand aktiver Forschung [Suj14]. MoBee steuert die an das System angeschlossenen Webservices über einen zentralen in WS-BPEL definierten Prozess, ist also ein Vertreter der Webservice Orchestration.

2.3 B2B Webservice

Die Bezeichnung Business-to-Business (B2B) beschreibt eine Anbieter-Kunde Beziehung zwischen zwei Unternehmen. Der Begriff „Unternehmen“, vor allem auf der Kundenseite, wird in der Literatur unterschiedlich definiert: Einige Quellen schließen in den Begriff allein agierende Händler mit ein [KuR06], andere erweitern die Definition auch auf Regierungsbehörden [Kot10]. Ein B2B Webservice bezeichnet somit einen Webservice, welcher nicht an Endkunden (B2C – Business-to-Consumer), sondern an Geschäftskunden gerichtet ist. Diese können den Service in ihr eigenes Angebot integrieren, oder die Funktionalität für ihre internen Prozesse nutzen. Da MoBee als ein solcher B2B-Service konzipiert ist, wird auf Funktionalität verzichtet, welche der direkten Interaktion des Endkunden mit dem System dient. Dies wären zum Beispiel eine grafische Eingabemaske für die Streckendaten oder ein Geocoding-modul für die direkte Eingabe von Start- und Zieladressen.

2.4 WS-BPEL

Die Web Service Business Process Execution Language (WS-BPEL) ist eine ausführbare Sprache für die Beschreibung von Geschäftsprozessen des OASIS-Konsortiums [OAS07]. Die Aktivitäten, welche in WS-BPEL ausgeführt werden, sind als Webservices implementiert, die Schnittstellen nach außen hin sind ebenfalls als WSDL-Ports definiert. WS-BPEL kann folglich als Sprache für Web Service Orchestration betrachtet werden. Der Kontrollfluss von WS-BPEL wird durch vordefinierte Aktivitäten, wie zum Beispiel Weichen, Iterationen oder Schleifen ausgedrückt. Für die interne Datenhaltung werden XML-Elemente verwendet, der

Datenzugriff geschieht über XPATH. BPEL-Prozesse lassen sich in zwei Typen unterteilen: Prozesse mit langer Laufzeit und Microflow-Prozesse.

Prozesse mit langer Laufzeit beinhalten Aufrufe von anderen Prozessen/Webservices, welche nicht sofort eine Antwort auf den Aufruf liefern. Meist handelt es sich hierbei um Geschäftsprozesse, an denen menschliche Entscheidungen oder Leistungen beteiligt sind. Microflow-Prozesse haben eine vergleichsweise kurze Laufzeit und kommunizieren nur mit anderen Computersystemen. Da MoBee vollautomatisch ist und außer dem Übermitteln der Eingabeparameter keinerlei menschlicher Interaktion bedarf, werden im Rahmen dieser Arbeit nur Microflow-Prozesse betrachtet.

2.5 WSDL

Die Web Service Description Language (WSDL) ist eine protokollunabhängige Sprache für die Beschreibung von Web Services [W3C01]. Sie ist ein Standard des W3C und basiert auf der Markupsprache XML. Ein WSDL-Dokument beschreibt den vollen Funktionsumfang eines Webservices. Die Definition geschieht über 6 XML-Elemente:

- types – Beschreibt die Datentypen für den Nachrichtenaustausch
- message – Beschreibt die konkreten Nachrichten, die für die Kommunikation mit dem Webservice verwendet werden
- portType – Beschreibt die Operationen (Funktionen), welche der Webservice anbietet
- binding – Definiert Protokoll und Datenformat für die unter portType beschriebenen Operationen
- port – Definiert einen Endpunkt (URI) für ein Binding
- service – Zusammenfassung aller Ports eines PortType

Im Rahmen dieser Arbeit wird WSDL genutzt, um die Schnittstellen der einzelnen Subprozesse (siehe 4.4) und die notwendigen Schnittstellen kooperierender Mobilitätsanbieter (siehe 5.3) zu definieren.

2.6 WGS84

Das World Geodetic System 1984 (WGS84) ist ein räumliches Bezugssystem für Positionsangaben auf der Erde [Nat84]. Das Koordinatensystem, welches WGS84 zugrunde liegt, ist ein kartesisches Koordinatensystem, dessen Zentrum am Schwerpunkt der Erde liegt. Auf diesem ist ein Elipsoid definiert, welches eine Näherung an die Form der Erdkugel darstellt. Die Längen- und Breitengrade dieses Elipsoides werden verwendet, um Positionsangaben auf der Erdoberfläche zu beschreiben.

MoBee benutzt WGS84 für die Positionsangabe von Start- und Zielpunkten für Routenvorschläge und für die räumlichen Berechnungen auf den Verfügbarkeitsbereichen der Mobilitätsanbieter. Positionsangaben von kooperierenden Mobilitätsanbietern, welche in einem anderen räumlichen Bezugssystem definiert sind, werden vor der weiteren Verarbeitung umgerechnet.

3 Verwandte Arbeiten

Im folgenden Kapitel werden bereits bestehende Arbeiten zum Thema Verkehrsplanung und Routing im Allgemeinen besprochen und deren Funktionsweise und -umfang von MoBee abgegrenzt.

3.1 Pathingalgorithmen

Pathingalgorithmen im Sinne dieser Arbeit sind solche, welche das Problem des kürzesten Pfades auf einem gewichteten Graphen lösen. Ein gewichteter Graph ist hierbei als G = (V, E) definiert, wobei V die Menge der Knoten, E die Menge der Kanten bezeichnet. Die Kanten besitzen eine Gewichtung, welche die Kosten darstellt, die Kante zu passieren. Im folgenden Abschnitt werden zwei der bekanntesten Pathingalgorithmen kurz vorgestellt.

3.1.1 Dijkstra Algorithmus

Der nach seinem Erfinder Edsger W. Dijkstra benannte Dijkstra Algorithmus [Dij59] gehört zur Klasse der Greedy-Algorithmen [Gil88], das heißt es wird bei der Routenberechnung immer derjenige Knoten als nächster Wegpunkt gewählt, welcher zum Zeitpunkt das beste Ergebnis verspricht, also die kürzeste Route vom Startknoten aus darstellt. Die Kostenfunktion f(x) ist also die Summe aller Kanten welche passiert werden müssen, um den Knoten X vom Startknoten aus zu erreichen. Einmal besuchte Knoten werden nicht wieder als Wegpunkt in Betracht gezogen. Dieses Vorgehen wird wiederholt, bis entweder der Zielknoten erreicht wird, oder die optimale Route zu jedem Knoten vom Startpunkt aus bekannt ist. Abbildung 3-1 veranschaulicht dieses Vorgehen: Vom Startknoten A aus soll der kürzeste Pfad zum Endknoten H ermittelt werden. Von Knoten A aus sind die Knoten B, C und D erreichbar. Nach Prüfung der Kantengewichtung wird Knoten C als nächster Wegpunkt gewählt, da dieser mit den geringsten Kosten zu erreichen ist.

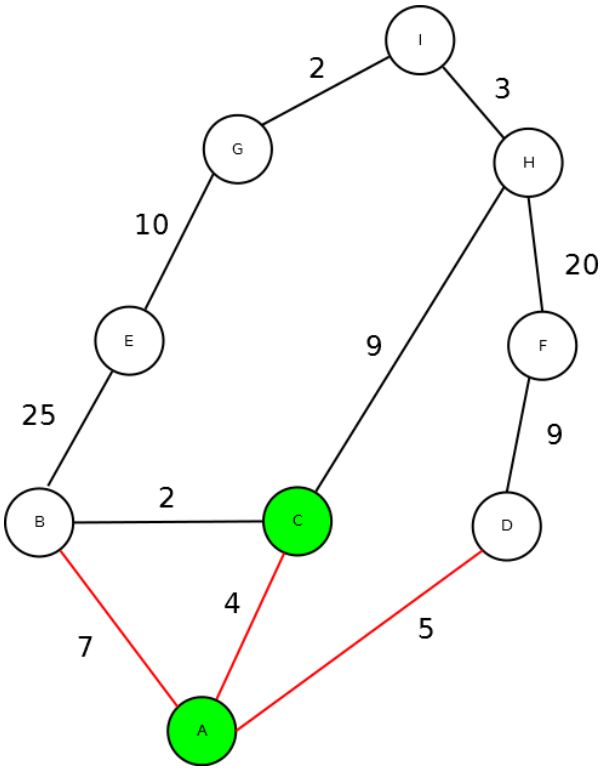


Abbildung 3-1 Wahl des nächsten Knotens beim Dijkstra Algorithmus

Knoten C wird nun, genauso wie zuvor Knoten A, als besucht markiert. Der Knoten A am nächsten gelegene, nicht besuchte Knoten ist nun Knoten D. Knoten H wäre zwar nun theoretisch auch zu erreichen, allerdings betragen die Kosten der dorthin bekannten Route 4 + 9 = 13. Die nun bekannte kürzeste Distanz zu Knoten B beträgt hingegen 2 + 4 = 6. Es wird folglich, wie zuvor mit Knoten C, mit Knoten D verfahren: Knoten D wird als besucht markiert, wodurch die Kosten um den nun erreichbaren Punkt F zu erreichen zu 5 + 9 = 14 werden. Dieses Verfahren wird nun wiederholt, bis der Endknoten H erreicht wird. Da in jedem Schritt immer nach der kürzesten Route zu einem beliebigen erreichbaren Knoten gesucht wird, kann man nach Erreichen des Endknotens davon ausgehen, dass keine effizientere Route zum Endknoten existiert. Abbildung 3-2 zeigt die Markierungen des Graphen nach Auffinden der kürzesten Route zu Knoten H.

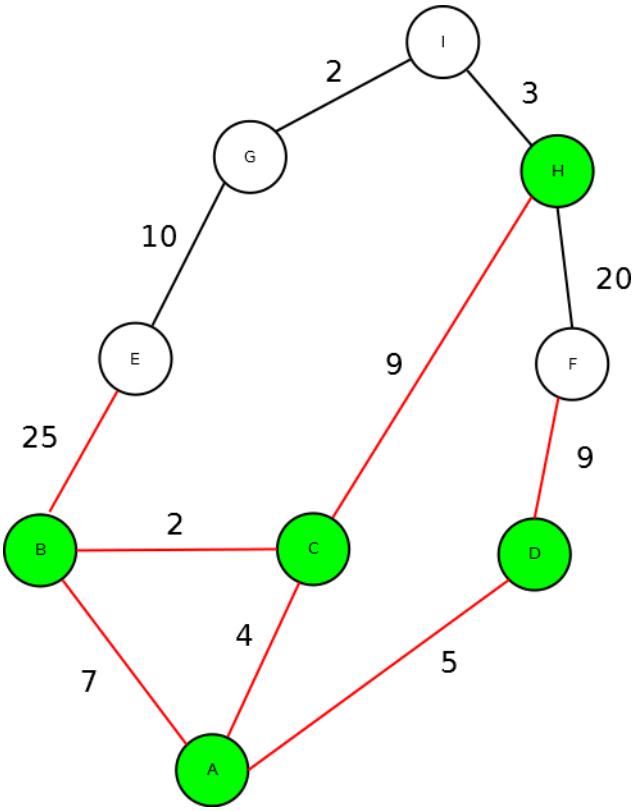


Abbildung 3-2 Dijkstra-Graph nach Auffinden der kürzesten Route zu H

Man kann leicht erkennen, dass es kostspieliger wäre, einen der anderen erreichbaren Knoten (Knoten E oder Knoten F) zu erreichen, als die gefundene Route zu H zu nutzen. Es ist somit unmöglich, dass eine effizientere Route von A zu H existiert.

Obwohl der Dijkstra Algorithmus ohne Einschränkungen auf zusammenhängenden gerichteten Graphen mit positiver Kantengewichtung korrekte Ergebnisse liefert, ist seine Performanz nicht immer optimal. Dies hängt damit zusammen, dass der Dijkstra Algorithmus, so wie alle Greedy-Algorithmen, unabhängig von der bekannten Struktur des Graphen immer nur den zurzeit am effizientesten erscheinenden Weg wählt. Dieses Problem kann durch Erweiterungen des Dijkstra-Algorithmus umgangen werden, wie zum Beispiel durch den in 3.1.2 beschriebenen A*-Algorithmus.

3.1.2 A* Algorithmus

Der erstmals 1968 beschriebene A* Algorithmus stellt eine Erweiterung des Dijkstra-Algorithmus um einen heuristischen Bestandteil [Har68] dar. Die Kostenfunktion f(x) für einen noch nicht besuchten Knoten setzt sich beim A* Algorithmus aus zwei Unterfunktionen zusammen:

- g(x) ist die aus dem Dijkstra Algorithmus bekannte Kostenfunktion. Die Kosten aller Kanten, welche passiert werden müssen, um den Knoten X zu erreichen.

- h(x) ist eine heuristische Funktion zur Abschätzung der Distanz des Knoten X vom Zielknoten.

Der Vorteil dieser Herangehensweise wird in Abbildung 3-3 deutlich. Als Schätzfunktion h(x) wird in diesem Beispiel die Luftlinie zwischen dem zu betrachtenden Knoten und dem Zielknoten F gewählt. An dieser Stelle muss betont werden, dass h(x) die *Kosten* abschätzt, den Zielknoten zu erreichen. Der Wert der Funktion h(x) selbst stellt keine Schätzung dar, sondern ist genau bestimmbar. Die Schätzung besteht vielmehr darin, dass die genaue Relation zwischen dem Wert von h(x) und der Entfernung zum Zielpunkt auf dem Graphen unbekannt ist.

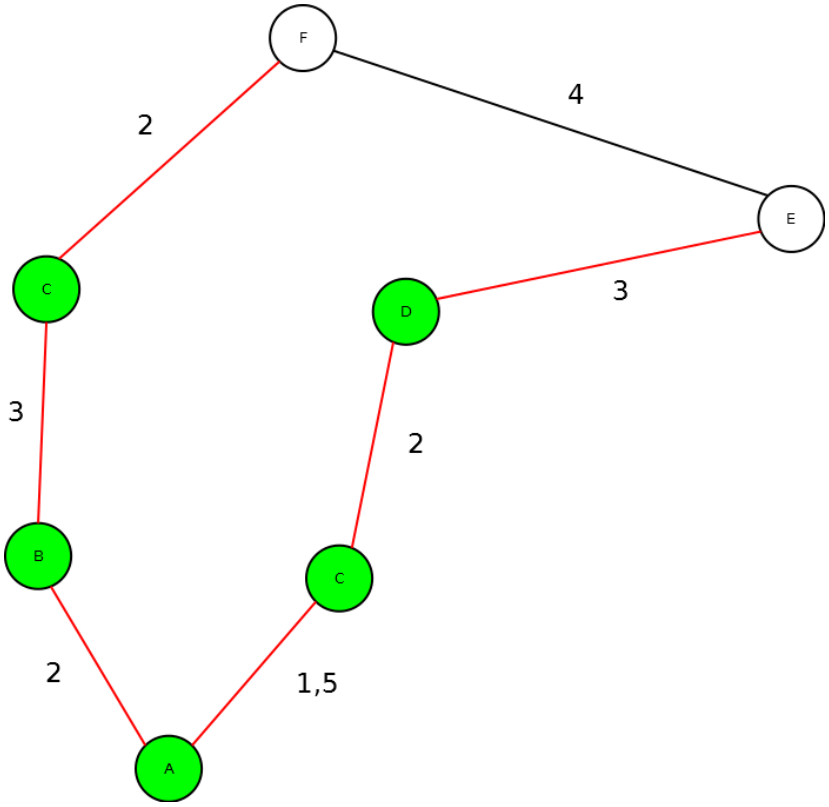


Abbildung 3-3 Ausgangspunkt unterschiedlicher Vorgehensweise von Dijkstra und A*

Bis zu diesem Punkt in der Wegfindung wäre der Ablauf des Dijkstra und A* Algorithmus identisch. Der Dijkstra Algorithmus würde an dieser Stelle Knoten E als nächstes betrachten, da die Kosten diesen zu erreichen mit 1,5 + 2 + 3 = 6,5 geringer sind als die Kosten, Knoten F zu erreichen (2 + 3 +2 = 7). Unter Zuhilfenahme der heuristischen Funktion wären beim A* Algorithmus allerdings die Kosten von Knoten E größer, da dieser sich räumlich vom Zielknoten F entfernt. Der A* Algorithmus wäre in diesem Fall also effizienter, da er sich die Betrachtung des Knotens E sparen würde. Abbildung 3-4 zeigt eine Gegenüberstellung des nächsten Schrittes der beiden Algorithmen.

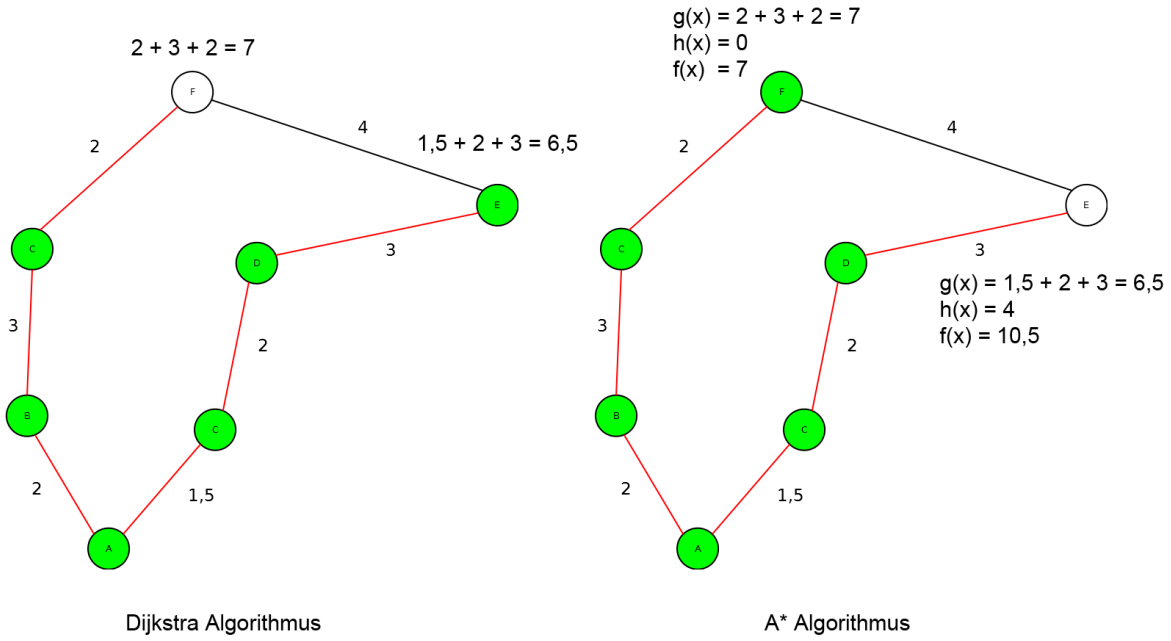


Abbildung 3-4 Unterschiedliche Vorgehensweise von Dijkstra und A*

An dieser Stelle ist nochmals zu erwähnen, dass es sich bei h(x) um eine heuristische Funktion, also einen Schätzwert, handelt. Die Tatsache, dass der Wert von g(x) in diesem Beispiel genau der Kantengewichtung zwischen den Knoten entspricht liegt daran, dass einerseits als h(x) die Luftlinie zwischen den Knoten gewählt wurde und andererseits in diesem Beispiel der letzte Schritt der Wegfindung betrachtet wird.

3.2 Theoretische Routingansätze

Im folgenden Abschnitt werden die bisher gängigen Ansätze für die Implementierung von Routingalgorithmen für Verkehrsplanung betrachtet.

3.2.1 Routing von dynamisch gerouteten Mobilitätsanbietern

Routing von dynamisch gerouteten Mobilitätsanbietern, also Routing auf Straßennetzen, wird auf einem klassischen, gewichteten Graphen durchgeführt. Hierbei wird das Straßennetz als ein Graph G = (V,E) definiert, wobei die Menge der Knoten V Kreuzungen oder Abzweigungen und die Menge der Kanten E die eigentlichen Straßen repräsentieren. Das Problem des kürzesten Pfades ist bereits lange Gegenstand von Forschungsarbeiten, die das Problem betreffenden Algorithmen (A*, Dijkstra, etc.) gehören zum Standardrepertoire der theoretischen Informatik (siehe 3.1). Aktuelle Forschungsarbeiten zu diesem Thema befassen sich hauptsächlich mit der Steigerung der Effizienz bereits bekannter Algorithmen, zum Beispiel Landmark A* [Gol05] oder SHARC [Bau09]. Bestehende Mobilitätsplanungsdienste, welche dynamisch geroutete Mobilitätsoptionen anbieten, beschränken sich entweder auf Vermittlung von Fahrzeugen (MyTaxi, car2go) und bieten nicht die Routenplanung selbst, oder haben die interne Funktionsweise ihres Dienstes nicht offengelegt (AnachB, Moovel). Es ist jedoch davon auszugehen, dass auch diese Dienste eine Variante des gewichteten Graphen verwenden, da bisher kein alternativer Ansatz veröffentlicht wurde.

3.2.2 Routing von linienbasierten Mobilitätsanbietern

Das Routing auf linienbasierten Mobilitätsanbietern erfolgt ebenfalls auf gewichteten Graphen. Sämtliche verfügbare Forschungsarbeiten zu diesem Thema basieren auf diesem Konzept. Es haben sich jedoch zwei unterschiedliche Ansätze zu der Fragestellung herauskristallisiert, woraus der Graph gebildet werden soll:

Der *Time-Expanded*-Ansatz [Fra00] nutzt einen Graphen, dessen Knotenmenge aus Paaren zwischen einem Zeitereignis (Ankunft/Abfahrt) und einem Ort (Haltestelle) besteht. Die Kanten repräsentieren in diesem Graphen entweder Bewegung in Raum und Zeit (Fahrt zu einer anderen Haltestelle) oder nur in der Zeit (Verweilen an einer Haltestelle).

Der *Time-Dependent*-Ansatz [Bro04] erzeugt einen Graphen, in welchem jeder Knoten eine Haltestelle des betreffenden Liniennetzes abbildet. Die Kanten repräsentieren jeweils eine elementare Verbindung (Fahrt ohne Zwischenstopp) zwischen den Haltestellen. Die Gewichtung der Kanten wird bei diesem Ansatz zur Laufzeit festgelegt und ist von den jeweils zum fraglichen Zeitpunkt verfügbaren Linien von diesem Knoten aus abhängig.

Darüber hinaus existieren einige Hybridansätze, wie zum Beispiel [Ant12], welche die beiden Graphentypen kombinieren.

3.2.3 Multimodales Routing

Echtes multimodales Routing wurde in der Forschung bisher nur sehr selten behandelt. Der von Horn beschriebene *Journey-Planner* in [Hor02] implementiert einen *Time-Dependent*-Graphen linienbasierter Mobilitätsanbieter, auf dem eine Breadth-First Suche durchgeführt wird. Der ebenfalls in [Hor02] beschriebene *Fleet-Scheduler* wird über einen Message Broker mit diesem verbunden, sodass Taxifahrten als erster, beziehungsweise letzter Streckenabschnitt einer vorgeschlagenen Reiseroute genutzt werden können. Yu und Lu beschreiben in [YuH10] einen evolutionären Algorithmus, welcher auf einem *Time-Expanded*-Graphen arbeitet. Der von ihnen vorgeschlagene Algorithmus ist auf linienbasierte Mobilitätsanbieter sowie auf Taxifahrten anwendbar.

Der bisher einzig vollständig multimodale Ansatz ist der von Hrncir und Jakob in [Hrn13] beschriebene *Generalised Time-Dependent Graph.* Dieser Graph besteht aus mehreren Untergraphen: Einen für das Straßennetz und einen für jeden integrierten linienbasierten Mobilitätsanbieter. Verknüpfungen zwischen den Graphen werden durch Mappings zwischen Knoten des Straßengraphen (Adressen) und Knoten der Liniengraphen (Haltestellen) hergestellt. Dieser Ansatz bietet fast vollständige multimodale Verkehrsplanung. Die einzige Ausnahme bilden hier Carsharing-Services und unter Umständen Mietfahrräder, da keine Möglichkeit vorgesehen ist, zur Laufzeit den Standort verfügbarer Fahrzeuge in den Graphen zu integrieren.

