HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

COMPUTER SCIENCE

INSTITUT
FÜR
SOFTWARE

Computing Isochrones in Multi-Modal, Schedule-Based Public Transport Networks

Student Research Project Thesis

Department of Computer Science
University of Applied Science Rapperswil

Spring Term 2011

Author(s):	Marco Birchler
	Kevin Lynn
Advisor:	Prof. Stefan Keller
Project Partner:	-
External Co-Examiner:	Claude Eisenhut, Eisenhut Informatik AG Burgdorf
Internal Co-Examiner:	-

Datum Erstellung: 21.02.2011
Letzte Aktualisierung: 03.06.2011

Impressum

Auflagenzahl: 3
Jahr: 2011
Herausgeber: Birchler, Marco; Lynn, Kevin
http://cipt.ecomb.ch

Abstract

Modern decision support systems in the area of public transport networks often amount to reachability problems, where time tables and itineraries are influential factors. Isochrones are a recommended solution to these questions.

Isochrones are defined as the set of all points reachable from a specific point of interest given an equal time span. This paper describes the realization of a service oriented architecture component (i.e. service) that computes isochrones in multi-modal, schedule-based public transport networks.

The web service supports the well-known interface standards Web Feature Service (WFS) and Web Processing Service (WPS) by the Open Geospatial Consortium, Inc. (OGC), an international standards organization that is leading the development of standards for geospatial information.

The presented algorithm computes contemporary and exact isochrones based on the underlying time table data of the official Swiss public transport network, covering the 2010 / 2011 period. Given an arbitrary starting coordinate, a departure time and the latest possible arrival time, the service computes all potential shortest routes, such that a combination of walking and public transport are within the time limit.

The resulting isochrones can be demonstrated using a visualization component, like Quantum GIS or a web browser that supports the Google Maps API.

The web service has been successfully tested with Quantum GIS.

Further development includes integration into a website for public use, performance improvement in the area of algorithm design, parallelisation, caching and the use of a graph database.

For more information, please visit our website at http://cipt.ecomb.ch.

Management Summary

Initial Position

The web provides us with many different kinds of services. Nowadays, location and links to real places are conventional services expected by web users. Location and maps answer the question of "where", e.g. "where am I" or "where is it". The immediate and obvious follow-up question is "how do I get there", which is solved by diverse routing services, e.g. turn-by-turn navigation. This is the direction part.
The service we present in this project answers the question of reachability, e.g. "what is in my reach" or "wherever can I be within one hour".
This sort of question can emerge in offerings that rely heavily on the web, such as real estate or job portals. A customer may want to know all residences for possible rental, from where the office is reachable within a certain time limit. Another example could be a retail store choosing a location which minimizes traveling distance in order to increase their zone of attraction. Isochrones are a solution to these questions.

Isochrones are defined as the set of all points reachable from a specific point of interest given an equal time span.

The goal of this project is to develop a service oriented architecture component that computes isochrones in multi-modal, schedule-based public transport networks. The service must support the well-known interface standards Web Feature Service (WFS) and Web Processing Service (WPS) by the Open Geospatial Consortium, Inc. (OGC).