3.3 Bereits veröffentlichte Dienste zur multimodalen Verkehrsplanung

Im folgenden Abschnitt werden zwei bereits einsatzbereite multimodale Verkehrsplanungsdienste vorgestellt. Diese sind das Moovel System der Daimler AG und der österreichische Routenplaner AnachB.

3.3.1 Moovel

Moovel ist das multimodale Reiseplanungssystem der Daimler AG [Dai14]. Es bietet linienbasierte Mobilitätsanbieter, Carsharing über car2go und Taxifahrten als Transportmöglichkeiten an. Der Dienst ist an Endkunden gerichtet, der Zugriff erfolgt entweder über die Website des Systems oder über eine Smartphone-App. Obwohl sich der Dienst selbst als multimodal bezeichnet, beschränkt er sich auf die Aufzählung mehrerer alternativer Verbindungsmöglichkeiten zwischen Start- und Zielpunkt. Er erfüllt somit nicht die in 2.1 aufgestellte Definition eines multimodalen Verkehrsplanungsdienstes. Da es sich bei Moovel um ein kommerzielles System handelt wurde der Quellcode, beziehungsweise der zu Grunde liegende Algorithmus, nicht offengelegt. Aufgrund der Tatsache, dass die verschiedenen Reiseoptionen unterschiedlich viel Zeit benötigen, um ein Ergebnis zu liefern, ist es jedoch wahrscheinlich, dass es sich bei Moovel um ein Beispiel von Webserviceaggregation handelt. Zumindest lässt es sich dadurch vermuten, dass externe Daten für die Routenvorschläge genutzt werden.

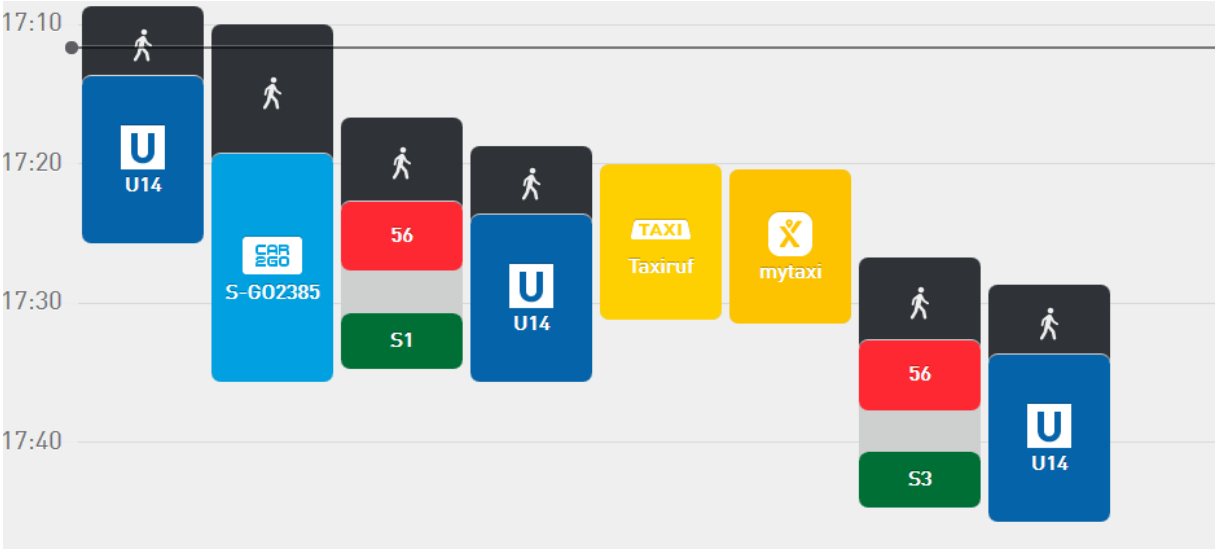


Abbildung 3-5 Ausgabemaske von Moovel

3.3.2 AnachB

AnachB ist ein multimodaler Verkehrsplanungsdienst für die Region Niederösterreich. Er bietet multimodale Verkehrsplanung mit öffentlichen Verkehrsmitteln, dem Fahrrad und privaten PKW über eine Park-and-Ride Funktion. Der Dienst richtet sich an Endkunden und ist über seine Website bedienbar. Hervorzuheben ist, dass AnachB stets mit aktuellen Informationen von Verkehrssensoren, Baustellen-, Störungs-, Unfall- und Fahrplandatenbanken versorgt wird, welche in die Routenberechnung mit aufgenommen werden. Der Quellcode von AnachB wurde nicht offengelegt, somit lassen sich keine Aussagen über die zu Grunde liegende Funktionsweise des Systems treffen.

Abbildung 3-6 Park-and-Ride Ausgabemaske von AnachB

3.4 Gegenüberstellung zu MoBee

In diesem Abschnitt werden die Limitierungen der in den vorangegangenen Abschnitten vorgestellten Ansätze und Systeme zur Routenplanung erörtert. Anschließend werden aus diesen die Eckpunkte für den Ansatz des MoBee Systems abgeleitet.

3.4.1 Nachteile bisheriger Ansätze und Lösungen

Alle hier beschriebenen theoretischen Ansätze zur Routenplanung, ob multimodal oder nicht, besitzen einen gemeinsamen Nachteil: Sie alle arbeiten mit einer bekannten Datenmenge. Dies führt zu dem Umstand, dass diejenigen linienbasierten Mobilitätsanbieter, welche in die Routenplanung mit aufgenommen werden sollen, Informationen über ihr Liniennetz offenlegen und aktuell halten müssten. Wie die Erfahrung gezeigt hat, sind gerade die größeren Anbieter (zum Beispiel die Deutsche Bahn, welche auch im Rahmen dieser Arbeit kontaktiert wurde und nicht zur Kooperation bereit war) nicht gewillt, dies zu tun. Auch muss bedacht werden, dass die Daten aller Mobilitätsanbieter normalisiert und zusammengeführt werden müssten. Dies mag einen einmaligen Aufwand darstellen, er ist jedoch in Anbetracht der vielen offenen und proprietären Formate nicht unerheblich.

Darüber hinaus ist es schwierig, dynamisch geroutete Mobilitätsanbieter in diese Ansätze mit aufzunehmen: Der Ansatz in [Hor02] kann nur deshalb Taxifahrten mit einbeziehen, weil in diesem Modell die Flottenverwaltung des Taxiunternehmens vom systemeigenen *Fleet-Scheduler* übernommen wird. Er bietet keine Schnittstelle für externen Dateninput. Auch der bisher einer echten multimodalen Verkehrsplanung am nächsten kommende theoretische Ansatz von Hrncir und Jakob [Hrn13], könnte nur dann Taxis und Carsharinganbieter sinnvoll in

die Routenplanung mit aufnehmen, wenn der von ihnen vorgeschlagene *Generalised Time-Dependent Graph* (vgl. 3.2.3) laufend mit aktuellen Daten versorgt werden würde.

Die bisher verfügbaren Implementierungen multimodaler Verkehrsplanungssysteme besitzen auch einige nicht zu übersehende Nachteile: Moovel bietet keine echte Multimodalität, sondern nur eine Auflistung von Alternativen, AnachB ist nur für einen relativ kleinen Teilbereich Österreichs verfügbar. Beiden gemein ist die Eigenschaft, dass sie sich nur an Endkunden richten. Eine weitere Benutzung durch Geschäftskunden ist mit den zur Verfügung stehenden grafischen Schnittstellen nicht möglich.

Auffällig bei beiden vorhandenen Diensten ist auch die geringe Anzahl von kooperierenden Unternehmen. Dies mag damit zusammenhängen, dass der Typische Anwendungsfall (Use-Case) für einen multimodalen Verkehrsplanungsdienst einen relativ kleinen Raum umfasst. Es gibt jedoch auch Use-Cases, in denen Regionen übergreifend geroutet werden muss (siehe 4.2.3). Hierfür sind die bisherigen Systeme jedoch denkbar ungeeignet.

3.4.2 Ansatz von MoBee

Im vorherigen Abschnitt wurden folgende Limitationen bisheriger Ansätze identifiziert:

- Notwendigkeit eines vollständigen, normalisierten Datensatzes mit Daten aller kooperierender Mobilitätsanbieter
- Meist keine Möglichkeit, aktuelle Daten in die Routenberechnung mit aufzunehmen
- Keine echte Multimodalität bei aggregierenden Planungsdiensten
- Geringe Reichweite aufgrund von geringer Anzahl beteiligter Mobilitätsanbieter.

MoBee umgeht diese Probleme durch das Routing mithilfe bereits bestehender Dienste der kooperierenden Mobilitätsanbieter:

Da auf die bereits vorhandenen Routingsysteme der einzelnen Mobilitätsanbieter zurückgegriffen wird, ist *kein vereinheitlichter Datensatz notwendig*. Das Einzige, was normalisiert werden muss, sind die externen Schnittstellen der bereits vorhandenen Routingsysteme (siehe 5.3). Dies ist jedoch nicht zwingend notwendig, da mit verhältnismäßig geringem Aufwand Adapter für die bestehenden Schnittstellen implementiert werden können (siehe 4.6).

Die *Aktualität von Informationen* ist durch die Verwendung von Daten direkt vom Mobilitätsanbieter ebenfalls gewährleistet, da davon ausgegangen werden kann, dass die Mobilitätsanbieter ihre eigenen Dienste mit den aktuellsten, ihnen zur Verfügung stehenden Daten versorgen.

Auch das *Miteinbeziehen von Carsharinganbietern und Taxidiensten* ist kein Problem, da die Verfügbarkeitsdaten ihrer Fahrzeugflotten ebenfalls zur Laufzeit in die Streckenberechnung mit aufgenommen werden können.

MoBee bietet *echtes multimodales Routing*. Die Routingdaten der verschiedenen Mobilitätsanbieter werden zu Routenvorschlägen kombiniert, nicht nur als Alternativen aufgezeigt. Darüber hinaus können beliebig viele Mobilitätsanbieter mit geringem Aufwand an das System angeschlossen werden. MoBee ist somit *nicht auf eine bestimmte Region* beschränkt. Durch die rekursive Natur des in 4.2 beschriebenen TiGeR-Algorithmus ist es theoretisch möglich, transnationale Mobilitätsanbieter mit einzubeziehen und somit weltweites Routing anzubieten.

4 Implementierung

Im folgenden Kapitel wird die Beispielimplementierung des MoBee Konzeptes im Rahmen dieser Arbeit vorgestellt. Behandelt werden die verwendete Middleware, der dem MoBee Konzept zugrunde liegende Algorithmus TiGeR mitsamt einem Ablaufbeispiel und die Softwarearchitektur der Implementierung.

4.1 Verwendete Middleware

Im folgenden Abschnitt wird die in der Implementierung von MoBee verwendete Middleware aufgelistet und kurz beschrieben.

4.1.1 IBM WebSphere Application Server/Process Server

Der WebSphere Application Server (WAS) ist, wie der Name schon sagt, ein Applikationsserver für Webanwendungen und das Flaggschiff der IBM WebSphere Suite [IBM14]. Der WAS basiert auf offenen Standards, wie zum Beispiel JavaEE, XML und Webservices. Die Hauptfunktion des WAS besteht darin, eine Laufzeitumgebung für JavaEE-Anwendungen anzubieten. Diese werden in Form einer Enterprise Archive Datei (EAR) auf dem Server installiert (deployed). Eine EAR enthält typischerweise Java-Klassen als Bytecode, Java Server Pages für die Nutzeroberfläche und Konfigurationsdaten in XML-Form, sogenannte Deployment-Deskriptoren. Im Rahmen des MoBee Systems wird der Websphere Application Server als Laufzeitumgebung für die Web Service Wrapper (siehe 4.6) genutzt.

WebSphere Process Server wird von IBM als eigenständiges Produkt vermarktet, ist jedoch im Kern eine Spezialkonfiguration des WebSphere Application Server. Durch eine Anzahl an vorinstallierten JavaEE-Anwendungen wird es dem Process Server möglich, Anwendungen mit serviceorientierter Architektur (SOA) als Laufzeitumgebung zu dienen. Insbesondere die durch die Umrüstung gewonnene Fähigkeit, WS-BPEL Prozesse auszuführen, prädestiniert den WebSphere Process Server als Middleware für das MoBee Projekt.

4.1.2 IBM DB2 Server/DB2 Spatial Extender

DB2 ist das relationale Datenbanksmanagementsystem (RDBMS) der Firma IBM [IBM141]. Als Abfragesprache wird die Structured Query Language (SQL) genutzt. Der von DB2 verwendete SQL-Dialekt entspricht weitgehend ANSI-SQL und es stehen für viele Programmiersprachen DB2-Datenbanktreiber zur Verfügung. Es ist daher möglich, Datenbankabfragen aus Programmen heraus mit Embedded SQL zu realisieren.

Beim DB2 Spatial Extender handelt es sich um eine Erweiterung für DB2, welche die Arbeit mit räumlichen Datentypen ermöglicht [IBM142]. Räumliche Datentypen sind hierbei geometrische Formen, wie zum Beispiel Punkte, Linien oder Polygone, welche in Bezug zu einem räumlichen Bezugssystem stehen. Es stehen out-of-the-box eine Vielzahl vordefinierter räumlicher Bezugssysteme (unter anderem das vom Global Positioning System verwendete WGS84) bereit, es ist jedoch auch möglich, eigene Bezugssysteme zu definieren. Das räumliche Bezugssystem ist für jedes räumliche Datenobjekt frei wählbar. Bei geometrischen Opera-

tionen (Finden des Schnittpunktes, Prüfung auf Identität, etc.) sind Objekte mit unterschiedlichen räumlichen Bezugssystemen kompatibel, die Objekte werden automatisch in das räumliche Bezugssystem des Ergebnisobjektes konvertiert. Da in MoBee die Datenbank der Speicherung der Availability Areas (siehe 4.2.1.2) von Mobilitätsanbietern dient, ist eine Erweiterung der Datenbankfunktionalität um räumliche Datentypen unabdingbar. Die im Spatial Extender bereits vorhandenen Datenbankfunktionen für geometrische Operationen erlauben es darüber hinaus, sämtliche räumlichen Berechnungen auf den Datenbankserver auszulagern (siehe 4.5).

4.2 Routingalgorithmus (TiGeR)

Aus den in 3.4.1 aufgezeigten Problematiken mit konventionellen Routingalgorithmen geht hervor, dass ein neuer Routingalgorithmus für das MoBee-System notwendig ist.

Ein Algorithmus, welcher zusammengesetzte Routen aus mehreren diskreten Routingdiensten erstellen soll, muss folgende Anforderungen erfüllen:

- Auswahl der für die geplante Route in Frage kommenden Mobilitätsanbieter
- Auswahl geeigneter Teilstrecken der Gesamtroute
- Verknüpfung der gewählten Teilstrecken

Im folgenden Kapitel wird ein Routingalgorithmus beschrieben, welcher multimodale Routenplanung durch Aggregation bereits bestehender Webservices ermöglicht: Tiered Geospatial Routing (TiGeR). Es wird darüber hinaus gezeigt, in welcher Weise die oben genannten Anforderungen von TiGeR erfüllt werden.

4.2.1 Terminologie

Im folgenden Abschnitt werden einige TiGeR-spezifische Begriffe kurz erläutert.

4.2.1.1 Mobilitätsanbieter

Mobilitätsanbieter im Sinne von TiGeR sind alle externen Dienstleister, welche Routenvorschläge für die von ihnen angebotenen Transportmittel über eine Webservice-Schnittstelle anbieten. Es wird ferner unterschieden zwischen linienbasierten (Bus, Bahn) und dynamisch gerouteten Mobilitätsanbietern (Carsharing, Taxi etc).

4.2.1.2 Availability Area

TiGeR arbeitet mit dem Konzept der Availability Areas (A2). Dies sind diejenigen räumlichen Bereiche, in welchem ein Mobilitätsanbieter seine Dienste anbietet. Die A2 werden in TiGeR lokal vorgehalten, brechen also mit dem Konzept der reinen Aggregation bereits extern vorhandener Daten. Zum Zeitpunkt der Erstellung dieser Arbeit stellt kein Mobilitätsanbieter diese Daten als Service bereit. Es kann auch als unwahrscheinlich betrachtet werden, dass dies jemals der Fall sein wird (Ausrichtung der Services auf Endkunden). Aus diesem Grunde ist diese Instanz von lokaler Datenhaltung unvermeidbar.

4.2.1.3 Routenvektor

Als Routenvektor wird in TiGeR eine gerade Linie zwischen zwei geografischen Punkten (Ziel- und Endpunkt einer zu berechnenden Route oder Ziel- und Endpunkt einer zu berechnenden Teilstrecke einer Route) bezeichnet. Die Richtung des Vektors ist vom Start- zum Zielpunkt.

4.2.1.4 Anbietertypen (Tiers)

Mobilitätsanbieter werden in zwei verschiedene Typen unterteilt. Typ-0 Anbieter sind dynamisch geroutete Mobilitätsanbieter, wie zum Beispiel Taxi- oder Carsharingunternehmen. Typ-1 Anbieter sind linienbasierte Mobilitätsanbieter, welche eine lokal eng begrenzte A2 besitzen, zum Beispiel Stadtbahnen oder Buslinien. Typ-2 Anbieter sind solche, welche Regionen übergreifende A2 besitzen, zum Beispiel die Deutsche Bahn oder die Österreichische Bundesbahn.

4.2.2 Ablauf

Abbildung 4-1 zeigt den Ablauf von TiGeR als Flußdiagramm.

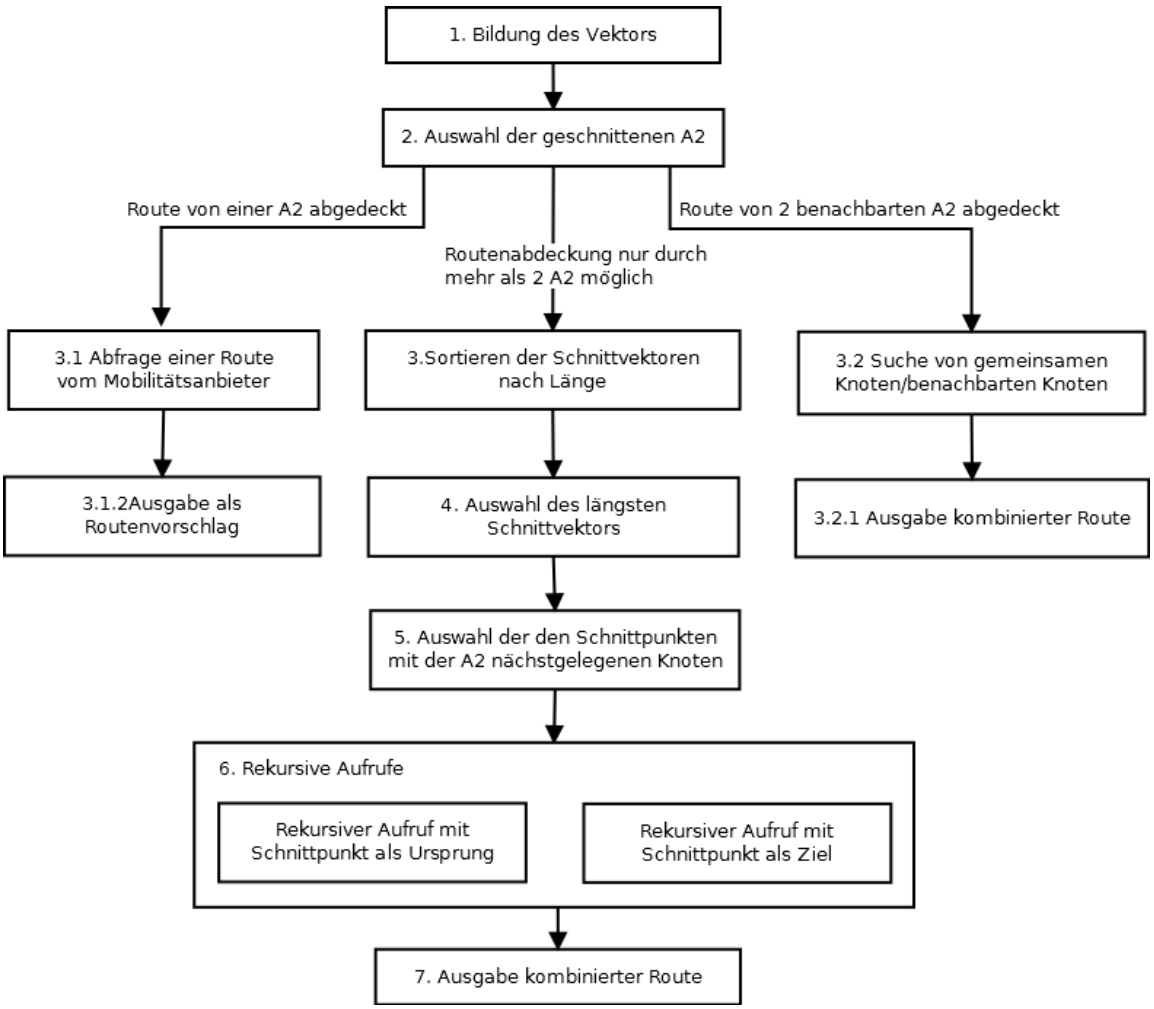


Abbildung 4-1 Ablauf von TiGeR als Flussdiagramm

Die Schritte im Diagramm werden im Folgenden einzeln erläutert.

1. **Bildung des Vektors**
   In diesem Schritt wird aus den Eingabekoordinaten ein Routenvektor erzeugt. Er dient der Auswahl der in Frage kommenden A2.

2. **Auswahl der geschnittenen A2**
   Aus der lokalen Datenbank werden diejenigen A2 ausgewählt, welche vom Routenvektor geschnitten werden. Die Datenbank gibt hierbei die den A2 zugehörigen Mobilitätsanbieter, die Schnittpunkte der A2 mit dem Routenvektor sowie die Länge des Schnittvektors zurück.

3. **Routenberechnung je nach Abdeckungsfall**
   An diesem Punkt wird zwischen drei unterschiedlichen Fällen unterschieden: Abdeckung des Vektors durch einen Typ-0/1 Mobilitätsanbieter, Abdeckung durch zwei benachbarte Tier-1 Mobilitätsanbieter und Abdeckung durch 3 oder mehr Teilstrecken. Dabei ist zu beachten, dass diese Fälle sich nicht gegenseitig ausschließen.

   Im Falle einer Abdeckung durch einen einzelnen Typ-0/1 Anbieter (Schritt 3.1) wird eine Route von dem zur A2 zugehörigen Mobilitätsanbieter abgerufen. Diese wird dann als Routenvorschlag ausgegeben.

   Im Falle einer Abdeckung des Routenvektors durch zwei benachbarte A2 von Typ-0/1 Anbietern wird ein gemeinsamer Knoten der beiden Mobilitätsanbieter, beziehungsweise werden zwei nahe beieinanderliegende Knoten gesucht. Die beiden Teilrouten werden anschließend von den jeweiligen Mobilitätsanbietern abgerufen und zusammen als Routenvorschlag ausgegeben.

   Im Falle einer Abdeckung unter Zuhilfenahme eines Typ-2 Mobilitätsanbieters oder mindestens 3 Teilstrecken werden die geschnittenen A2 nach der Länge des Schnittvektors sortiert und der Algorithmus fährt mit Schritt 4 fort.

4. **Auswahl des längsten Schnittvektors**
   Der Mobilitätsanbieter mit dem größten Schnittvektor auf der A2 wird ausgewählt.

5. **Auswahl der den Schnittpunkten mit der A2 nächstgelegenen Knoten**
   Die den Schnittpunkten mit dem Routenvektor nächstgelegenen Knoten des Mobilitätsanbieters werden gesucht und eine Route zwischen diesen beiden Punkten abgefragt.

6. **Rekursive Aufrufe**
   Der Algorithmus ruft sich selbst auf, mit den Schnittpunkten mit der A2 des Mobilitätsanbieters aus 5. als Ursprungs- beziehungsweise Zielpunkt.

7. **Ausgabe kombinierter Route**
   Die kombinierte Route, zusammengesetzt aus den Teilrouten aus 5. und 6. wird als Routenvorschlag ausgegeben.

4.2.3 Ablaufbeispiel

Im folgenden Abschnitt wird, nachdem im vorherigen Abschnitt der Ablauf des TiGeR-Algorithmus in der Theorie erklärt wurde, eben jener Algorithmus anhand eines realen Beispiels nochmals veranschaulicht. Dieses Beispiel befasst sich mit folgendem Szenario: Ein Geschäftsmann möchte vom Hilton Innsbruck zurück nach Hause in die Krefelder Straße nach Stuttgart. Um seine Heimreise zu planen, nutzt er MoBee, von dessen Pilotphase er von einem Freund bei IBM gehört hat. Er übergibt Start- und Zieladresse, sowie seine gewünschte Ankunftszeit, an die Green eMotion B2B Plattform. Diese greift auf einen ebenfalls auf der Plattform verfügbaren Geocoding-Service zu und übermittelt die so erhaltenen Start- und Zielkoordinaten zusammen mit der Ankunftszeit an das MoBee System. Das MoBee System verfährt nun wie folgt:

Zuerst wird der Routenvektor gebildet. Dieser verläuft in diesem Fall von (47.261957, 11.395971) nach (48.810369, 9.212706). Der Routenvektor wird nun an die Datenbank übermittelt und von ihm geschnittene Availability Areas geprüft. Die den geschnittenen Availability Areas zugehörigen Mobilitätsanbieter werden nun der Prozesslogik übergeben. Abbildung 4-2 zeigt den erstellten Routenvektor zusammen mit den geschnittenen A2 der Verkehrsverbünde Tirol und Stuttgart, sowie dem Verfügbarkeitsbereich des Stuttgarter car2go Dienstes.

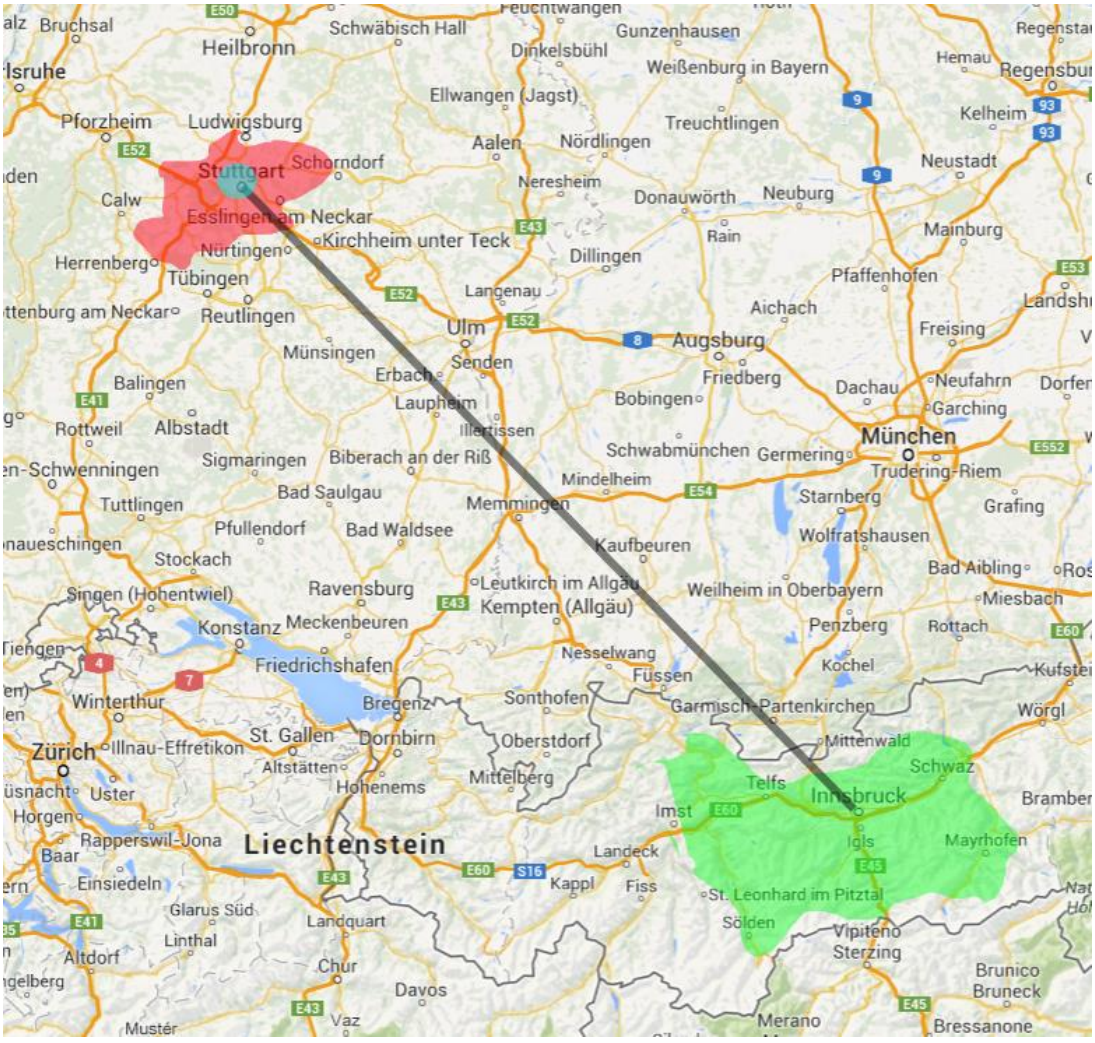


Abbildung 4-2 Routenvektor Innsbruck - Stuttgart mit geschnittenen A2

Die ebenfalls geschnittenen A2 der Österreichischen Bundesbahn und der Deutschen Bahn sind in dieser Grafik nicht markiert, da sie beide die komplette Fläche der Grafik abdecken würden. Des Weiteren ist zu erwähnen, dass die hier gezeigten Availability Areas nicht detailgetreu die echten A2 der Mobilitätsanbieter abbilden. Sie stellen nur eine Näherung dar und dienen dem erleichterten Verständnis dieses Beispiels.

MoBee prüft nun, ob der gesamte Routenvektor von einer Availability Area eines Typ-0 oder Typ-1 Anbieters beinhaltet wird. Aus Abbildung 4-2 ist ersichtlich, dass dies nicht der Fall ist. Auch die Prüfung auf Abdeckung durch zwei sich überlappende Typ-0 oder Typ-1 Mobilitätsanbieter verläuft negativ. Da beide Möglichkeiten für eine nichtrekursive Routenerzeugung nun ausgeschlossen sind, wird mit der Bildung einer Tiered Route begonnen.

Zuerst werden die einzelnen Schnittvektoren der von der Datenbankabfrage zurückgegebenen Availablity Areas der Länge nach sortiert. Hierbei wird ersichtlich, dass die A2 der Österreichischen Bundesbahn und der Deutschen Bahn die längsten Schnittvektoren besitzen. MoBee stellt nun eine Anfrage nach den Endpunkten des Routenvektors naheliegenden Knoten an den Webservice der Österreichischen Bundesbahn. Der Webservice gibt die Haltestellen Innsbruck Hauptbahnhof und Bad Cannstatt Bahnhof zurück. Diese beiden Haltestellen werden nun als Ziel- beziehungsweise Startpunkt für zwei neue Instanzen des MoBee-Prozesses genutzt.

Die Teilstrecke Hilton - Innsbruck Hauptbahnhof wird zuerst auf Abdeckung durch einen einzelnen Typ-0 oder Typ-1 Anbieter überprüft. Hier wird von der Datenbank der Mobilitätsanbieter VVT gefunden. Da die Strecke relativ kurz (< 1km) ist, werden zusätzlich Fußweg und Taxifahrt gewählt. Die Teilstrecke Bad Cannstatt Bahnhof – Krefelder Straße wird von den A2 von VVS und car2go abgedeckt. Auch hier werden aufgrund des kurzen Routenvektors Taxifahrten und Fußweg mitaufgenommen.

Nun wird mit der Routenbildung begonnen. Für die Strecke Bad Cannstatt Bahnhof - Krefelder Straße wird eine direkte Fahrt mit dem VVS verworfen, da der VVS-Webservice keine geeignete Strecke mit seinen Fahrzeugen zur gewählten Uhrzeit liefert. Es wird allerdings eine Alternativstrecke über den Stuttgarter Hauptbahnhof gefunden. Es wird ebenso am Cannstatter Bahnhof ein car2go Fahrzeug gefunden. Eine Fahrtstrecke mit Diesem und der Fußweg werden jeweils so berechnet, dass der Fahrgast zur gewünschten Zeit in der Krefelder Straße ankommt. Danach wird nach jeweils einer Fahrt mit der ÖBB gesucht, welche zum ermittelten Abfahrtszeitpunkt am Bahnhof Bad Cannstatt ankommt. Die so ermittelten Abfahrtszeitpunkte der ÖBB-Strecken werden nun als Ankunftszeitpunkte für die Teilstrecke Hilton – Innsbruck Hauptbahnhof genutzt. Die hieraus entstandenen Vorschläge für die jeweiligen Teilrouten werden nun kombiniert und jeweils als Routenvorschlag von MoBee ausgegeben. Die Green eMotion Plattform gibt diese nun an den Geschäftskunden weiter, welcher sie an den Endkunden weiterleitet.

4.3 Architekturübersicht und Systemkontext

MoBee ist mit der Zielsetzung konzipiert, auf der Green eMotion B2B-Integrationsplattform ausgeführt zu werden. Einen Überblick über das Zusammenspiel zwischen dem MoBee System und den umliegenden Softwarekomponenten bietet Abbildung 4-3. Routenanfragen werden von dem Endkunden über den Geschäftskunden, welcher bei Green eMotion registriert ist, an die Green eMotion Plattform weitergeleitet. Diese übergibt die Anfrage anschließend an das MoBee System. MoBee ist über Webservice Schnittstellen mit den kooperierenden Mobilitätsanbietern und einem Karten/Routendienst verbunden. Von diesen werden Informationen über die benötigten (Teil-)Routen abgefragt und von MoBee zu einem oder mehreren Routenvorschlägen kombiniert. Die kombinierten Routenvorschläge werden wiederum über die Green eMotion Plattform zuerst an den Geschäftskunden, dann an den Endnutzer weitergeleitet. Alternativ kann die Funktionalität des MoBee Systems auch als Teil eines aggregierten Dienstes der Green eMotion Plattform aufgerufen werden. Hierbei würde ein Dienst angeboten werden, welcher als eine seiner Komponenten eine multimodale Reiseplanung beinhaltet. Dieser würde dann die Ausgabedaten von MoBee beinhalten, ohne dass der Kunde den Dienst explizit aufruft.

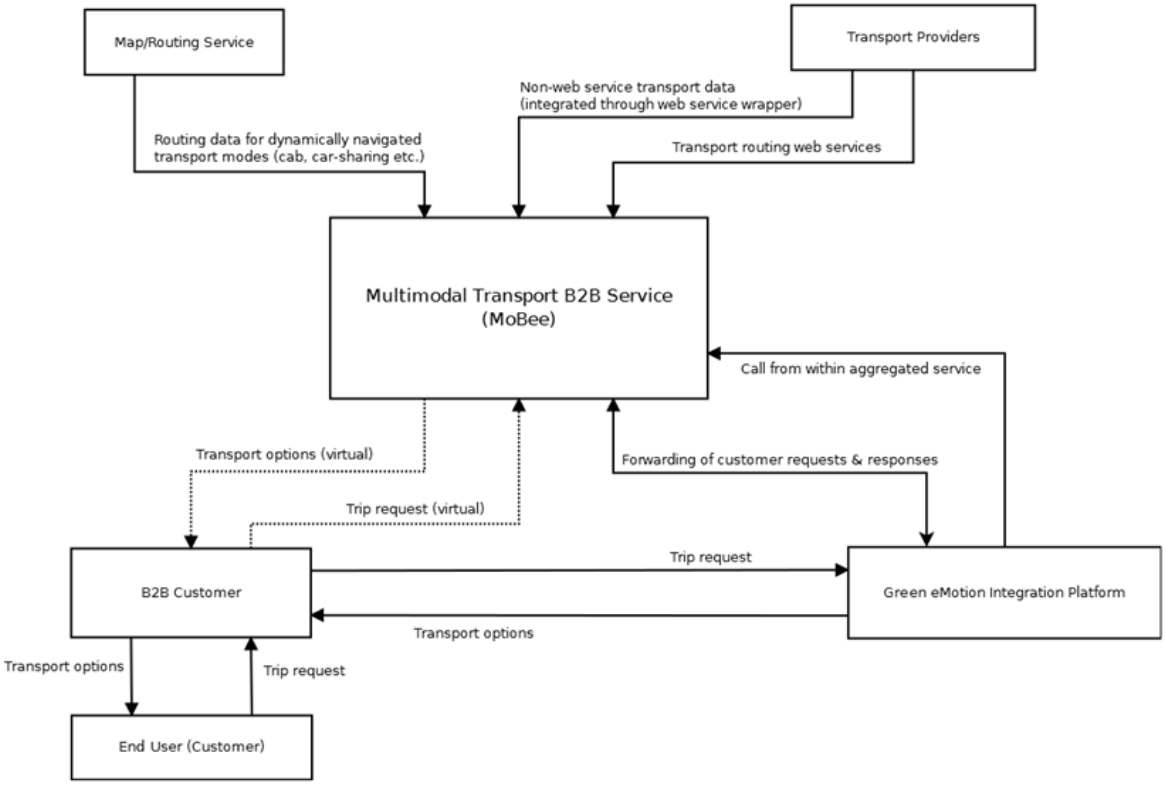


Abbildung 4-3 MoBee System Context Diagram

MoBee besteht aus drei Subsystemen, welche jeweils einen Teil der Funktionalität des Systems implementieren. Diese Subsysteme sind: WS-Wrapper, Prozesslogik und Geospatial

Subsystem. Die Subsysteme bilden eine 2-Schichten-Architektur (siehe Abbildung 4-4). Die Kommunikation zwischen den einzelnen Subsystemen geschieht über das Simple Object Access Protocol (SOAP).

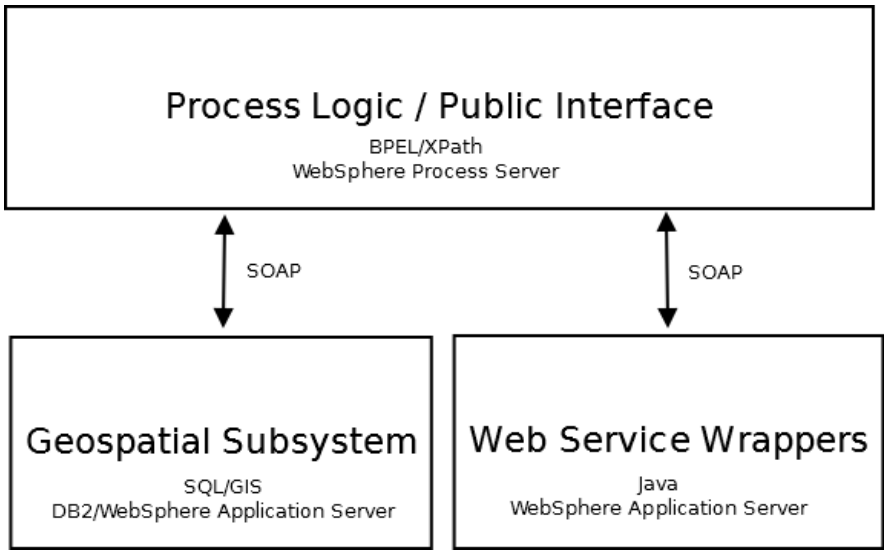


Abbildung 4-4 MoBee Systemarchitektur

Die Prozesslogik behandelt das eigentliche Routing (siehe Abschnitt 4.2) und beinhaltet die öffentliche Schnittstelle des Systems. Das Geospatial Subsystem (GSS) implementiert die räumlichen Berechnungen, welche zur Auswahl der für das Routing in Frage kommenden A2 benötigt werden. Zudem liefert es die für die jeweils in der Prozesslogik laufenden Prozesse erforderlichen räumlichen Daten (Routenvektoren, Schnittmengen von A2 etc). Die Web Service Wrapper dienen der Kommunikation mit Mobilitätsanbietern, welche keinen Webservice nach 5.3 anbieten. Zum Zeitpunkt der Diplomarbeit sind dies natürlich alle angebundenen Webservices (da die Schnittstellen aus 5.3 noch nicht veröffentlicht sind), somit sind die Wrapper derzeit ein großer Teil der Implementierung. Sie sind allerdings kein Teil des MoBee Systems als solchem. Im Folgenden wird die Implementierung der einzelnen Subkomponenten im Detail beschrieben.

4.4 Prozesslogik

Die Prozesslogik von MoBee ist, wie bereits erwähnt, als BPEL-Microflow implementiert. Der Prozess gliedert sich hierbei in eine Anzahl von Subprozessen, welche jeweils einen spezifischen Aufgabenbereich abdecken. Auf oberster Ebene befindet sich der Hauptprozess, welcher mit der Datenbank kommuniziert und die Routenvorschläge der typspezifischen Routingprozesse aggregiert. Ihm untergeordnet sind die drei typspezifischen Subprozesse: GetSimpleRoutes, HandoffTrip und TieredTrip, welche jeweils einen Typ von Route erzeugen und an den Hauptprozess übergeben. An unterster Stelle finden sich die Prozesse, welche die generischen Queries für die jeweiligen Typen von Mobilitätsanbietern anbieten: Linienbasier-

te Anbieter, Carsharinganbieter und Taxis. Diese Prozesse greifen über die in 5.3 beschriebenen Webservice Ports auf die Informationssysteme der Mobilitätsanbieter zu. Abbildung 4-5 zeigt die Struktur der Subprozesse, deren Verknüpfung miteinander und die Verknüpfungen mit den genutzten externen Webservice Schnittstellen. Rauten stellen in diesem Diagramm BPEL-Prozesse dar, Pfeile Verweise auf Webservices. Die folgenden Abschnitte betrachten die einzelnen Subprozesse im Detail.

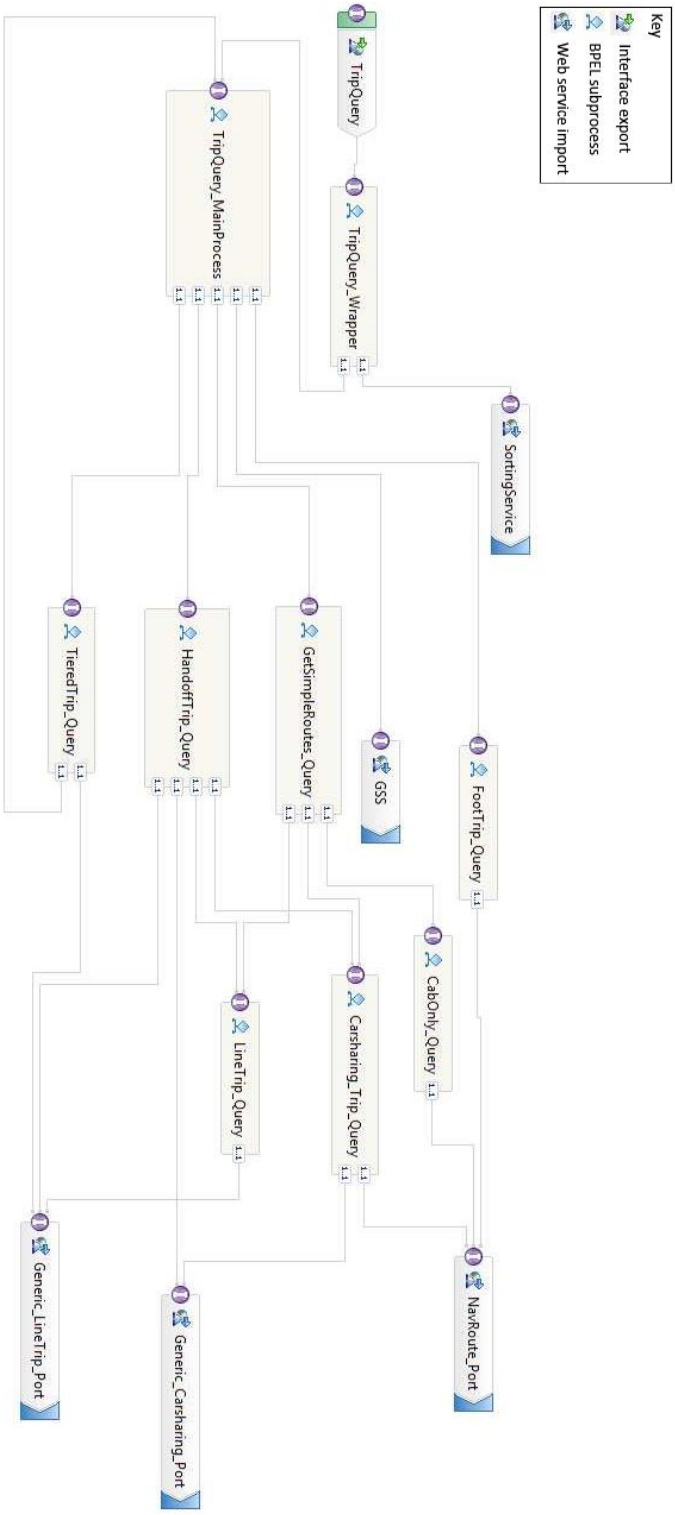


Abbildung 4-5 Assemblydiagramm MoBee

4.4.1 GetSimpleRoutes Query

Der GetSimpleRoutes Query behandelt den einfachsten Fall: Der an den Hauptprozess übergebene Routenvektor wird von einem einzelnen Mobilitätsanbieter komplett abgedeckt. Der Prozess empfängt von einer Instanz des Hauptprozesses das Ergebnis einer Datenbankabfrage nach 4.5.2.2, welche alle vom Routenvektor geschnittenen Mobilitätsanbieter beinhaltet. Der GetSimpleRoutes Prozess iteriert nun über die Menge von Mobilitätsanbietern. Für jeden Anbieter, dessen Availability Area den Routenvektor komplett abdeckt, wird der dem Anbietertyp zugehörige Subprozess für die Routengenerierung aufgerufen und dessen Resultat der Liste von Routenvorschlägen hinzugefügt. Diese Liste wird letztendlich dem aufrufenden Prozess übergeben. Abbildung 4-6 veranschaulicht den Aufruf der entsprechenden Subprozesse.