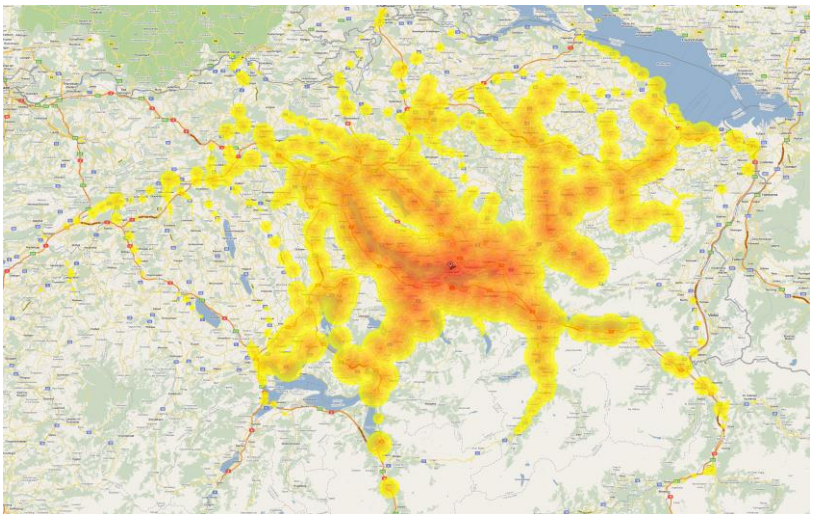


Figure 1 Isochrones in Google Maps

The underlying algorithm should compute contemporary and exact isochrones, where absolute performance and response time is not the primary issue. The service should use real-world time table data from the Swiss public transport network.

Other applications found that use isochrone calculation are urban planning systems, e.g. at the Municipality of Bolzano-Bozen (1) where the application calculates isochrones for point of interest zone of attraction using multi-modal schedule-based transport networks, and at the University of Applied Sciences Western Switzerland (2) where the goal was to find bakery stores within reach and to provide directions.

The service was developed over the course of one semester at the HSR University of Applied Sciences using an iterative approach. After establishing the infrastructure, the

first step was to import the time table data into the database. The format of the data is known as the HAFAS raw data format (3), a standard data format for public transport systems. In parallel, the services' WPS interface was developed.
In a second step, a graph representation for the transport network and a suitable algorithm were evaluated.
The implementation of an adequate algorithm was the final step. As an additional requirement, found because testing the service over WPS with our client application Quantum GIS was not possible, we implemented a WFS interface.
Identified risks involve the possibility of not finding a suitable algorithm with the ability to calculate isochrones within a contemporary time limit. A second risk would be not discovering a suitable library for graph representation and WPS service.

Findings

As a result of our work, a suitable algorithm was found that is able to compute isochrones in a contemporary time limit. The algorithm can answer 80% of all requests in fewer than five seconds and is based on the well-known Dijkstra shortest path algorithm (4).

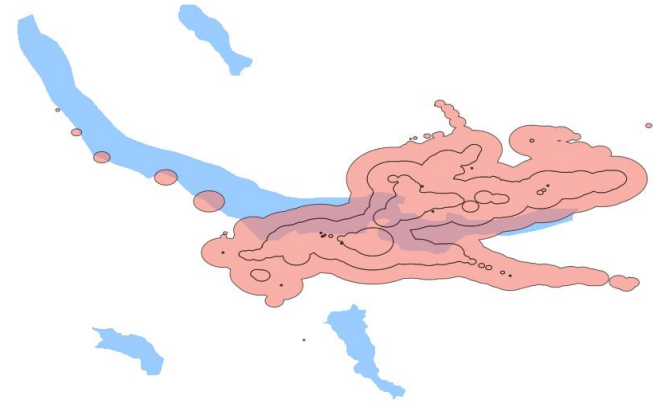


Figure 2 Isochrones in Quantum GIS

A user of the web service is able to choose an arbitrary start coordinate on the map, a start and end time which limits the search scope, and the number of isochrones, which divides the chosen time span into evenly distributed intervals.

After the algorithm has found all reachable line sections on the public transportation network, for each station the remaining walking time is shown as a circle around the station, which is a wanted approximation.

The service provides interfaces for both WPS and WFS. Figure 2 displays a WFS request in Quantum GIS. As an addition, the service provides an output that can be presented in Google Maps, shown in Figure 1.
The time table source data is in the HAFAS raw data format and the database stores the complete time table of the official Swiss public transport network, valid for the 2010 / 2011 time period.

Though the web service is built as a private feasibility study, to our knowledge, such a service is available nowhere on the web for the general public. Our work presents the fundamentals for such a service.

All primary targets set for our work could be achieved.