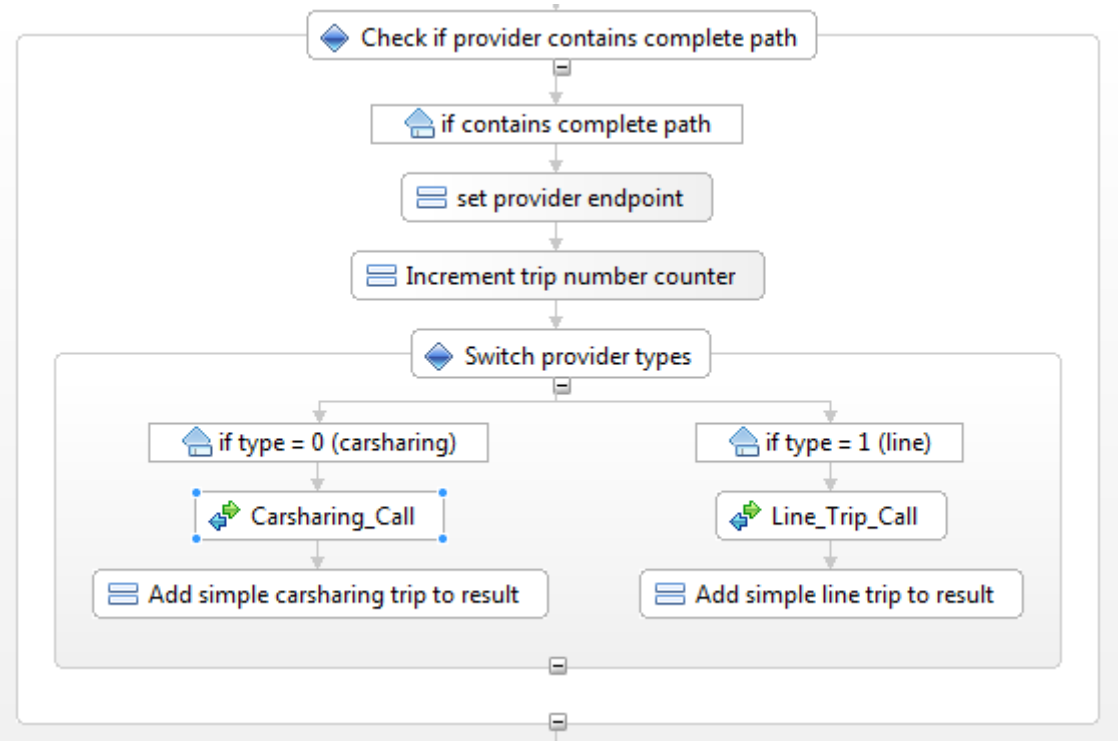


Abbildung 4-6 Aufruf der entsprechenden Subprozesse im GetSimpleRoutes Prozess

Taxirouten werden hier gesondert behandelt. Es sind in der Datenbank von Mobilitätsanbietern keine Taxiunternehmen enthalten, da sich zum Erstellungszeitpunkt der Arbeit keine am Taxiunternehmen fanden, welche im Rahmen dieser Arbeit kooperieren wollten. Da davon ausgegangen wird, dass Taxis überall verfügbar sind, wird der CabOnly Subprozess ohne Überprüfung einer Availability Area aufgerufen. Diese Funktionalität ist jedoch kein Platzhalter: Auch in einem fertigen System mit Anbindung an reale Taxiunternehmen kann es vorkommen, dass keines dieser Unternehmen die benötigte Teilstrecke bedient. In diesem Fall wird ein Schätzwert für ein generisches Taxiunternehmen geliefert. Da in dieser Version des

MoBee Konzeptes die Buchung der Teilstrecken manuell vorgenommen werden muss, ist die fehlende Kopplung an ein bestimmtes Taxiunternehmen noch unproblematisch. Sollte in späteren Versionen ein Buchungssystem integriert werden (siehe 7.1), muss dieser Fall gesondert behandelt werden.

4.4.2 TieredTrip Query

Der TieredTrip Query ist das Herzstück des beschriebenen TiGeR Algorithmus. Er empfängt vom Hauptprozess eine Liste der von dem vorgegebenen Routenvektor geschnittenen Mobilitätsanbietern, zusammen mit deren Serviceendpunkten und Schnittvektoren. Diese Liste wird zuerst nach dem längsten Schnittvektor durchsucht. Der diesem Vektor zugehörige Mobilitätsanbieter (Backbone) wird in einer lokalen BPEL-Variable gespeichert. Dies wird in Abbildung 4-7 veranschaulicht.

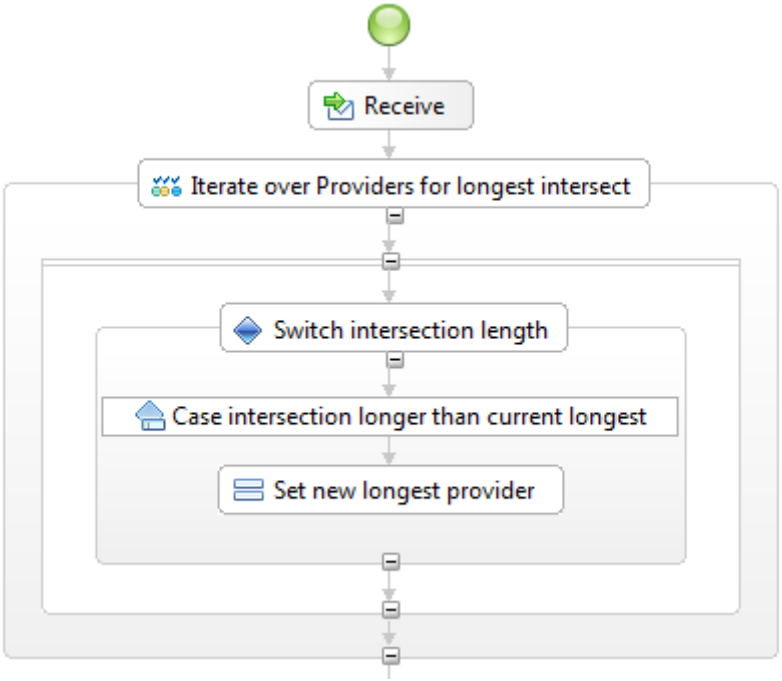


Abbildung 4-7 Ausschnitt TieredQuery Subprozess

Danach wird der Endpoint der Referenz auf den generischen Webservice Port im BPEL-Prozess zugewiesen. Diese Servicereferenz wird nun aufgerufen und dazu genutzt, die den Start- und Endpunkten des Schnittvektors am nächsten gelegenen Knoten des Backbones abzurufen. Der Hauptprozess wird nun rekursiv aufgerufen, um eine Streckenberechnung für die Teilstrecke zwischen globalem Startpunkt und dem Startpunkt des Backbones zu erzeugen. Die Ankunftszeiten der sich hieraus ergebenden Möglichkeiten werden nun als Abfahrtszeiten für Fahrten mit dem Backbone genutzt, welcher anschließend für jede mögliche Route für die erste Teilstrecke aufgerufen wird. Für die Teilstrecke hinter dem Verfügbarkeitsbereich des Backbones wird nun noch einmal rekursiv der Hauptprozess aufgerufen. Nach jedem dieser Aufrufe werden nun die Routenketten zu einem Routenvorschlag zusammengefügt. Schluss-

endlich wird die Liste der Routenvorschläge an die aufrufende Instanz des Hauptprozesses übergeben. Abbildung 4-8 veranschaulicht dieses Vorgehen.

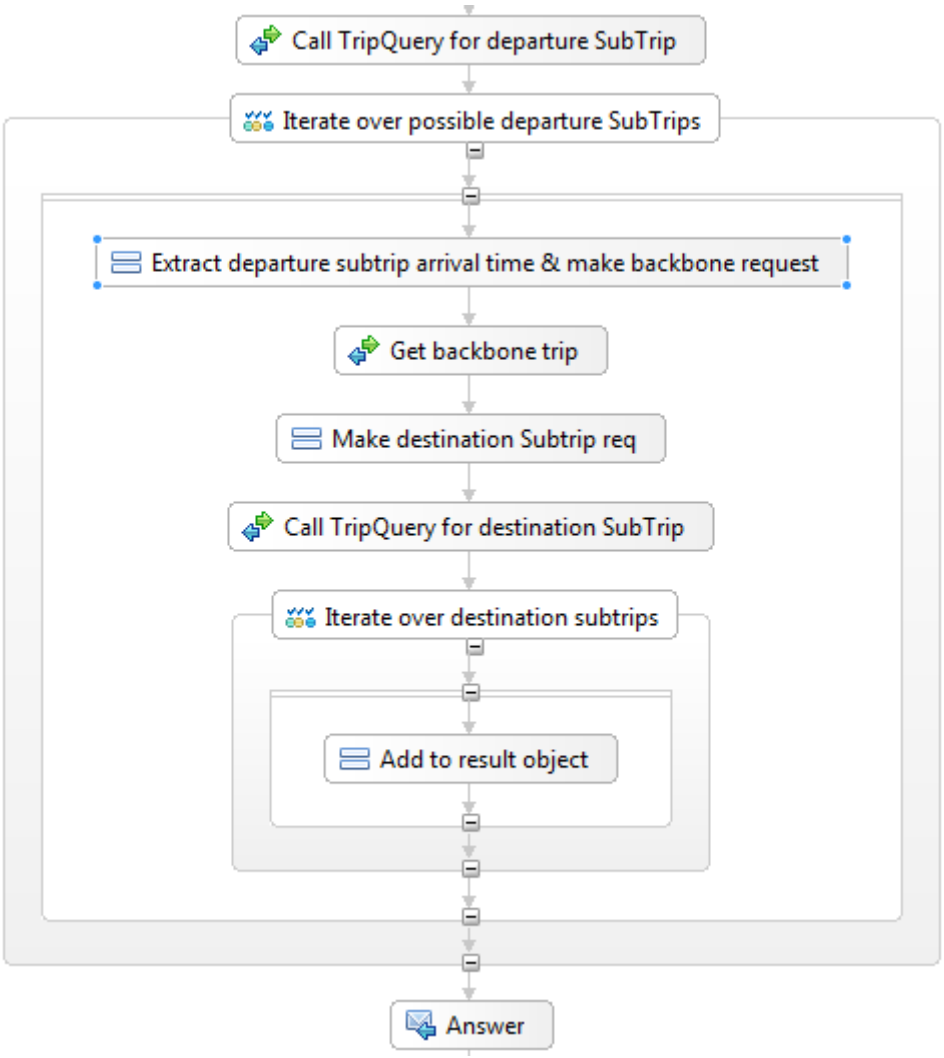


Abbildung 4-8 Bildung einer Tiered Route

Im Falle der Angabe einer gewünschten Ankunftszeit, anstatt einer gewünschten Abfahrtszeit, wird umgekehrt verfahren: Der „hintere“ Teil des ermittelten Routenvektors wird zuerst rekursiv an den TripQuery Subprozess übergeben und dessen Abfahrtszeit als gewünschte Ankunftszeit für die Teilstrecke auf dem Backbone genutzt.

4.4.3 Handoff Query

Der Handoff Query behandelt den Sonderfall, dass ein Routenvektor von zwei sich berührenden oder sich überlappenden Availability Areas abgedeckt wird. Der Query selbst kümmert sich nicht um das Auffinden dieser A2 und den ihnen zugehörigen Mobilitätsanbietern, dies wird, wie in 4.5.2.3 beschrieben, vom GSS übernommen. Der Handoff Query empfängt vom Hauptprozess eine Liste von Paaren aus Mobilitätsanbietern, welche den Routenvektor zu-

sammen abdecken. Nun wird geprüft, um welche Anbietertypen es sich dabei handelt: Im Falle von zwei Typ-1 Anbietern wird eine einfache Routenabfrage (siehe 4.4.4) über die komplette Wegstrecke an den auf dem Routenvektor „vorderen“ Anbieter gestellt. Dies hat den Hintergrund, dass alle Typ-1 Anbieter, welche zum Erstellungszeitpunkt dieser Arbeit in MoBee integriert sind, zumindest die größeren Knoten ihrer Nachbarnetze beinhalten und auch dort Routenvorschläge liefern können. Dies hat deutlich effizientere Routenvorschläge zum Ergebnis, als eine Verknüpfung zweier diskreter Routenanfragen über einen gemeinsamen Knoten, da der zur Berechnung genutzte Routenvektor nicht notwendigerweise über dem realen Liniennetz liegt

Im Falle von zwei Typ-0 Anbietern wird zuerst geprüft, ob sich in der Nähe des Übergabepunktes ein verfügbares Fahrzeug desjenigen Anbieters befindet, dessen Availability Area den „hinteren“ Teil des Routenvektors abbildet. Wurde ein Fahrzeug gefunden, wird dessen Position als Zielpunkt für eine Routenanfrage an den ersten Anbieter, und als Startpunkt für eine Anfrage an den zweiten genutzt. Die beiden Teilrouten werden nun verknüpft und ausgegeben. Abbildung 4-9 bietet eine grafische Darstellung dieses Vorgangs.

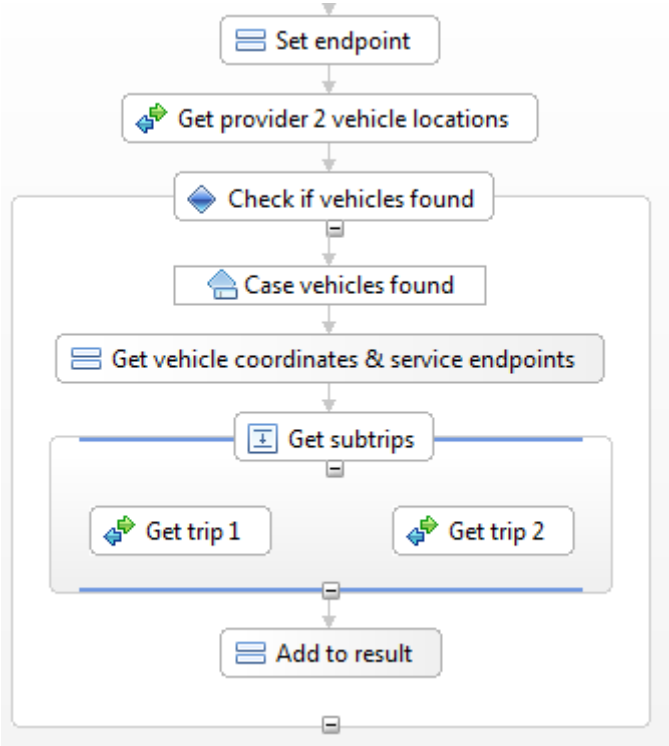


Abbildung 4-9 Handoff Query mit zwei Typ-0 Anbietern

Im Falle einer Mischung aus Typ-0 und Typ-1 Anbietern wird ähnlich vorgegangen, mit dem Unterschied, dass im Falle eines Typ-1 Anbieters als „erstem“ Anbieter auf eine Verifikation von vorhandenen Fahrzeugen verzichtet werden kann.

4.4.4 LineTrip Query

Der LineTrip Query Subprozess behandelt das Abfragen möglicher Routen von einem Typ-1 Mobilitätsanbieter. Er empfängt Start- und Zielpunkt, als auch den Serviceendpoint eines Typ-1 Anbieters. Der Serviceendpoint wird der Referenz auf den generischen Webservice Port (siehe 5.3.1) zugewiesen. Danach werden zuerst Knoten in der Nähe von Start- und Zielpunkt, dann mögliche Routen zwischen diesen Knoten abgefragt. Die Ergebnismenge wird dann dem aufrufenden Prozess übergeben.

4.4.5 CarsharingTrip Query

Dieser Subprozess liefert mögliche Reiserouten mit Typ-0 Anbietern welche Carsharing anbieten. Wie der LineTrip Subprozess empfängt auch er Start-, Zielpunkt und Serviceendpoint eines Mobilitätsanbieters. Nachdem über den generischen Typ-0 Port (siehe 5.3.2) dem Startpunkt nahegelegene verfügbare Fahrzeuge des Mobilitätsanbieters gefunden wurden, wird über den angebundenen Routingservice (siehe 5.3.3) eine Autoroute bezogen. Diese wird dann, zusammen mit dem Standort des verfügbaren Fahrzeugs, an den aufrufenden Prozess übergeben.

4.4.6 FootTrip Query

Dieser Subprozess dient der Ermittlung von Fußrouten. Er beinhaltet einen Aufruf des externen Routingdienstes. Diesem werden Start- und Zielpunkt übergeben. Das Resultat des Routenvorschlages wird dann in das MoBee-interne Format *FootTripType* konvertiert (siehe 5.1.2).

4.4.7 CabTrip Query

Der CabTrip Query Subprozess ist in dieser Implementierung von MoBee ein Platzhalter, da sich kein Online Taxiservice dazu bereit erklärte, eine Schnittstelle zu den benötigten Daten in ihrem System zur Verfügung zu stellen. Der Prozess ruft eine Autoroute vom Routingservice ab und errechnet aus deren Parametern (Dauer, Wegstrecke) Schätzwerte für Dauer und Kosten einer Taxifahrt. Diese werden an den aufrufenden Prozess übergeben. Wie bereits erwähnt, könnte diese Funktionalität nach Integration von Taxiunternehmen weiterverwendet werden, um auch Routenvorschläge mit dem Taxi für solche Gebiete anbieten zu können, welche nicht innerhalb der Availability Area eines solchen integrierten Taxiunternehmens liegen.

4.4.8 TripQuery_MainProcess

Der Hauptprozess nimmt vom Wrapper oder einem TieredTrip Query Start- und Endpunkte der gewünschten Route an und leitet diese je nach Bedarf an die entsprechenden Subprozesse weiter. Dazu wird zuerst die Datenbank nach 4.5.2.2 abgefragt, um diejenigen Mobilitätsanbieter zu erhalten, deren Availability Areas vom Routenvektor geschnitten werden oder diesen komplett enthalten. Danach wird der GetSimpleRoutes Query (siehe 4.4.1) aufgerufen, um eventuell vorhandene einfache Routen zu erhalten. Sollten keine einfachen Routen gefunden werden, werden die kombinierten Routenqueries aufgerufen. Deren Ergebnisse werden zusammengefasst und an den aufrufenden Subprozess übergeben.

4.4.9 TripQuery_Wrapper

Dieser Subprozess dient dazu, die Sortierfunktion von rekursiven Aufrufen des TripQuery_MainProcess Subprozesses abzukapseln. Er nimmt als Eingabeparameter Start- und Zielpunkt der gewünschten Route, zusammen mit Sortier- und Begrenzungsparameter (siehe 4.7) entgegen. Nachdem eine Instanz von TripQuery_MainProcess eine Liste von Routenvorschlägen zurückgibt, gibt er diese Liste zusammen mit dem Sortierparameter an den Sorting Service weiter. Die von Diesem sortierte Liste stellt die Ausgabe des TripQuery_Wrapper Prozesses dar.

4.5 Geospatial Subsystem

Der folgende Abschnitt behandelt das Geospatial Subsystem (GSS) der MoBee Implementierung. Das GSS ermöglicht es, Informationen über vorhandene Mobilitätsanbieter via Embedded SQL aus einer Datenbank abzurufen. Der Aufbau und die genaue Funktionsweise des GSS werden im Folgenden beleuchtet.

4.5.1 Übersicht

Das GSS hat die Aufgabe anhand eines Routenvektors die in Frage kommenden Mobilitätsanbieter für eine zu erstellende Reiseroute zu liefern. Zu diesem Zwecke enthält es eine Datenbank mit einer Tabelle aller angeschlossenen Mobilitätsanbieter. Abbildung 4-10 zeigt die Struktur der Tabelle.

Name	Data Type	Length
ENDPOINT	VARCHAR	1024
CARRIERNAME	VARCHAR	64
AREA	DB2GSE.ST_MULTIPOLYGON	
TYPE	INTEGER	

Abbildung 4-10 Datenbankstruktur GSS

Das Feld ENDPOINT beinhaltet die Endpunktreferenz des dem Mobilitätsanbieter zugehörigen Webservice.

CARRIERNAME beinhaltet die Textbezeichnung des Mobilitätsanbieters.

Das Feld AREA enthält die Availability Area des Mobilitätsanbieters als ST_Multipolygon. Hierbei handelt es sich um einen räumlichen Datentyp des DB2 Spatial Extenders, welcher ein in sich abgeschlossenes Polygon aus Koordinaten eines räumlichen Bezugssystems (im Falle von MoBee WGS84) beinhaltet.

Das Feld TYPE enthält die Typisierung des Moblitätsanbieters nach 4.2.1.4.

Das GSS bietet über eine Webservice Schnittstelle drei Funktionen für räumliche Berechnungen an:

- Suche nach allen Mobilitätsanbietern, deren Availability Areas eine bestimmte WGS84-Koordinate beinhalten.

- Suche nach allen Mobilitätsanbietern, deren A2 von einem Routenvektor geschnitten werden. Rückgabe der Mobilitätsanbieter und der jeweiligen Schnittvektoren.

- Suche nach Paaren von Mobilitätsanbietern, welche zusammen einen Routenvektor abdecken.

Der Webservice Port, seine Operationen und die ihnen zu Grunde liegenden Datentypen werden in 5.2 im Detail beschrieben.

4.5.2 Queries und Spatial Functions

Die Abfrage von Daten vom DB2 Server erfolgt über vorgefertigte SQL-Queries (Embedded SQL) innerhalb einer JavaEE Anwendung, welche räumliche Berechnungsfunktionen (Spatial Functions) des DB2 Spatial Extenders verwenden. Diese werden im Folgenden genauer betrachtet.

4.5.2.1 Abfrage von beinhaltenden Availability Areas

```
1  SELECT ENDPOINT, TYPE, CARRIERNAME FROM mobee.availareas
2  WHERE
3  db2gse.ST_WITHIN(db2gse.ST_POINT(X, Y,1003), AREA
4  )
5  = 1
```

In diesem Query werden Endpoint, Typ und Name aller Mobilitätsanbieter abgerufen, deren Availability Areas den Punkt (X,Y) beinhalten. Die Spatial Function ST_WITHIN übergibt 1, falls der Punkt (X,Y) in AREA liegt, 0 falls nicht. Die Funktion ST_POINT ist die Konstruktorfunktion für den Räumlichen Typ selben Namens. Die Zahl 1003 ist hierbei die Kennzahl des zu verwendenden räumlichen Bezugssystems, in diesem Fall WGS84.

4.5.2.2 Abfrage von geschnittenen Availability Areas

```
1  WITH IntersectionsTable AS (
2  SELECT ENDPOINT, TYPE, CARRIERNAME,
3
4    db2gse.ST_TOLINESTRING(
5       db2gse.ST_INTERSECTION(
6       db2gse.ST_LINESTRING('linestring(XY1 XY2',1003),
7       AREA)
8     ) AS IntersectionLine
9
10  FROM  mobee.availareas )
11
12  SELECT ENDPOINT, TYPE, CARRIERNAME,
13    db2gse.ST_LENGTH(IntersectionLine) AS IntersectionLength,
14    db2gse.ST_STARTPOINT(IntersectionLine) AS StartPoint,
15    db2gse.ST_ENDPOINT(IntersectionLine) AS INTERSECTIONENDPOINT
16
17    FROM IntersectionsTable
18
19  WHERE db2gse.ST_ISEMPTY(INTERSECTIONLINE) != 1
```

Der hier beschriebene Query ist zweiteilig. Im ersten Teil wird eine virtuelle Tabelle erzeugt, welche eine zusätzliche Spalte mit dem Schnittvektor der jeweiligen Availability Area enthält. Der zweite Teil extrahiert die Start- und Endpunkte des Schnittvektors. Die Zweiteilung erfolgt aus Performancegründen: Durch das Zwischenspeichern des Schnittvektors werden die leistungsintensiven Spatial Functions ST_INTERSECTION und ST_LINESTRING pro Mobilitätsanbieter nur einmal aufgerufen.

4.5.2.3 Abfrage von benachbarten Mobilitätsanbietern

```
1   WITH ContainingAreas AS(
2   SELECT ENDPOINT, TYPE, CARRIERNAME, AREA
3   FROM mobee.availareas WHERE
4
5   db2gse.ST_WITHIN(db2gse.ST_POINT(X1, Y1,1003),AREA) != 0
6   OR db2gse.ST_WITHIN(db2gse.ST_POINT(X1, Y1,1003),AREA) != 0
7   ),
8
9   RawIntersects AS (
10    SELECT ENDPOINT AS SERVICEENDPOINT, TYPE AS CARRIERTYPE,
11          CARRIERNAME,
12
13       db2gse.ST_TOLINESTRING(
14            db2gse.ST_INTERSECTION(
15                db2gse.ST_LineString(
16                    'linestring(XY1, XY2)',1003),
17                AREA)
18      )
19    AS IntersectionLine FROM ContainingAreas
20  ),
21
22  IntersectionTable AS (
23    SELECT SERVICEENDPOINT, CARRIERTYPE,
24       CARRIERNAME, IntersectionLine,
25       db2gse.ST_STARTPOINT(IntersectionLine) AS StartPoint,
26       db2gse.ST_ENDPOINT(IntersectionLine) AS EndPoint
27    FROM RawIntersects
28    WHERE db2gse.ST_LENGTH(InterSectionLine) != 0
29  )
30
31  SELECT
32  IT1.CARRIERNAME AS CARRIERNAME1,
33  IT1.SERVICEENDPOINT AS ENDPOINT1,
34  IT1.CARRIERTYPE AS TYPE1,
35  IT2.CARRIERNAME AS CARRIERNAME2,
36  IT2.SERVICEENDPOINT AS ENDPOINT2,
37  IT2.CARRIERTYPE AS TYPE2,
38  IT1.ENDPOINT AS HANDOVER
39
40  FROM IntersectionTable IT1, IntersectionTable IT2
41  WHERE
42
43  (db2gse.ST_TOUCHES(IT1.Endpoint,IT2.IntersectionLine) = 1
44  OR db2gse.ST_WITHIN(IT1.ENDPOINT,IT2.IntersectionLine) = 1)
45  AND db2gse.ST_EQUALS(IT1.ENDPOINT, IT2.ENDPOINT) != 1
46  AND db2gse.ST_EQUALS(IT1.STARTPOINT, IT2.STARTPOINT) != 1
```

Die Zeilen 1 bis 25 sind funktionsgleich mit dem Query aus 4.5.2.2. Deren Resultat wird wiederum in einer virtuellen Tabelle gespeichert. Danach wird nach Anbieterpaaren gesucht, deren Schnittvektoren sich überlappen oder berühren. Im weiteren Verlauf werden diese dann zusammen mit einem, beiden Schnittvektoren gemeinsamen, Punkt ausgegeben.

4.6 Web Service Wrapper

Wie bereits erwähnt müssen Mobilitätsanbieter, je nach Typ, einen Webservice Port aus 5.3 implementieren. Da dies vor einer kommerziellen Implementierung und der anschließenden Vermarktung des MoBee Systems jedoch nicht geschehen wird, ist eine Kapselung der real existierenden Webschnittstellen notwendig. Diese Funktion übernehmen die Web Service Wrapper (WSW). Die WSW gliedern sich in fünf Komponenten:

- WS-Port
  Java Implementierung des jeweiligen Porttyps. Entweder Generic LineTrip Port, Generic Carsharing Port oder NavRoute Port. Die Komponente beinhaltet auch Java-Repräsentationen der XML-Datentypen, welche im jeweiligen Porttyp Verwendung finden.

- HTTP-Accessor
  Behandelt die Kommunikation mit dem jeweiligen Anbieter. HTTP-Request entweder als POST oder GET. Beinhaltet teilweise auch einen Trust Manager für SSL-Verbindungen.

- JavaEE Core
  Steuermodul für die restlichen Komponenten, instanziiert diese auch.

- SAX-Parser
  Wandelt den vom Partner übermittelten XML-Stream in das vom Porttyp definierte Datenformat um.

- Partnerspezifische Dienstmodule
  Einige Partner übermitteln ihre Daten in einer Weise, welche nicht nur strukturell, sondern auch bezüglich des Datenformates von der MoBee-Spezifikation abweicht (Positionsdaten in Gauß-Krüger anstatt WGS84, Haltestellenliste unsortiert etc.). Die WSW dieser Partner erfordern folglich zusätzliche Dienstmodule, welche die gelieferten Daten in das korrekte Format umformatieren.

Abbildung 4-11 stellt die Komponenten eines WSW dar und zeigt den Datenfluss eines Querys an den Mobilitätsanbieter. Die Pfeile mit durchgezogener Linie stellen hierbei den Datenfluss vom MoBee System zum Partnerservice dar, gepunktete Pfeile den Datenfluss in die entgegengesetzte Richtung.

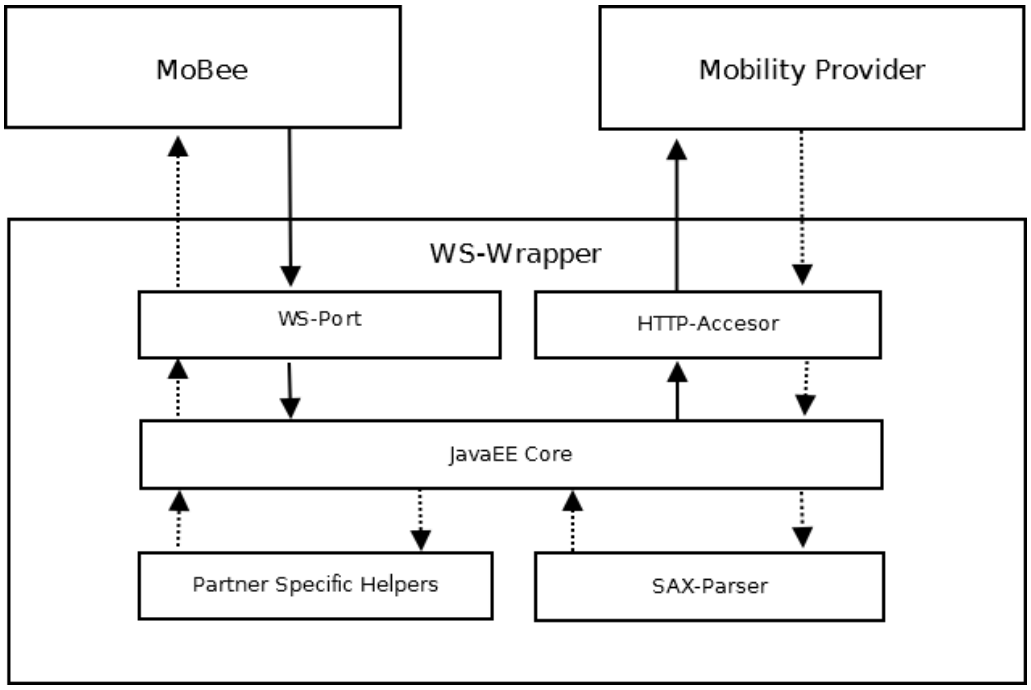


Abbildung 4-11 Architektur der WSW

4.7 Sorting Service

Der Sorting Service stellt einen Sonderfall der WSW dar. Anstatt die Datenausgabe des Webdienstes eines Mobilitätsanbieters zu kapseln, formatiert dieser Wrapper die Ausgabe der Prozesslogik. Zu diesem Zwecke beinhaltet die externe Schnittstelle des MoBee Systems (siehe 5.1) die Parameter *SortBy* und *SecondaryCap*. *SortBy* gibt an, nach welcher Metrik die gelieferten Routenvorschläge sortiert werden sollen. Mögliche Werte sind hierbei die Werte *DURATION* (Fahrtdauer) und *FARE* (Fahrtkosten). *SecondaryCap* dient dazu, jene Routenvorschläge herauszufiltern, welche in der sekundären Metrik stark vom Durchschnittswert aller berechneten Routenvorschläge abweichen. Hierfür wird als Parameter *SecondaryCap* ein Integer übergeben, der einen Prozentsatz des Durchschnittes angibt, welchen die sekundäre Metrik nicht überschreiten darf.

5 Webservice Ports

Im folgenden Kapitel werden die diversen Webserviceschnittstellen beschrieben, welche MoBee für die interne Kommunikation mit seinen Subkomponenten und die Kommunikation mit externen Anbietern verwendet.

5.1 TripQuery Wrapper (Hauptschnittstelle)

Der TripQuery Wrapper Port stellt die Hauptschnittstelle des MoBee Systems dar. Über diesen werden Routenanfragen an das System gestellt und Routenvorschläge zurückgegeben.

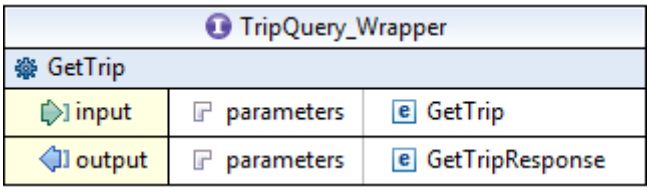


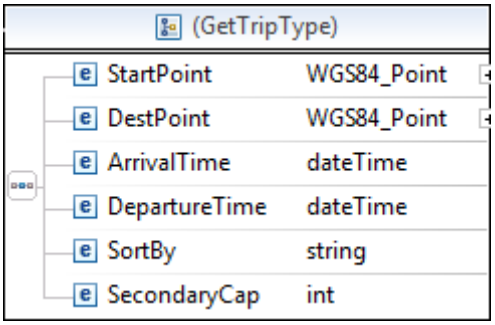
Abbildung 5-1 TripQuery Wrapper Port

5.1.1 Operationen

- GetTrip
  Liefert Routenvorschläge abhängig von Start- und Zielkoordinaten und Sortierparametern

5.1.2 Datentypen

GetTripType



Funktion:

Containerelement für die Eingabedaten der Operation *GetTrip*

Felder:

- StartPoint (WGS84_Point)
  Koordinaten des Startpunktes für die zu ermittelnden Routenvorschläge

- DestPoint (WGS84_Point)
  Koordinaten des Zielpunktes für die zu ermittelnden Routenvorschläge

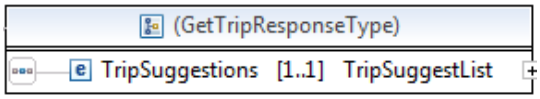
- ArrivalTime (dateTime)
  Zielankunftszeit für die zu ermittelnden Routenvorschläge

- DepartureTime (dateTime)
  Zielabfahrtszeit für die zu ermittelnden Routenvorschläge

- SortBy(string)
  ENUM, welcher die Metrik angibt, nach welcher sortiert werden soll

- SecondaryCap(int)
  Obergrenze für sekundäre Metriken

GetTripResponseType



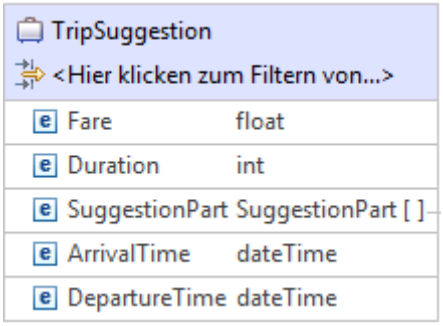
Funktion:

Containerelement für die Ausgabedaten der Operation *GetTrip*

Felder:

- TripSuggestions (TripSuggestList)
  Liste von *TripSuggestion*

TripSuggestion



Funktion:

Enthält die Eckdaten (Fahrtzeit, Kosten) eines Routenvorschlages, sowie eine Liste von Teilstrecken, welche zusammen die vorgeschlagene Route ergeben.

Felder:

- Fare (float)
  Fahrtkosten des Routenvorschlags in Euro

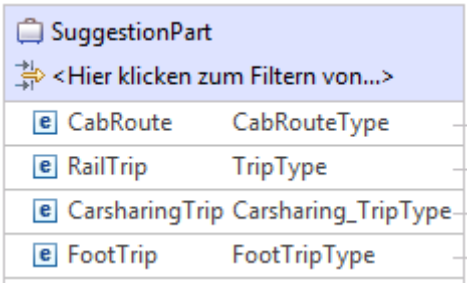
- Duration (int)
  Gesamte Fahrtdauer des Routenvorschlages in Minuten

- SuggestionPart (SuggestionPart[])
  Array von Elementen des Typs *SuggestionPart*. Teilstrecken ergeben zusammen den Routenvorschlag

- ArrivalTime (dateTime)
  Ankunftszeit des Routenvorschlags

- DepartureTime (dateTime)
  Abfahrtszeit des Routenvorschlags

SuggestionPart



Funktion:

Enthält eine Teilstrecke des gesamten Routenvorschlages. Jedes Element vom Typ *SuggestionPart* enthält genau ein Element, welches entweder vom Typ *CabRouteType*,

TripType oder *Carsharing_TripType* ist. Die Datentypen der Sorten von Teilstrecken, welche die Dienste von Mobilitätsanbietern nutzen, sind in 5.3 beschrieben.

Felder:

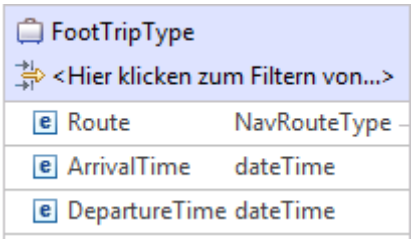
- CabRoute (CabRouteType)
  Teilstrecke, welche mit dem Taxi zurückgelegt wird

- RailTrip (TripType)
  Teilstrecke, welche mit linienbasierten Verkehrsanbietern zurückgelegt wird

- CarsharingTrip (Carsharing_TripType)
  Teilstrecke, welche mit einem Carsharingauto zurückgelegt wird

- FootTrip (FootTripType)
  Teilstrecke, welche zu Fuß zurückgelegt wird

FootTripType



Funktion:

Enthält eine Teilstrecke, welche zu Fuß zurückgelegt wird

Felder:

- Route (NavRouteType)
  Route, welche gelaufen werden muss (siehe 5.3.3)

- ArrivalTime (dateTime)
  Ankunftszeit der Laufstrecke

- DepartureTime (dateTime)
  Startzeit der Laufstrecke

5.2 Database Connector (GSS Kommunikation)

Der Database Connector Port regelt die Kommunikation des MoBee-Hauptprozesses mit dem DB2-Server. Er stellt Operationen zur Verfügung, welche Informationen über Mobilitätsanbieter und ihre Availability Areas in Bezug auf geodätische Punkte liefern.

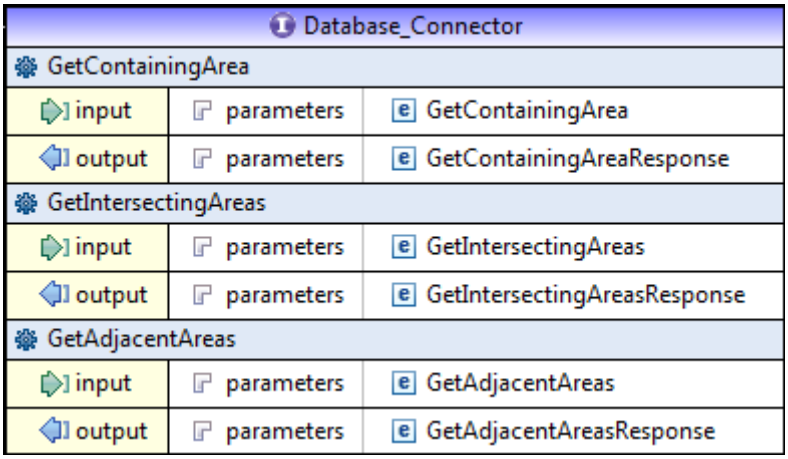


Abbildung 5-2 Database Connector Port

5.2.1 Operationen

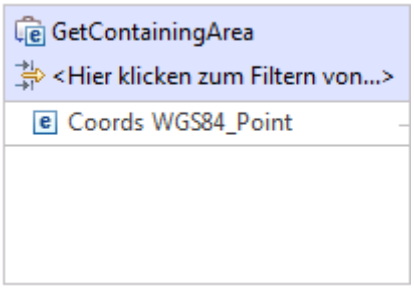
- GetContainingArea
  Liefert alle Availability Areas, welche einen übergebenen Punkt einschliessen

- GetIntersectionAreas
  Liefert alle Availability Areas, welche von einem übergebenen Routenvektor geschnitten werden, zusammen mit den Schnittvektoren und ihren Start- und Endpunkten

- GetAdjacentAreas
  Liefert eine List von Anbieterpaaren, welche jeweils einen Routenvektor zusammen abdecken

5.2.2 Datentypen

GetContainingArea



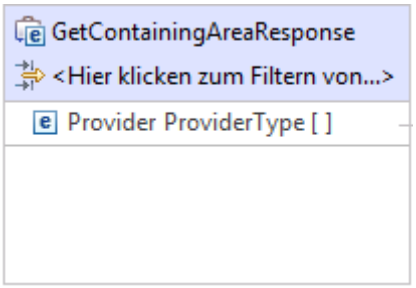
Funktion:

Containerelement für die Eingabedaten der Operation *GetContainingArea*

Felder:

- Coords (WGS84_Point)
  Diejenigen Coordinaten, auf die die Availability Areas geprüft werden sollen

GetContainingAreasResponse



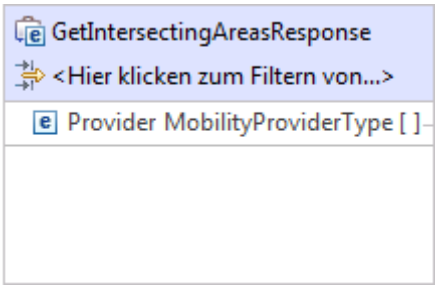
Funktion:

Containerelement für die Ausgabedaten der Operation *GetContainingArea*

Felder:

- Provider (ProviderType[])
  Array von Providern, dessen Availability Areas den Eingabepunkt enthalten

GetIntersectingAreasResponse



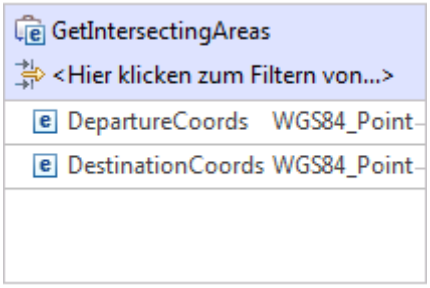
Funktion:

Containerelement für die Ausgabeoparameter der Operation *GetIntersectingAreas*

Felder:

- Provider (MobilityProviderType[])
  Array von Mobilitätsanbietern inklusive ihrer Schnittvektoren

GetIntersectingAreas



Funktion:

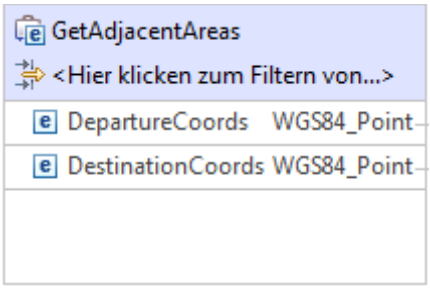
Containerelement für die Eingabedaten der Operation *GetIntersectingAreas*

Felder:

- DepartureCoords (WGS84_Point)
  Startpunkt für den Routenvektor

- DestinationCoords (WGS84_Point)
  Endpunkt für den Routenvektor

GetAdjacentAreas



Funktion:

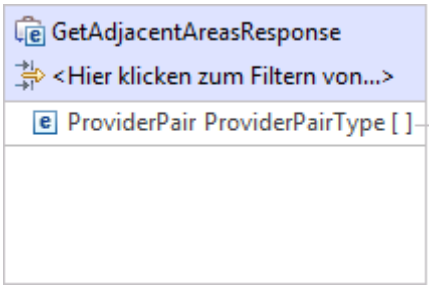
Containerelement für die Eingabedaten der Operation *GetAdjacentAreas*

Felder:

- DepartureCoords (WGS84_Point)
  Startpunkt des abzudeckenden Routenvektors

- DestinationCoords (WGS84_Point)
  Endpunkt des abzudeckenden Routenvektors

GetAdjacentAreasResponse



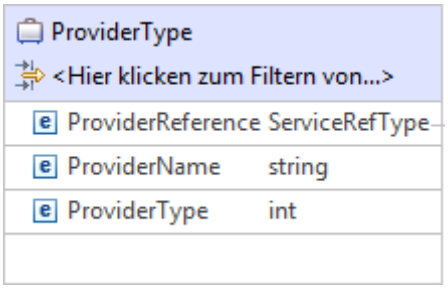
Funktion:

Containerelement für die Ausgabedaten der Operation *GetAdjacentAreas*

Felder:

- Providerpair (ProviderPairType[])
  Array von Paaren von Mobilitätsanbietern inklusive der ihnen zugehörigen Schnittvektoren.

ProviderType



Funktion:

Datenelement für das Kapseln einer Servicereferenz zusammen mit dem Klartextnamen des Mobilitätsanbieters und dessen Typ.

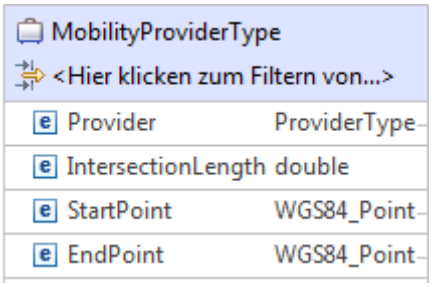
Felder:

- Provider Reference(ServiceRefType)
  Referenz auf den dem Mobilitätsanbieter zugehörigen Webservice

- ProviderName(string)
  Anzeigename des Mobilitätsanbieters

- ProviderType(int)
  MoBee-interne Klassifizierung des Anbietertyps

MobilityProviderType



Funktion:

Datenelement für die Kapselung eines Mobilitätsanbieters mit dessen Schnittvektor und dessen Start- und Endpunkt.

Felder:

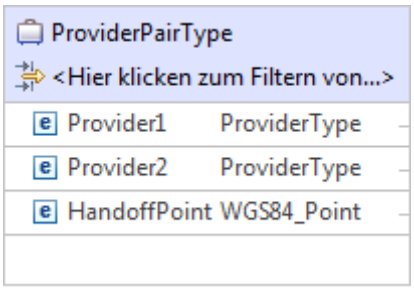
- Provider (ProviderType)
  Enhält die Servicereferenz, den Klartextnamen und den Typ des Mobilitätsanbieters

- IntersectionLength (double)
  Länge des Schnittvektors

- StartPoint (WGS84_Point)
  Startpunkt des Schnittvektors

- EndPoint (WGS84_Point)
  Endpunkt des Schnittvektors

ProviderPair



Funktion:

Datenelement, welches zwei Mobilitätsanbieter beinhaltet, die einen Routenvektor gemeinsam abdecken, zusammen mit einem Übergabepunkt.

Felder:

- Provider1 (ProviderType)
  Mobilitätsanbieter, welcher den ersten Teil des Routenvektors abdeckt

- Provider2 (ProviderType)
  Mobilitätsanbieter, welcher den zweiten Teil des Routenvektors abdeckt

- HandoffPoint (WGS84_Point)
  Übergabepunkt zwischen den Verfügbarkeitsbereichen der Mobilitätsanbieter

5.3 Externe Webservice Ports

Um mit dem MoBee-System zu kommunizieren, müssen Webservices, je nach ihrer Funktionalität, einen von drei Webservice Ports implementieren. Diese werden im Folgenden, zusammen mit den ihnen zu Grunde liegenden Datentypen, beschrieben.

5.3.1 Generic LineTrip Port

Der Generic LineTrip Port ist der Porttyp für linienbasierte Mobilitätsanbieter. Er bietet Funktionalität für das Auffinden von Knoten (Haltestellen) in Abhängigkeit von WGS84-Koordinaten und dem Abruf von Strecken zwischen zwei aufgefundenen Knoten.

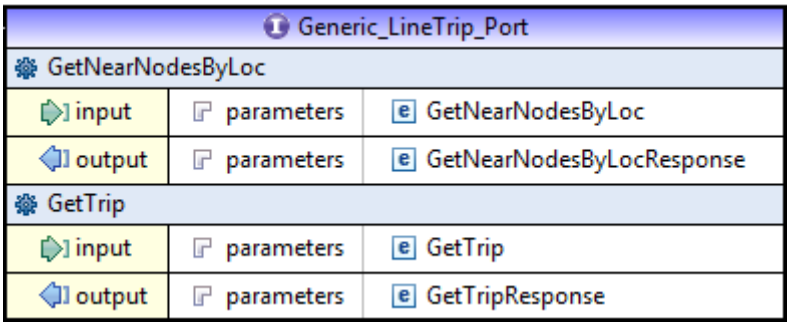


Abbildung 5-3 Generic LineTrip Port

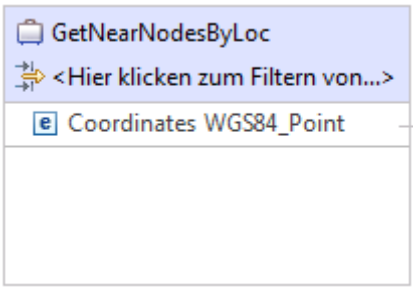
5.3.1.1 Operationen:

- GetNearNodesByLoc
  Gibt einer WGS84-Koordinate naheliegende Knoten des Mobilitätsanbieters zurück

- GetTrip
  Gibt Routenvorschläge zwischen zwei Knoten des Mobilitätsanbieters zurück

5.3.1.2 Datentypen:

GetNearNodesByLoc



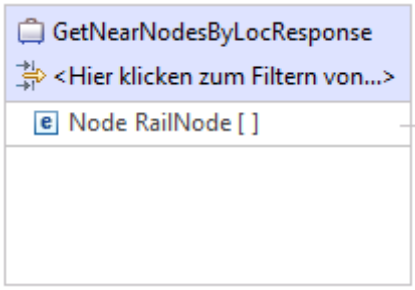
Funktion:

Containerelement für die Eingabedaten der Operation *GetNearNodesByLoc*

Felder:

- Coordinates (WGS84_Point)
  Diejenigen Koordinaten, in dessen Nähe Knoten gefunden werden sollen

GetNearNodesByLocResponse



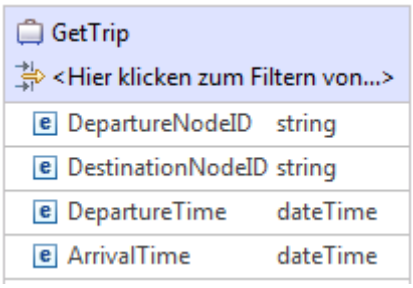
Funktion:

Containerelement für die Ausgabedaten der Operation *GetNearNodesByLoc*

Felder:

- Node (RailNode[])
  Array von Knoten, welche den übergebenen Koordinaten nahe liegen

GetTrip



Funktion:

Containerelement für die Eingabedaten der Operation *GetTrip*

Felder:

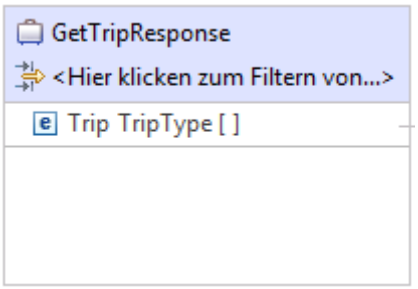
- DepartureNodeID (string)
  Serviceinterner Identifikationsstring für den Abfahrtknoten

- DestinationNodeID (string)
  Serviceinterner Identifikationsstring für den Zielknoten

- DepartureTime(dateTime)
  Zielabfahrtszeit

- ArrivalTime(DateTime)
  Zielankunftszeit

GetTripResponse



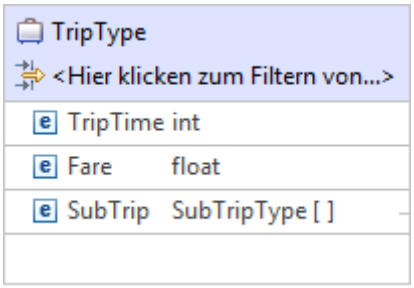
Funktion:

Containerelement für die Ausgabedaten der Operation *GetTrip*

Felder:

- Trip (TripType[])
  Dient der Rückgabe von vorgeschlagenen Routen zwischen den übergebenen Haltestellen

TripType



Funktion:

Enthält Fahrtzeit und Fahrtkosten, sowie die einzelnen Teilstrecken einer vorgeschlagenen Reiseroute

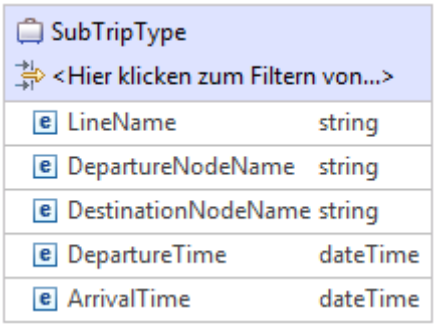
Felder:

- TripTime (int)
  Komplette Fahrtzeit der vorgeschlagenen Route in Minuten

- Fare (float)
  Kosten der Fahrt in Euro

- SubTrip (SubTripType[])
  Teilrouten der Fahrtstrecke

SubTripType



Funktion:

Enthält die Parameter einer Teilroute

Felder:

- LineName (string)
  Name der Linie der Teilroute (z.B. U14)

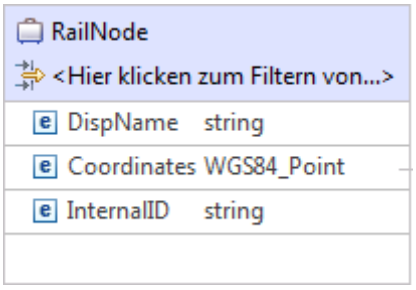
- DepartureNodeName (string)
  Anzeigename der Abfahrtshaltestelle

- DestinationNodeName (string)
  Anzeigename der Zielhaltestelle

- DepartureTime (dateTime)
  Abfahrtszeit der Teilroute

- ArrivalTime (dateTime)
  Ankunftszeit der Teilroute

RailNode



Funktion:

Enthält die relevanten Daten eines Knotens, sowohl für die MoBee-interne Verarbeitung, als auch für die Rückgabe an den externen Webservice.

Felder:

- DispName (string)
  Anzeigename des Knotens

- Coordinates (WGS84_Point)
  WGS84-Koordinaten des Knotens für die MoBee-interne Verarbeitung

- InternalID (string)
  Interner Identifikationsstring des Knotens

5.3.2 Generic_Carsharing Port

Der Generic Carsharing Port ist der Porttyp für die Kommunikation mit Carsharinganbietern. Er bietet Funktionalität für das Auffinden von verfügbaren Fahrzeugen.

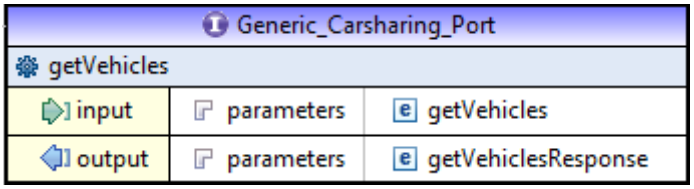


Abbildung 5-4 Generic_Carsharing Port

5.3.2.1 Operationen

- getVehicles
  Gibt eine Liste von verfügbaren Fahrzeugen zurück, sortiert nach Nähe zu den Zielkoordinaten.

5.3.2.2 Datentypen

getVehicles

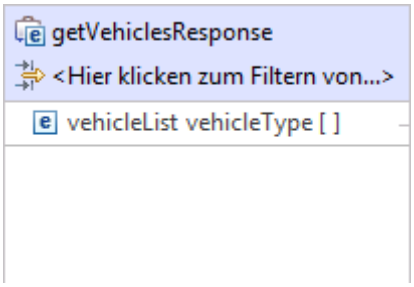
Funktion:

Containerelement für die Eingabedaten der Operation *getVehicles*

Felder:

- Coordinates (WGS84_Point)
  Koordinaten, in deren Nähe nach verfügbaren Fahrzeugen gesucht werden soll

getVehiclesResponse



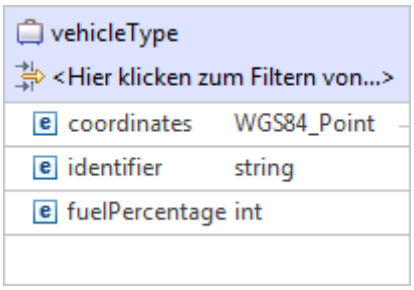
Funktion:

Containerelement für die Ausgabedaten der Operation getVehicles

Felder:

- VehicleList (vehicleType)
  Array, welches die gefundenen Farhzeuge enthält

vehicleType



Funktion:

Enthält die zur Weiterverarbeitung notwendigen Daten eines gefundenen Fahrzeugs

Felder:

- Coordinates (WGS84_Point)
  WGS84-Koordinaten des Fahrzeugs

- Identifier (string)
  Serviceinterner Identifikationsstring des Fahrzeugs

- FuelPercentage (int)
  Prozentualer Treibstofffüllstand des Fahrzeugs

5.3.3 NavRoute Port

Der Navigations- und Routingport (NavRoute) behandelt die Kommunikation mit einem Routingservice für straßengebundene Verkehrsmittel. Er bietet Funktionalität für den Abruf von Streckeninformationen.

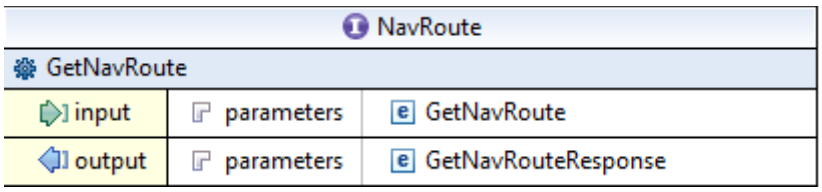


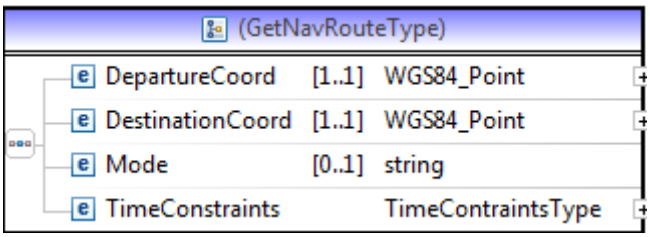
Abbildung 5-5 NavRoute Port

5.3.3.1 Operationen

- GetNavRoute
  Gibt eine Straßenroute zwischen zwei WGS84-Koordinaten zurück.

5.3.3.2 Datentypen

GetNavRoute



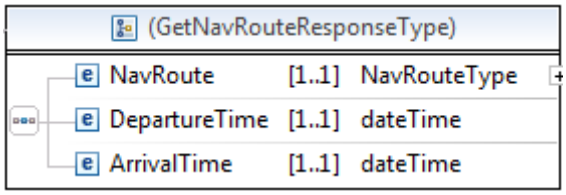
Funktion:

Containerelement für die Eingabedaten der Operation *GetNavRoute*

Felder:

- DepartureCoord (WGS84_Point)
  Abfahrtskoordinaten für die gelieferte Route
- DestinationCoord (WGS84_Point)
  Ankunftskoordinaten für die gelieferte Route
- Mode (string)
  Serviceinterner Identifikationsstring für den Navigationsmodus (fahren, laufen)
- TimeConstraints (TimeConstraintsType)
  Zeitbeschränkungen (Ankunft, Abfahrt) für die gelieferte Route

GetNavRouteResponse



Funktion:

Containerelement für die Ausgabedaten der Operation *GetNavRoute*

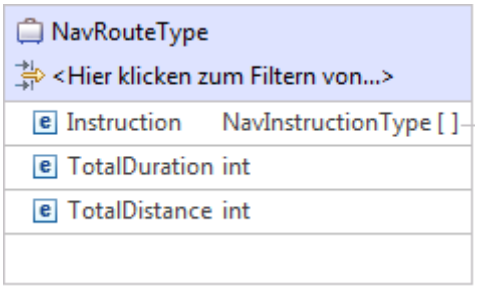
Felder:

- NavRoute (NavRouteType)
  Containerelement für die vorgeschlagene Route

- DepartureTime (dateTime)
  Abfahrtszeit

- ArrivalTime (dateTime)
  Ankunftszeit

NavRoute



Funktion:

Containerelement für die vorgeschlagene Route.

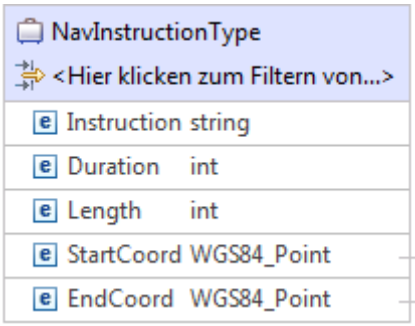
Felder:

- Instruction (NavInstructionType[])
  Array von Teilstrecken/Fahranweisungen

- TotalDuration (int)
  Gesamtfahrtzeit in Minuten

- TotalDistance (int)
  Gesamtfahrtstrecke in Metern

NavInstructionType



Funktion:

Containerelement für Teilstrecken/Fahrtanweisungen

Felder:

- Instruction (string)
  Fahrtanweisung in natürlicher Sprache

- Duration (int)
  Dauer der Teilstrecke in Sekunden

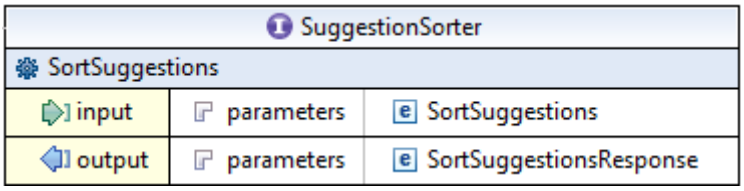
- Length (int)
  Streckenlänge in Metern

- StartCoord (WGS84_Point)
  Abfahrtskoordinate der Teilstrecke

- EndCoord (WGS84_Point)
  Zielkoordinate der Teilstrecke

5.4 SuggestionSorter Port (Sorting Service)

Der SuggesionSorter Port dient der Kommunikation der BPEL-Prozesse der Prozesslogik mit dem in 4.7 beschriebenen Sorting Service.

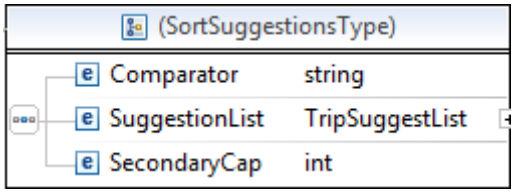


5.4.1 Operationen

- SortSuggestions
  Sortiert eine Liste von Routenvorschlägen nach den übergebenen Parametern

5.4.2 Datentypen

SortSuggestionsType



Funktion:

Containerelement für die Eingabeparameter der Operation *SortSuggestions*

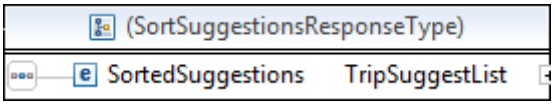
Felder:

- Comparator (string)
  Metrik, anhand derer die übergebenen Routenvorschläge sortiert werden sollen

- SuggestionList (TripSuggestList)
  Liste von Routenvorschlägen, welche sortiert werden soll (siehe 5.1.2)

- SecondaryCap (int)
  Wert ab welchem die sekundären Metriken als Ausschlusskriterium dienen (siehe 4.7)

SortSuggestionsResponseType



Funktion:

Containerelement für die Ausgabedaten der Operation *SortSuggestions*

Felder:

- SortedSuggestions (TripSuggestList)
  Sortierte Liste von Routenvorschlägen

6 Evaluierung

Im folgenden Kapitel werden die Erfahrungen aus dem Test des MoBee Systems, die dabei zu Tage getretenen Stärken und Schwächen des Konzeptes, sowie mögliche Erweiterungen des Konzeptes betrachtet.

Ziel der Arbeit war, wie in 1.1 beschrieben, einen multimodalen Planungsdienst zu Entwickeln, welcher:

- Echte Mulitmodalität anbietet
- Eine große Anzahl von Moblitätsanbietern aufnehmen kann
- Für Bahn, ÖPNV, Carsharing, Taxis und Fußweg einsetzbar ist
- Diese Ziele durch die Aggregation bereits bestehender Webservices erreicht

Diese Kriterien können als erfüllt betrachtet werden. Dennoch sind einige besondere Eigenschaften, positiv wie negativ, zu Tage getreten. Diese sollen im folgenden Kapitel erörtert werden.

6.1 Stärken des MoBee Konzeptes

Im folgenden Abschnitt werden die Stärken des MoBee Konzeptes sowie dessen Implementierung im Rahmen dieser Arbeit betrachtet.

6.1.1 Auflistung von Alternativen für simple Routen

Besonders positiv sind die Ergebnisse des MoBee Konzeptes im Bereich der Verkehrsplanung innerhalb des Verfügbarkeitsbereiches einzelner Mobilitätsanbieter. Dies ist auch nicht weiter verwunderlich, da im Grunde genommen nur die Routenvorschläge der bestehenden Dienste normalisiert und als alternative Routenvorschläge ausgegeben werden. Im Falle von Taxi- und Carsharingunternehmen werden die zur Verfügung gestellten Daten noch um Streckendaten des Routingdienstes erweitert. Dies stellt jedoch auch nur eine Zusammenführung zweier diskreter Datensätze dar und ist deshalb wenig fehleranfällig. Abbildung 6-1 zeigt eine Gegenüberstellung der vorgeschlagenen Routen von MoBee und Moovel für die Strecke zwischen Krefelder Straße 21, Stuttgart und Jasminweg 10, Stuttgart. Zu beachten ist hierbei, dass für die Fahrt mit dem Verkehrsverbund Stuttgart von beiden Systemen dieselbe Route geliefert wurde. Zusätzlich dazu wurde von MoBee direkt in der Krefelder Straße ein car2go Fahrzeug gefunden. Eine Fahrt mit Diesem würde etwas mehr als 20 Minuten früher das Ziel erreichen, als eine Fahrt mit dem von Moovel vorgeschlagenen Taxi.

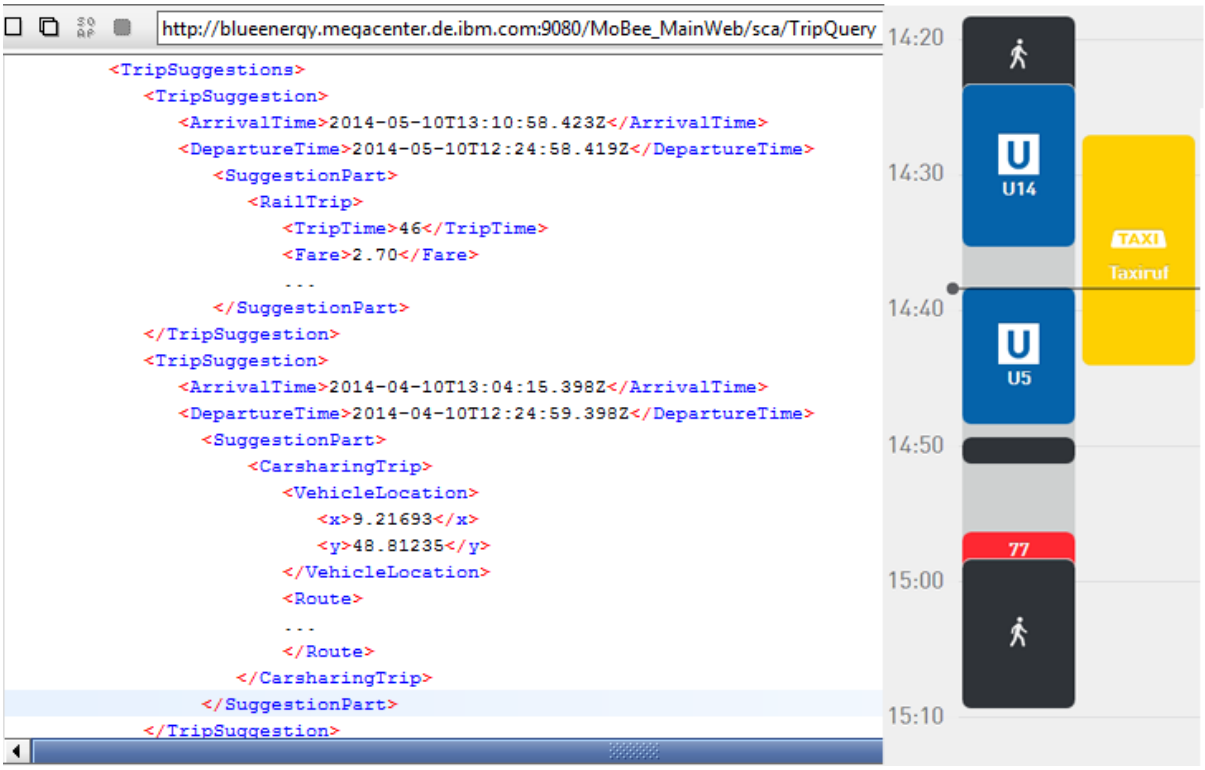


Abbildung 6-1 Vergleich Routingergebnisse MoBee und Moovel

6.1.2 Tiered Queries

Eine weitere hervorzuhebende Stärke von MoBee sind die Tiered Queries des TiGeR-Algorithmus. Die hierdurch gelieferten Routenvorschläge gehen weit über das hinaus, was bisher erhältliche Mobilitätsplanungsdienste ermöglichen: Carsharing- und Taxianbieter, öffentliche Verkehrsmittel auf regionaler und überregionaler Ebene werden zusammengeführt und für kombinierte Routenvorschläge verwendet. Überdies sind die errechneten Routenvorschläge für den Alltagsgebrauch praktikabel.

Da bisher kein anderer verfügbarer Planungsdienst diese Funktionalität anbietet, kann hier leider kein direkter Vergleich zu herkömmlichen Lösungen gezogen werden. Es soll allerdings an dieser Stelle noch einmal das Beispiel aus 4.2.3 herangezogen werden Füttert man die MoBee Implementierung mit den im aufgeführten Beispiel verwendeten Start- und Zielkoordinaten, (47.261957, 11.395971) und (48.810369, 9.212706), entspricht das Ergebnis fast genau dem händisch in 4.2.3 Erhaltenen. Abbildung 6-2 zeigt einen von MoBee gelieferten kombinierten Routenvorschlag für die genannte Strecke.

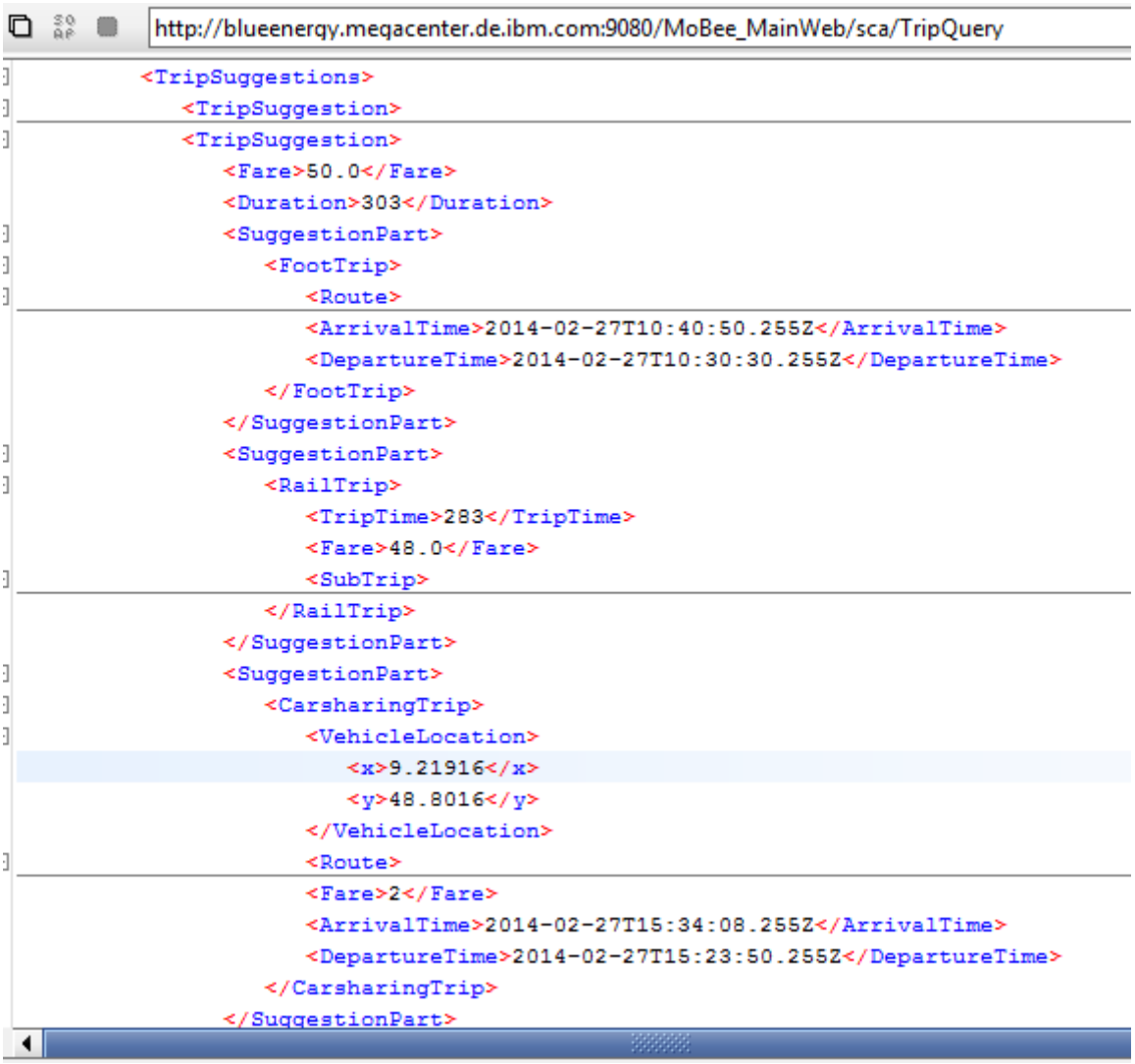


Abbildung 6-2 MoBee Routenvorschlag Innsbruck Hilton nach Krefelder Straße 21, Stuttgart

Als Backbone-Haltestellen werden wie im Beispiel der Innsbrucker Hauptbahnhof und der Bahnhof Bad Cannstatt in Stuttgart verwendet. Es werden folgende kombinierte Routen vorgeschlagen:

- Fußweg – ÖBB – Fußweg
- Fußweg – ÖBB – Carsharing
- Fußweg – ÖBB – Taxi
- Taxi – ÖBB – Fußweg
- Taxi – ÖBB – Carsharing
- Taxi – ÖBB – Taxi

Das Fehlen von kombinierten Routenvorschlägen unter Zuhilfenahme von Typ-1 Mobilitätsanbietern ist auf zweierlei Gründe zurückzuführen: Einerseits wurde, wie im Beispiel vorhergesagt, keine effiziente Route mit dem Verkehrsverbund Stuttgart zwischen dem Cannstatter Bahnhof und der Krefelder Straße gefunden. Zum anderen ist der Verkehrsverbund Tirol nicht an die dieser Arbeit zugrunde liegende Implementierung angeschlossen. Wie in Abbildung 6-3 ersichtlich ist, werden Typ-1 Mobilitätsanbieter auch in die Routenvorschläge mit aufgenommen, so fern eine entsprechende Teilroute gefunden wurde.


```
<TripSuggestion>
   <Fare>9.2</Fare>
   <Duration>72</Duration>
   <SuggestionPart>
      <RailTrip>
         <TripTime>15</TripTime>
         <Fare>2.20</Fare>
         <SubTrip>
            <LineName>U14</LineName>
            <DepartureNodeName>Erwin-Schoettle-Platz</DepartureNodeName
            <DestinationNodeName>Hauptbf (A.-Klett-Pl.)</DestinationNod
            <DepartureTime xsi:type="xsd:dateTime">2014-04-27T14:25:15.
            <ArrivalTime xsi:type="xsd:dateTime">2014-04-27T14:36:15.35
         </SubTrip>
      </RailTrip>
   </SuggestionPart>
   <SuggestionPart>
      <RailTrip>
         <TripTime>37</TripTime>
         <Fare>7.0</Fare>
         <SubTrip>
            <LineName>OEBB mockup line</LineName>
            <DepartureNodeName>Stuttgart Hbf</DepartureNodeName>
            <DestinationNodeName>Tübingen Hbf</DestinationNodeName>
            <DepartureTime xsi:type="xsd:dateTime">2014-04-27T14:30:22.
            <ArrivalTime xsi:type="xsd:dateTime">2014-04-27T15:07:22.25
         </SubTrip>
      </RailTrip>
   </SuggestionPart>
```


Abbildung 6-3 Tiered Trip Stuttgart – Tübingen

Wie bereits erwähnt, existiert zurzeit kein multimodales Verkehrsplanungssystem, welches die in diesem Beispiel verwendete Route komplett planen kann. Um auf die hier gezeigte Route zu kommen, müsste der Fahrgast mindestens drei verschiedene Fahrplansysteme beziehungsweise Lokalisierungsdienste benutzen. Darüber hinaus wäre nicht direkt ersichtlich, welche Kombination von Mobilitätsanbietern die jeweils schnellste oder günstigste Variante darstellt. Um diese Information zu erhalten, würde die Anzahl der zu benutzenden Webdienste nochmals steigen. Zusätzlich müssten diese Dienste alle mehrfach, also einmal pro Anbieterkombination, aufgerufen werden. Schließlich müsste der Endnutzer die Auswahl der optimalen Routenkombination selbst treffen, was erneut mit Mehraufwand verbunden wäre.

6.1.3 Leichte Erweiterbarkeit

Eines der Designprinzipien von MoBee ist die leichte Erweiterbarkeit durch die Verwendung von standardisierten Webservice Ports. Dieses Ziel wurde weitestgehend erreicht: Um dem System einen neuen Mobilitätsanbieter hinzuzufügen genügt es in der Theorie, die Servicereferenz und die Availability Area der Datenbank hinzuzufügen. In der Praxis ist dies allerdings

noch nicht möglich, da kein bestehender Webservice die in dieser Arbeit definierten Ports implementiert. Es stellte sich jedoch während der Implementierungsphase heraus, dass nachdem eine sinnvolle Architektur für die Wrapper Applikationen gefunden wurde, sich der Aufwand für die Integration eines neuen Dienstes sehr in Grenzen hielt. In einer eventuellen kommerziellen Implementierung könnten die Wrapper durchaus ein fester Bestandteil des Systems werden, da der mit ihnen verbundene Entwicklungsaufwand die leichte Erweiterbarkeit des Systems nicht nennenswert beeinträchtigt.

6.1.4 Verzicht auf proprietäre Funktionalität

MoBee kommt, in der jetzigen Implementierung, vollkommen ohne proprietäre Technologien aus. Alle Komponenten der Implementierung, sowohl in der Datenhaltung, als auch in der der Prozesslogik, basieren auf offenen Standards. Die in dieser Arbeit eingesetzte Middleware besteht zwar aus proprietären IBM-Produkten, das System wäre jedoch auch auf einer komplett auf Open Source basierender Middleware lauffähig. Die WebSphere Server könnten zum Beispiel durch Apache Tomcat/ODE [Apa141], der DB2 Server inklusive Spatial Extender durch PostGIS [OSG14] ersetzt werden.

6.2 Schwächen des Konzeptes und Lösungsvorschläge

Im folgenden Abschnitt werden erkannte Schwachstellen des MoBee Konzeptes betrachtet und mögliche Lösungsvorschläge für eine erweiterte Version des Systems besprochen.

6.2.1 Übergabepunktberechnung bei Handoff Queries

Ein großer Schwachpunkt des MoBee-Konzeptes sind die Handoff Queries. Im Falle zweier sich überlappender oder nebeneinander liegender Verfügbarkeitsbereiche von linienbasierten Mobilitätsanbietern liefert das System noch gute Ergebnisse, da die Tatsache ausgenutzt wird, dass alle verfügbaren Fahrplansysteme zumindest in begrenztem Umfang die Linien und Haltestellen ihrer Nachbardienste beinhalten. Die ermittelten Routen gleichen somit denen aus der Berechnung einer simplen Route (vgl. 6.1.1). Sobald jedoch ein dynamisch gerouteter Mobilitätsanbieter mitverwendet wird, sind die von MoBee gelieferten Ergebnisse suboptimal. Der Grund hierfür ist die Ermittlung des Übergabepunkts. Wie in 4.5.2.3 beschrieben, wird hierfür der Mittelpunkt der Luftlinie zwischen Start- und Zielpunkt der zu berechnenden Route verwendet. Da Mobilitätsanbieter mit ihrem Verfügbarkeitsbereich meist einen Ballungsraum abdecken, liegt dieser Punkt meist in spärlich besiedeltem Gebiet. Diesem Punkt nahegelegene Haltestellen sind meist nur über Umwege erreichbar und nur spärlich mit dem restlichen Netz Verbunden. Auch sind hier nur selten verfügbare Fahrzeuge von Typ-0 Mobilitätsdienstleistern (Taxi, Carsharing) zu finden. Routen, welche von Handoff Queries errechnet werden tendieren folglich dazu, extrem zeitaufwendig und somit unbrauchbar zu sein.

Das Problem ließe sich beheben, wenn auf der Seite der linienbasierten Mobilitätsanbieter bekannt wäre, welche Haltestellen aus dem Netz des Mobilitätsanbieters heraus gut erreichbar sind. In diesem Falle könnte im Grenzgebiet der beiden in Frage kommenden Mobilitätsanbieter nach solchen Haltestellen gesucht und deren Koordinaten als Übergabepunkt genutzt werden. So wäre gewährleistet, dass kein Umweg eingeplant wird, nur um einen mehr oder weniger willkürlich ausgewählten Übergabepunkt zu erreichen. Eine Möglichkeit, solche gut verbundenen Haltestellen ausfindig zu machen wäre, den Generic LineTrip Port um eine Opera-

tion zu erweitern, welche die Anzahl an verfügbaren Linien an einer gegebenen Haltestelle ausgibt. Anhand der Anzahl der jeweils eine Haltestelle anfahrenden Linien und eines geeignetes Grenzwertes wäre es so möglich zu erkennen, wie gut verknüpft und somit leicht zu erreichen ein potentieller Übergabepunkt ist. Da alle linienbasierten Mobilitätsanbieter in ihren Webdiensten den Aushangfahrplan verfügbar machen, wäre dies ohne weiteres möglich.

In dem Falle, dass sich die Availability Areas von zwei Typ-0 Mobilitätsanbietern überlappen, könnte ein Vektor gebildet werden, welcher orthogobal zum Routenvektor ist und diesen am von der Datenbank errechneten Übergabepunkt schneidet. Entlang Diesem könnte dann nach Orten gesucht werden, an dem Fahrzeuge des den zweiten Teil der Route bedienenden Anbieters verfügbar sind.

6.2.2 Lange Laufzeit von Routenberechnungen

Die Laufzeit von Routenberechnungen ist mit ca. 15 bis 20 Sekunden in der jetzigen Implementierung des MoBee Systems extrem lang. Dies ist nicht etwa auf komplexe Berechnungen oder eine ineffiziente Programmierung zurückzuführen. Anfragen an die Datenbank werden in etwa einer Sekunde abgearbeitet, die Subprozesse selbst enthalten keine komplexen Kalkulationen. Vielmehr ist der Grund für die lange Laufzeit im Grundkonzept von MoBee zu finden: Der teilweise mehrfache Aufruf mehrerer externer Werbservices. Diese lassen sich, gerade bei der zeitaufwändigen Erstellung von Tiered Queries, nur begrenzt parallelisieren, da die Ausgabewerte vorhergegangener Routenteile für die Anfrage neuer Routenteile benötigt werden.

Ein damit verwandtes Problem ist die Tatsache, dass in der jetzigen Implementierung pro Kombination von Mobilitätsanbietern (zum Beispiel ÖPNV-Bahn-Carsharing) nur ein Routenvorschlag erzeugt wird. Es wäre algorithmisch betrachtet eine Leichtigkeit, an dieser Stelle mehrere leicht zeitversetzte Routenvorschläge anzubieten. Der Entwicklungsaufwand würde sich auf das Hinzufügen einer Iteration beschränken, da zeitversetzte Routenvorschläge von allen Mobilitätsanbietern angeboten werden und auch bereits im Datenmodell von MoBee integriert sind. Das Problem ist vielmehr, dass der Zeitaufwand für die Routenberechnung auf mehrere Minuten anschwellen würde und das System damit für den Praxisgebrauch ungeeignet werden würde.

Es gibt mehrere Möglichkeiten, dem MoBee System einen höheren Grad an Parallelität und somit eine kürzere Laufzeit zu ermöglichen.

1. **Parallelisierung der Subprozesse für simple und kombinierte Routenerzeugung**.

   Durch ein gleichzeitiges Ausführen dieser Subprozesse könnte mit der Bildung von kombinierten Routen begonnen werden bevor bekannt ist, ob eine simple Route zum Ziel existiert. Sollte eine solche simple Route gefunden werden, würden die Subprozesse für die Bildung kombinierter Routen terminiert und die simple Route als Routenvorschlag ausgegeben werden.

2. **Heuristische Ermittlung von Teilstrecken beim TieredTrip Subprozess**

   Anstatt bei der Bildung eines TieredTrip jeweils die Ankunftszeit der vorangegangenen Teilstrecke als Abfahrtszeit für die nächste Teilstrecke zu nutzen, könnte auf Basis der Länge des Routenvektors zu Beginn der Routenbildung ein Schätzwert für die Fahrtzeit der Teilstrecken gebildet werden. Auf Basis dieser Schätzwerte könnten die Anfragen an die Webdienste der Mobilitätsanbieter parallel ausgeführt werden. Linienbasierte Verkehrsanbieter geben üblicherweise über ihre Webdienste mehrere Routenvorschläge um die gewählte Fahrtzeit herum an. Im Anschluss an die Abfrage aller Teilstrecken von den Webdiensten könnten dann diejenigen ausgewählt werden, welche auf die tatsächliche Ankunftszeit der vorangegangenen Teilstrecke passen. Bei Typ-0 Anbietern ist die ermittelte Route nicht zeitabhängig, die Diskrepanz zwischen der geschätzten und tatsächlichen Ankunftszeit der vorherigen Teilstrecke könnte folglich ohne Weiteres auf Abfahrts- und Ankunftszeit des Typ-0 Streckenabschnittes aufgerechnet werden.

6.2.3 Verfügbarkeit von geteilten Fahrzeugen bei kombinierten Routen

Ein weiteres Problem mit dem MoBee Konzept stellen geteilte Fahrzeuge, wie zum Beispiel Carsharingautos oder Leihfahrräder dar. Wenn eine kombinierte Route (Tiered Query oder Handoff Query) eine Route mit einem geteilten Fahrzeug als nicht-erstem Streckenabschnitt erzeugt, werden für die Routenberechnung die Standorte der Fahrzeuge zum Erstellungszeitpunkt der Route herangezogen. Da diese Fahrzeuge jedoch zu jedem Zeitpunkt von einem anderen Nutzer des Verleihdienstes ausgeliehen und somit von ihrem Standort entfernt werden können, ist nicht sichergestellt, dass die für die Routenberechnung genutzten Fahrzeuge zum Zeitpunkt des Eintreffens des Fahrgastes noch vorhanden sind. Je früher im Voraus eine Route geplant werden soll, desto wahrscheinlicher wird es, dass das für die Planung genutzte Fahrzeug nicht mehr verfügbar ist.

Diese Problematik lässt sich nicht vollständig beheben. Es könnte jedoch dadurch abgemildert werden, dass an potentiellen Übergabepunkten nicht nur nach der Verfügbarkeit von Fahrzeugen gesucht wird, sondern auch deren Anzahl kalkuliert wird. Unter Zuhilfenahme eines Erfahrungswertes über die Fluktuationsrate bei Fahrzeugen des betreffenden Anbietertyps könnte somit eine Wahrscheinlichkeit dafür errechnet werden, dass bei Ankunft des Fahrgastes noch ein Fahrzeug des Anbieters zur Verfügung steht. Diese Vorgehensweise würde das angesprochene Problem allerdings nur bei zeitnahen Routenvorschlägen abmildern. Für die längerfristige Routenplanung wäre eine Datenbasis über die wahrscheinlichen Aufenthaltsorte von Fahrzeugen zu einer bestimmten Zeit vonnöten. Dies würde allerdings das Konzept der Routenberechnung auf Basis von Webserviceaggregation, welches dem MoBee System zugrunde liegt, aushebeln.

6.2.4 Ineffiziente Routen durch Unkenntnis der Haltestellen bei linienbasierten Verkehrsanbietern

Die Unkenntnis über die tatsächlich befahrenen Haltestellen bei linienbasierten Verkehrsanbietern führt zu teilweise ineffizienten Routen: Es kann vorkommen, dass die vom Mobilitäts-

anbieter angebotene Verkehrslinie eine suboptimale Strecke abfährt, um einen bestimmten räumlichen Punkt zu bedienen. Abbildung 6-4 zeigt eine solche ineffiziente Route.

Abbildung 6-4 Ineffiziente Route mit linienbasiertem Mobilitätsanbieter

Würde der Fahrgast an einer früheren Haltestelle vom Linienbasierten Mobilitätsanbieter auf einen dynamisch gerouteten Verkehrsanbieter umsteigen, könnte diese ineffiziente Teilstecke umgangen werden. Abbildung 6-5 zeigt eine solche Verbesserung unter Zuhilfenahme eines dynamisch gerouteten Verkehrsanbieters.

Abbildung 6-5 Theoretisch mögliche bessere Route mit dynamisch gerouteten Mobilitätsanbieter

Da MoBee allerdings keine Kenntnis über die tatsächlich befahrenen Haltestellen besitzt, gibt es keine Möglichkeit, diese möglichen Umstiegspunkte zu finden. Somit wird diese theoretisch durch Multimodalität ermöglichte Möglichkeit der Effizienzsteigerung verschenkt.

Um dieses Problem zu beheben, müsste zuerst einmal MoBee Kenntnis über die befahrenen Knoten einer Strecke erlangen. Die optimale Lösung wäre hierfür eine Erweiterung des Generic LineTrip Ports um eine Auflistung aller befahrenen Haltestellen einer vorgeschlagenen Route. Dies wäre technisch leicht umzusetzen, jedoch setzt es eine tatsächliche Implementierung des Porttyps seitens der Partnerunternehmen voraus. Beschränkt man sich auf die tatsächliche Funktionalität der vorhandenen Webdienste der Mobilitätsanbieter, so kommt nur eine Ermittlung vonseiten des MoBee Systems selbst in Betracht. Ein möglicher Weg, dies zu bewerkstelligen, wäre folgender Algorithmus:

1. Finde alle Knoten, welche entlang der Fahrtstrecke liegen (hierfür wäre eine Kombination aus dem bereits erstellten Routenvektor und der bereits vorhandenen Operation GetNearNodesByLoc aus 5.3.1 eine Option).

2. Prüfe, ob diese von der vom GetSimpleTrip Subprozess vorgeschlagenen Linie angefahren werden (hierfür wäre eine Erweiterung des Generic LineTrip Port um eine Operation notwendig, welche alle verfügbaren Linien an einer Haltestelle zurückgibt).

3. Erzeuge mithilfe des Routenvektors und den angefahrenen Haltestellen eine geordnete Liste von Haltestellen, welche die Fahrtstrecke abbilden.

Nachdem nun eine geordnete Liste der angefahrenen Haltestellen vorliegt, muss diese auf ineffiziente Streckensegmente untersucht werden. Die simpelste Variante wäre, ausgehend vom Zielpunkt alle Haltestellen als Startpunkte für neue Teilstrecken mit dynamisch gerouteten Mobilitätsanbietern zu verwenden und die sich daraus ergebenen kombinierten Routen mit der Originalroute, welche nur den linienbasierten Mobilitätsanbieter verwendet, zu vergleichen. Dieser Ansatz würde zwar unweigerlich die effizienteste mögliche Route hervorbringen, jedoch würde er auch eine große Zahl zusätzlicher Aufrufe des Routingdienstes nach sich ziehen. In Anbetracht der in 6.2.2 angesprochenen Problematik und der Tatsache, das bei einer kommerziellen Implementierung des MoBee Konzepts sicherlich Gebühren für die Nutzung des dynamischen Routingdienstes anfallen würden, ist dieser Ansatz jedoch problematisch. Ein eleganterer Ansatz wäre, die angefahrenen Haltestellen darauf zu untersuchen, inwieweit sich der Transfer von einer Haltestelle zur Nächsten darauf auswirkt, dem Ziel näher zu kommen. Wird so eine Menge von Haltestellen identifiziert, deren Durchfahren den Fahrgast weiter vom Ziel entfernt oder das Ziel überschießt, so wird die vorhergegangene Haltestelle als Startpunkt für eine neue Teilstrecke mit einem dynamisch gerouteten Mobilitätsanbieter genutzt.

6.2.5 Unrealistische Routingvorschläge durch Unkenntnis über die Verkehrssituation

Dynamisch geroutete Mobilitätstypen sind, bezüglich der veranschlagten Reisedauer, von der aktuellen Verkehrslage im zu durchfahrenen Straßenabschnitt abhängig. Gerade in Kombination mit linienbasierten Mobilitätstypen stellt dies ein Problem dar, da schon eine Abweichung der Reisedauer von wenigen Minuten vom errechneten Wert zu einem Verpassen der Abfahrtszeit des linienbasierten Mobilitätsanbieters führen kann.

Um dieses Problem abzumildern müssten Daten über die aktuelle Verkehrslage in die Routenberechnung mit einfließen. Dies wäre zum Beispiel über eine Anbindung an regionale Verkehrsleitzentralen möglich. Allerdings wäre es nicht ohne weiteres zu bewerkstelligen, anhand von Daten über Verkehrsstaus akkurate Schätzungen über die Zeitverzögerung auf der vorgeschlagenen Route zu liefern. Auch ein Umfahren der als von Staus betroffenen gemeldeten Straßen wäre schwierig, da Routingdienste bisher nicht die Möglichkeit anbieten, bestimmte Straßenabschnitte zu umfahren. Die einzige Möglichkeit auf dieses Problem einzugehen wäre dementsprechend, den Routenvorschlag mit einer Warnmeldung zu versehen, dass es auf der angegebenen Fahrtstrecke zu Verzögerungen kommen kann.

7 Ausblick und Zusammenfassung

Im folgenden Kapitel werden Erweiterungsvorschläge besprochen, welche das MoBee Konzept in Zukunft bereichern könnten und Designvorschläge für diese Erweiterungen kurz angerissen. Anschließend wird eine kurze Zusammenfassung der in dieser Arbeit gewonnenen Kenntnisse geboten.

7.1 Direkte Buchung von vorgeschlagenen Routen

Die Möglichkeit, von MoBee vorgeschlagene Reiserouten direkt aus dem System heraus zu buchen, würde einen erheblichen Mehrwert für das Konzept bedeuten. Gerade bei TieredTrips könnte es sich für den Endnutzer als mühsam herausstellen, bei drei oder mehr verschiedenen Mobilitätsanbietern eine Teilstrecke zu buchen. Hierfür könnte ein zentrales Buchungssystem konzipiert werden, welches die Buchungen bei allen an einer Route beteiligten Mobilitätsanbietern vornimmt. Der Endkunde müsste sich mit seinen Kontodaten bei MoBee oder einem den Dienst von MoBee nutzenden Drittanbieter registrieren. Dann könnten die Daten von MoBee an alle beteiligten Mobilitätsanbieter weitergeleitet werden, welche im Anschluss die Abbuchung vornehmen. Um dies umzusetzen, sind die bestehenden Webdienste der Mobilitätsanbieter jedoch ungeeignet. Diejenigen, welche eine Onlinebuchung annehmen, benötigen immer eine Registrierung des Fahrgastes beim Mobilitätsanbieter selbst. Es ist somit nicht möglich, die Buchung vorzunehmen, ohne das MoBee System mit einem eigenen Konto und der dazugehörigen Verwaltungssoftware auszustatten. Ein konkreter Designvorschlag für ein System dieses Umfanges, welches sich zudem mit einer anderen Softwaredomäne (Finanzen, Transactions, Sicherheit) befasst, würde den Rahmen dieses Kapitels, ja dieser Arbeit sprengen. Deshalb wird an dieser Stelle auf einen konkreten Designvorschlag verzichtet.

7.2 Aufnahme von Leihfahrrädern in die Routenplanung

Leihfahrräder (wie zum Beispiel Call-a-Bike) sind zurzeit nicht in der Routenplanung von MoBee integriert. Dies liegt daran, dass kein Anbieter gewillt war, im Zuge einer Diplomarbeit seine Webschnittstelle offen zu legen. Es wäre jedoch algorithmisch ohne Weiteres möglich Leihfahrräder in die Routenplanung mit aufzunehmen: Es müssten in der Datenbank lediglich Anbieter als Typ-0 Mobilitätsanbieter gespeichert werden. Das einzige Hindernis stellt das Routing der Radstrecken dar. Zurzeit bietet der NavRoute Service nur Streckenführung für PKW und Fußgänger. Es wäre jedoch ohne großen Aufwand möglich, den NavRoute Service um diese Funktionalität zu erweitern. Zu diesem Zwecke müssten nicht einmal mehr die vorhandene Schnittstelle für dynamische Routingdienste erweitert werden, da diese bereits einen Paramater für die Auswahl des Routentyps besitzt (siehe 5.3.3).

GetBikeTrip		
Eingaben	DepartureCoords	WGS84_Point
	DestinationCoords	WGS84_Point
	ProviderEndpoint	ServiceRefType
	TimeConstraints	TimeContraintsType
Ausgaben	NavRoute	NavRouteType
	DepartureTime	dateTime
	ArrivalTime	dateTime
	DepotCoordinates	WGS84_Point

Abbildung 7-1 Mögliche Schnittstelle für einen Leihfahrrad-Subprozess

Abbildung 7-1 zeigt eine mögliche Schnittstelle für einen *BikeTrip_Query*-Subprozess. Dieser wäre analog zum *Carsharing_Trip* Query implementiert. Eingabeparameter wären Start- und Zielkoordinaten, die Zeitbegrenzungen (Abfahrts- und Ankunftszeit) sowie ein Verweis auf den Webservice des Mobilitätsanbieters. Die Ausgabe würde neben der Fahrradroute als NavRoute Element (siehe 5.3.3.2) Abfahrts- und Ankunftszeit sowie die Koordinaten der Verleihstelle der Leihfahrräder beinhalten. Der Zugriff auf die Funktionalität des Mobilitätsanbieters könnte über die in Abbildung 7-2 gezeigte generische Schnittstelle erfolgen.

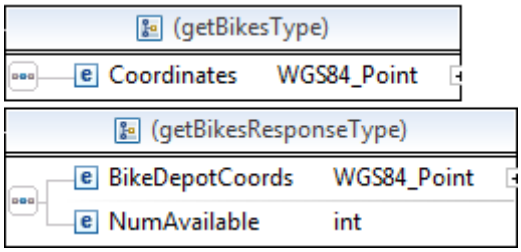


Abbildung 7-2 Generische Schnittstelle für Fahrradverleihanbieter

Darüber hinaus wäre es eventuell sinnvoll zu evaluieren, ob für den Fahrradverleih ein neuer Typ von Mobilitätsanbieter definiert werden sollte. Im Rahmen von örtlich begrenzter Routenplanung könnte der Fahrradverleih wie bereits erwähnt problemlos als Typ-0 Anbieter integriert werden. Bei Regionen übergreifender Routenplanung, vor allem bei Handoff Queries, müsste allerdings gegebenenfalls zwischen motorisierten und nichtmotorisierten Mobilitätsanbietern unterschieden werden. Es ist wenig sinnvoll und vor allem für einige Nutzer des Systems nicht möglich, eine derart lange Strecke mit dem Fahrrad zurückzulegen. Dieser Überlegung entgegen steht die Tatsache, dass die Verfügbarkeitsbereiche von Fahrradverleihern meist räumlich eng eingegrenzt sind und es deswegen als eher unwahrscheinlich betrachtet werden kann, dass sich die A2 eines Fahrradverleihers mit der eines benachbarten Mobilitätsanbieters überlappt.

7.3 Ausgabe von Grafiken für die Streckenvisualisierung

Im Datenmodell von MoBee sind die Positionsdaten aller Haltestellen und Wegpunkte für die dynamische Streckenführung enthalten. Es wäre also durchaus möglich, die Streckenführung grafisch darzustellen (besonders wenn, wie in 6.2.4 vorgeschlagen, die Standorte aller durchfahrenen Haltestellen ermittelt werden). Hierfür müsste von einem Drittanbieter (Open Streetmap oder Google Maps) über einen Webservice Kartenmaterial bezogen werden. Auf diesem könnten dann mit einem offen Grafikpaket (zum Beispiel ImageJ [Nat14]) die Haltestellen/Wegpunkte sowie der Streckenverlauf gezeichnet werden. Die so erstellten Grafiken könnten einem Geschäftskunden dann optional zusammen mit den Routenvorschlägen als Byte Array übermittelt werden. Somit könnte sich der Geschäftskunde eine eigene Implementierung einer Visualisierungsfunktion sparen, was den Mehrwert des MoBee Systems nochmals steigern würde.

Um dieses Feature zu realisieren müsste zuerst einmal der *Generic_LineTrip_Port* (siehe 5.3.1) um eine Operation erweitert werden, welche auf den Aushangfahrplan zugreift und diejenigen Linien zurückgibt, welche den gewählten Knoten durchfahren. Eine solche Operation müsste sowohl die gewählte Haltestelle eindeutig identifizieren als auch ein Format definieren, in welchem die Liste der durchfahrenen Haltestellen zurückgegeben wird.

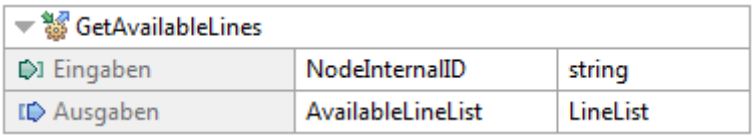


Abbildung 7-3 Mögliche Erweiterung des Generic_LineTrip_Port

In diesem Implementierungsvorschlag wird zur Identifizierung der Haltestellen der Typ String gewählt, da manche der bisher kooperierenden Mobilitätsanbieter Mischungen aus Zahlen und Buchstaben nutzen, um die von ihnen bedienten Haltestellen systemintern zu identifizieren. Die eigentliche Ermittlung der Haltestellenkette findet in diesem Vorschlag über einen eigenen BPEL-Subprozess statt. Dieser erwartet als Eingabeparameter den Routenvektor als Koordinatenpaar, die Referenz auf den Webservice des Mobilitätsanbieters und die Bezeichnung derjenigen Linie, auf deren Durchfahren geprüft werden soll.

Als erster Schritt wird in diesem neuen Subprozess, welchen wir *TraversedNodes_Query* nennen wollen, auf dem übergebenen Routenvektor Checkpoints definiert. Diese Checkpoints sind zueinander äquidistant und werden als Eingabeparameter für die Operation *GetNearNodesByLoc* (siehe 5.3.1.1) genutzt. Als nächstes wird für jeden Checkpoint über die Menge der von *GetNearNodesByLoc* gelieferten Haltestellen iteriert. Hierbei wird zuerst geprüft, ob die im Moment behandelte Haltestelle bereits geprüft wurde. Zu diesem Zwecke besitzt der *TraversedNodes_Query* eine Liste vom Typ String, welcher die systeminternen Identifikationsnummern aller bereits geprüften Haltestellen enthält. Wurde die Haltestelle noch nicht geprüft, wird die Operation *GetAvailableLines* des *Generic_LineTrip_Port* aufgerufen, um alle

Linien zu erhalten, welche die Haltestelle durchfahren. Diese Linien werden nun mit der gesuchten Linie verglichen. Wird diese gefunden, so wird die Haltestelle als *RailNode*-Element (siehe 5.3.1.2) der Liste der durchfahrenen Haltestellen hinzugefügt. Schließlich wird die Identifikationsnummer der Haltestelle der Liste der bereits geprüften Haltestellen hinzugefügt. Als letzter Schritt des Subprozesses wird die Liste der durchfahrenen Haltestellen als Antwort an den Aufrufenden Prozess übergeben. Abbildung 7-4 zeigt die Beispielimplementierung des vorgeschlagenen *TraversedNodes_Query* als BPEL-Flußdiagramm.

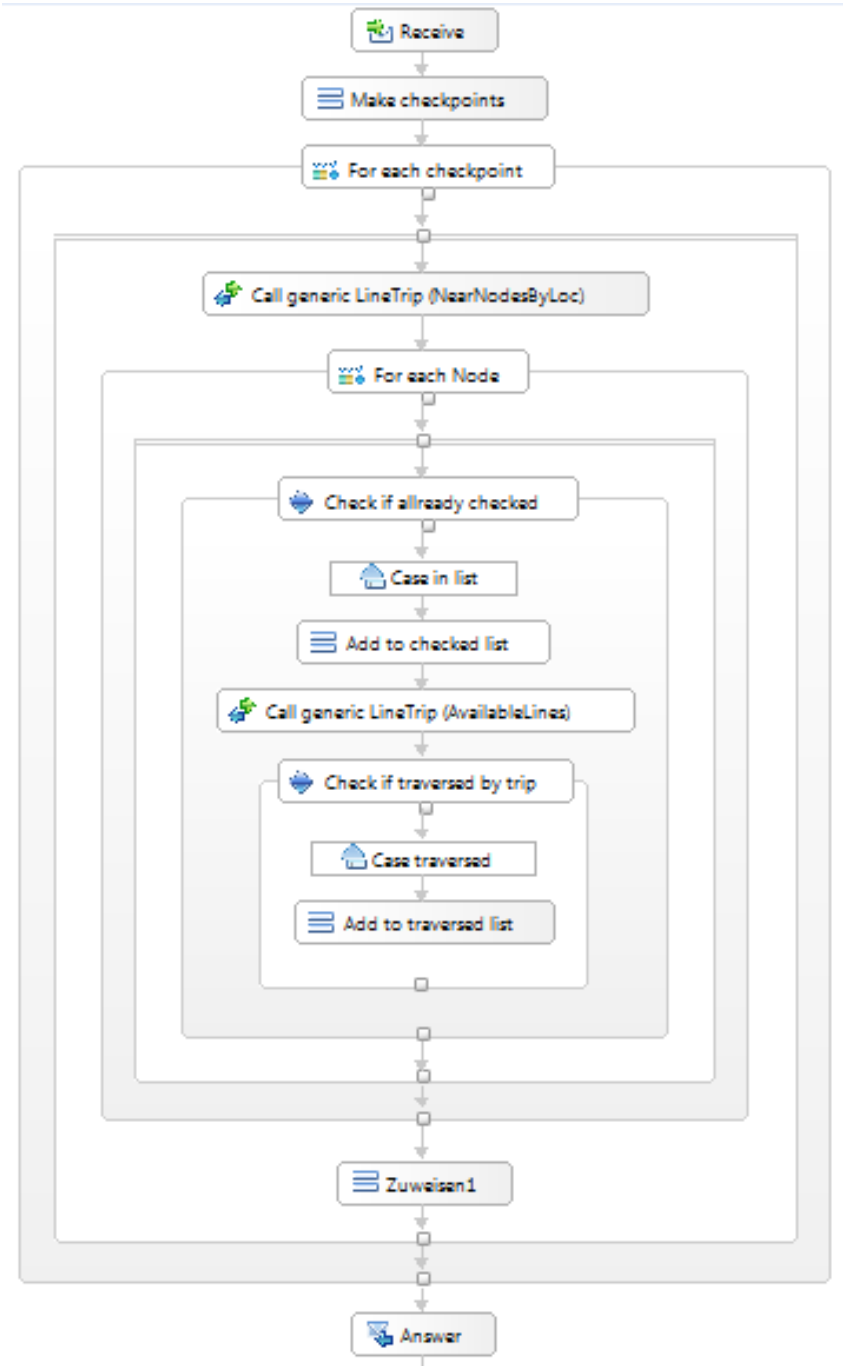


Abbildung 7-4 Designvorschlag für den TraversedNodes_Query

Nachdem nun eine Liste aller durchfahrenen Knoten vorliegt, kann aus diesen eine grafische Repräsentation der Fahrtstrecke erzeugt werden. Zu diesem Zwecke muss zuerst eine Karte generiert werden, welche das Gebiet des Routenvektors abdeckt. Hierfür stehen bereits Webdienste wie zum Beispiel Google Maps [Goo14] oder Open Streetmap [Ope14] zur Verfügung. Beide Dienste besitzen Funktionalität zum Erzeugen statischer Karten als Bilddateien.

Ist nun eine Karte des entsprechenden Gebietes vorhanden, müssen die Haltestellen und Wegpunkte (bei dynamisch gerouteten Mobilitätsanbietern) auf diese übertragen werden. Sind Länge und Breite des Kartenbildes sowie seine Eckpunkte bekannt, so lassen sich die Koordinaten der Haltestellen und Wegpunkte problemlos mittels eines Dreisatzes in Höhen- und Längenangaben in Pixeln auf dem Kartenbild umwandeln. Hierbei ist zu beachten, dass der genutzte Kartendienst nicht unbedingt dieselbe Projektion benutzt wie das MoBee System. Open Streetmap benutzt zwar die EPSG:4326-Projektion, welche dieselben Koordinaten benutzt wie das WGS84-System. Das weitaus verbreitetere Google Maps verwendet jedoch die Mercator-Projektion, welche in manchen Regionen stark von WGS84 abweicht. Für das eigentliche Hinzufügen der Grafiken, welche die Haltestellen und Wegpunkte repräsentieren, bietet sich ein offenes Grafikpaket wie das zuvor erwähnte ImageJ an. Das pixelgenaue Hinzufügen einer Grafik zu einer bereits bestehenden ist eine Standardfunktion und dürfte von allen Paketen beherrscht werden. Die in diesem Absatz beschriebene Funktionalität wird in diesem Implementierungsvorschlag als zusätzlicher Webservice implementiert, welcher an den *TripQuery_Wrapper*-Subprozess angeschlossen wird. Abbildung 7-5 zeigt die Schnittstelle dieser Webservice inklusive der für sie notwendigen zusätzlichem XML-Elemente. Alternativ könnte die Funktionalität zur Bildbearbeitung auch in den bereits bestehenden *SortingService* integriert werden. Dieser sollte dann jedoch umbenannt werden, zum Beispiel in *UtilityService*.

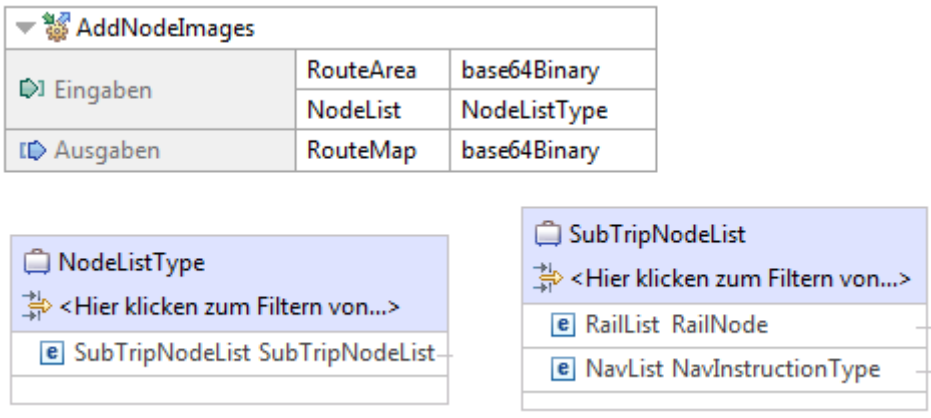


Abbildung 7-5 Schnittstelle und Datentypen für den Bildbearbeitungsservice

Die Schnittstelle des neuen *ImagingService* beinhaltet nur die Operation *AddNodeImages*. Dieser wird, zusammen mit der ursprünglichen Gebietsgrafik, ein Element vom Typ *NodeListType* übergeben. Dieses Element enthält eine Liste von Elementen des Typs *SubTripNo-*

deList, welche jeweils eine Teilstrecke des abzubildenden Routenvorschlages darstellen. Das *SubTripNodeList*-Element beinhaltet entweder eine Liste von Elementen des Typs *RailNode* oder des Typs *NavInstructionType*. Diese Elemente stellen im MoBee Datenschema (siehe 5.3) die einzelnen Wegpunkte/Haltestellen der Teilstrecken dar und enthalten jeweils deren WGS84-Koordinaten auf dessen Basis sich die Wegpunkte und Haltestellem auf der Gebietsgrafik verzeichnen lassen.

Das so erzeugte Bild muss via XML austauschbar sein, sowohl um die Übergabe des Bildes zwischen den einzelnen Subprozessen zu ermöglichen, als auch um es letztendlich dem Geschäftskunden des MoBee Systems zusammen mit den Routenvorschlägen ausliefern zu können. Zu diesem Zwecke wird in diesem Implementierungsvorschlag der XML-Typ *base64Binary* verwendet. Hierbei handelt es sich wie der Name schon vermuten lässt um die Darstellung eines Binärwertes zur Basis 64. Um das vom Grafikpaket erstellte Bild in eine Binärrepräsentation zur Basis 64 umzuwandeln, bietet sich das freie Framework Commons Codec [Apa14] der Apache Software Foundation an. Dieses enthält die Java-Konvertierungsklasse *Base64*, welche Binärdaten in deren Darstellung zur Basis 64 umwandelt.

7.4 Zusammenfassung

Das Ziel dieser Arbeit, einen echten multimodalen Verkehrsplanungsdienst zu entwickeln, welcher diese Funktionalität durch Webserviceaggregation umsetzt (vgl. 1.1), wurde erreicht. Wie in 6.1 gezeigt wurde, sind das MoBee Konzept und der ihm zugrunde liegende TiGeR-Algorithmus (vgl. 4.2) real umsetzbar und liefern die gewünschten Ergebnisse. Die Unzulänglichkeiten bestehender Lösungen für die multimodale Routenplanung, welche in 3.4 erörtert wurden, werden durch das MoBee Konzept behoben. Allerdings ist es fraglich, ob die hier beschriebene Implementierung des Konzeptes tatsächlich praxistauglich ist: Zum einen besteht das Problem der Abfragedauer (siehe 6.2.2). Die Frage, ob diese Problematik durch die im selben Kapitel vorgeschlagenen Verbesserungsmöglichkeiten abgemildert werden kann, müsste durch weitere Arbeit zum Thema geklärt werden. Zum anderen bedarf der TiGeR-Algorithmus selbst noch einiger weiterer Arbeit: Gerade das Problem, sinnvolle Handoff Querys zu erzeugen (siehe 6.2.1), wird gelöst werden müssen, bevor das MoBee Konzept in der Praxis eingesetzt werden kann.

Alles in allem kann jedoch gesagt werden, dass in dieser Arbeit ein vielversprechendes, neuartiges Konzept für die multimodale Routenplanung entwickelt wurde, welches bestehenden Lösungen in vielerlei Hinsicht überlegen ist. Bis MoBee der Sprung auf den Markt gelingt und die Basis für eine neue Form der Mobilitätsplanung geschaffen wird, wird allerdings erst weitere Arbeit zum Thema nötig sein.

Literaturverzeichnis

[Ant12] Antsfeld, Leonid und Walsh, Toby. 2012. Finding Optimal Paths in Multi-modal Public Transportation Networks using Hub Nodes and TRANSIT algorithm. *Artificial Intelligence and Logistics.* 2012, Bd. 7.

[Apa141] Apache Software Foundation, The. About Apache ODE. *ode.apache.org/.* [Online] [Zitat vom: 21. April 2014.] http://ode.apache.org/.

[Apa14] —. Apache Commons Codec. *http://commons.apache.org.* [Online] [Zitat vom: 20. April 2014.] http://commons.apache.org/proper/commons-codec/.

[Bau09] Bauer, R. und Delling., D. 2009. SHARC: Fast and robust unidirectional routing. *Journal of Experimental Algorithmics.* 2009, Bd. 14.

[Bro04] Brodal, Gerth Stølting. 2004. Time-dependent Networks as Models to Achieve Fast Exact Time-table Queries. *Electronic Notes in Theoretical Computer Science.* 2004, Bd. 92, S. 3–15.

[Dai14] Daimler AG. 2014. moovel. Mein A nach B. *moovel.com.* [Online] 2014. [Zitat vom: 10. April 2014.] www.moovel.com.

[Dij59] Dijkstra, Edsger W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik.* 1959, Bd. 1, 1, S. 269-271.

[Gil88] Gilles Brassard, Paul Bratley. 1988. *Algorithmics: theory and practice.* s.l. : Prentice Hall, 1988. S. 87-92.

[Gol05] Goldberg, A. V. und Harrelson, C. 2005. Computing the shortest path: A* search meets graph theory. *Proceedings of the 16th annual ACMSIAM symposium on Discrete algorithms.* 2005, S. 156-165.

[Goo12] Google. 2012. General Transit Feed Specification Reference. *developers.google.com.* [Online] 12.. Juni 2012. [Zitat vom: 10. April 2014.] developers.google.com/transit/gtfs/reference.

[Goo14] Google Inc. Static Maps API V2 Developer Guide. *developers.google.com.* [Online] [Zitat vom: 20. April 2014.] https://developers.google.com/maps/documentation/staticmaps/.

[HAF14] HAFAS - Die perfekte Verbindung zum Kunden. *hacon.de.* [Online] [Zitat vom: 16. April 2014.] http://www.hacon.de/hafas.

[Har68] Hart, Peter E., Nilsson, Nils J. und Raphael, Bertram. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics.* 1968, Bd. 4, 2.

[Hor02] Horn, M. 2002. Multi-modal and demand-responsive passenger transport systems: a modelling framework with embedded control systems. *Transportation Research Part A: Policy and Practice.* 2002, Bd. 36, 2, S. 167–188.

[Hrn13] Hrncir, J und Jakob, M. 2013. Generalised time-dependent graphs for fully multimodal journey planning. *Intelligent Transportation Systems - (ITSC), 2013 16th International IEEE Conference on.* 2013.

[IBM141] IBM. DB2 Database Software. *ibm.com.* [Online] [Zitat vom: 16. April 2014.] http://www-01.ibm.com/software/data/db2/.

[IBM142] —. DB2 Spatial Extender for Linux, UNIX and Windows. *ibm.com.* [Online] [Zitat vom: 16. April 2014.] http://www-03.ibm.com/software/products/en/db2spaext.

[IBM14] —. IBM - Prozessautomatisierung - WebSphere Process Server. *ibm.com.* [Online] [Zitat vom: 16. April 2014.] http://www-03.ibm.com/software/products/de/wps.

[Kar11] Karande, Aarti, Karande, Milind und Meshram, B. B. 2011. Choreography and Orchestration using Business Process Execution Language for SOA with Web Services. *International Journal of Computer Science Issues.* 2011, Bd. 8, 2, S. 224.

[Kha03] Khalaf, Rania und Leymann, Frank. 2003. On Web Services Aggregation. *Lecture Notes in Computer Science.* 2003, Bd. 2819, S. 1-13.

[Kot10] Kotler, Philip und Armstrong, Gary. 2010. *Principles of Marketing.* s.l. : Pearson Education, 2010.

[Kry04] Krygsmana, Stephan, Dijst, Martin und Arentze, Theo. 2004. Multimodal public transport: an analysis of travel time elements and the interconnectivity ratio. *Transport Policy.* 2004, Bd. 11, 3, S. 265–275.

[KuR06] Kumar, V., Reinartz, Werner. 2006. *Customer Relationship Management.* s.l. : Springer, 2006. S. 261.

[Kum06] Kummer, Sebastian. 2006. *Einführung in die Verkehrswirtschaft.* s.l. : UTB Verlag, 2006.

[Lor12] Matzat, Lorenz. 2012. Verpasste Open Data-Chance: Deutsche Bahn schenkt einzig Google seine Fahrplandaten. *Netzpolitik.org.* [Online] 17. September 2012. [Zitat vom: 20. April 2014.] https://netzpolitik.org/2012/verpasste-open-data-chance-deutsche-bahn-schenkt-einzig-google-seine-fahrplandaten/.

[Nat84] National Imagery and Mapping Agency. 1984. *Technical Report 8350.2.* s.l. : National Imagery and Mapping Agency, 1984.

[Nat14] National Institute of Mental Health. ImageJ - Image Processing and Analysis in Java. *http://imagej.nih.gov/.* [Online] [Zitat vom: 16. April 2014.] http://imagej.nih.gov/ij/.

[New] Newman, Peter und Kenworthy, Jeffrey. 1999. *Sustainability and Cities: Overcoming Automobile Dependence.* s.l. : Island Press, 1999.

[OAS07] OASIS. 2007. Web Services Business Process Execution Language Version 2.0. *oasis-open.org.* [Online] 11. April 2007. [Zitat vom: 10. April 2014.] docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[Ope14] Open StreetMap. Static map images. *http://wiki.openstreetmap.org.* [Online] [Zitat vom: 20. April 2014.] http://wiki.openstreetmap.org/wiki/StaticMap.

[OSG14] OSGeo. About PostGIS. *postgis.net.* [Online] [Zitat vom: 22. April 2014.] http://postgis.net/.

[Fra00] Schulz, Frank, Wagner, Dorothea und Weihe, Karsten. 2000. Dijkstra's algorithm on-line: an empirical case study from public railroad transport. *Journal of Experimental Algorithmics.* 2000, Bd. 5.

[Sie14] Siemens AG. Green eMotion Project. *greenemotion-project.eu.* [Online] [Zitat vom: 16. April 2014.] http://www.greenemotion-project.eu/.

[Suj14] Suji, K. Adlin und Sujatha, S. 2014. A Comprehensive Survey of Web Service Choreography, Orchestration and Workflow Building. *International Journal of Computer Applications.* 2014, Bd. 88, 13.

[W3C01] World Wide Web Consortium. 2001. Web Services Description Language (WSDL) 1.1. *w3.org.* [Online] 15. März 2001. [Zitat vom: 10. April 2014.] http://www.w3.org/TR/wsdl.

[W3C04] —. 2004. Web Services Glossary. *w3.org.* [Online] 11. Februar 2004. [Zitat vom: 10. April 2014.] www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice.

[YuH10] Yu, Haicong und Lu, Feng. 2010. A Multi-Modal Route Planning Approach With an Improved Genetic Algorithm. *Joint International Conference on Theory, Data Handling and Modelling in GeoSpatial Information Science.* Mai, 2010, S. 343-348.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Stuttgart, den 02. Mai 2014 _____