Future Prospects

Future improvements could be done in several areas:

Performance improvements:

Further work in algorithm design could yield a better approach to solving isochrone calculation. As the service is a feasibility study and only meant for private use, no time was invested for parallelisation, e.g. a client request currently does not run on its own thread. Another performance improvement could be pre-processing the results of all possible requests, which is possible because the time table is static for a time period. A caching approach could also yield performance gains.

Graph Database

Time table data currently resides in a relational database management system. An alternative approach to persist the graph directly could be achieved by using a graph database.

Join with Other Data Sources

The underlying time table data and diverse other data source could be included as a filter into algorithm processing, allowing the user to query the service in more customized ways with enhanced results.

Remaining Walking Time

As described above, at each station the remaining walking time is approximated with a circle around the station. The algorithm could be used in combination with a street network, like OpenStreetMaps (5), to display more detailed isochrones.

WFS/WPS Server

The OGC interface standard defines some additional features, e.g. WDSL, which could be implemented in a future release.

Table of Contents

List of Figures

List of Tables

Assignment of Tasks

Computing Isochrones in Multi-Modal, Schedule-Based Public Transport Networks

Studienarbeit von Marco Birchler und Kevin Lynn

Abteilung Informatik, Frühjahrssemester 2011

Ausgangslage

Das Ziel dieser Studienarbeit ist es, eine Applikation zu erstellen, welche die Grundlage für Heatmaps (sog. Isochronenkarten von Reisezeiten) mit dem öffentlichen Verkehr (ÖV) bildet.

Solche Karten könnten eingesetzt werden um beispielsweise künftig Fragen wie die folgenden zu beantworten:

- Auf einem Job- oder Immobilien-Portal: "Wenn man eine Immobilie an der Adresse XY erwerben würde, wo gibt es Arbeitsgeber, die man mit dem ÖV innerhalb einer Stunde erreicht?" oder: "Ich arbeite in der Firma X (am Standort XY), wo kann man Immobilien mieten, um bereits um 5:00 Uhr mit dem ÖV am Arbeitsplatz zu sein?".
- Auf Nebelkarte: Ausgehend von einer Position (mit Mausklick auf Webkarte): „Wo kann ich mit dem ÖV Freizeit-Destinationen erreichen, die nebelfrei sind?".

Wichtig ist, dass man zur Berechnung der Heatmap einen beliebigen Standort, bzw. ÖV-Haltestelle angeben kann. Es sollte auch möglich sein, beliebige Koordinaten oder Adressen als Startposition anzugeben. Die Daten stammen vorzugsweise von der SBB, ansonsten von einem anderen Industriepartner (z.B. ZVV). Das Kartenmaterial wird z.B. über OpenStreetMap bezogen. Falls eine zeitnahe Generierung der Isochronen nicht genügend performant gemacht werden kann, sollten nützliche Caching-Mechanismen entworfen werden.

Aufgabenstellung

1) Es sollten zuerst Algorithmen zur Berechnung der Isochronen evaluiert werden. Darauf aufbauend soll ein Algorithmus ausgewählt, ggf. verbessert und umgesetzt werden.
2) Die Applikation soll einen Webservice (sog. WPS) anbieten (mit einem passenden Format, z.B. GeoJSON oder GML sowie ggf. ein geläufiges Geo-Rasterformat).
3) Die Resultate der Arbeit sollen auf einer eigenen Webseite präsentiert werden.

Projektabwicklung

Termine

Siehe Termine: https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html (intern)
Gliedern Sie Ihre Arbeit in Teilschritte. Schliessen Sie jeden Teilschritt mit einem Meilenstein ab. Definieren Sie für jeden Meilenstein, welche Resultate dann vorliegen müssen!

Randbedingungen

Die Arbeitsweise ist iterativ, agil und test-basiert.

- Hardware:
  - Server: Standard-Desktop-PC, virtuelle Server
  - Clients: zu definieren.
- Software:
  - C# ggf. mit WCF.
  - PostgreSQL mit PostGIS.
  - WPS Service wird mit Quantum GIS sowie mit ad-hoc Tools getestet.

Inhalt der Dokumentation

- Die Projektdokumentation (Prosa) ist deutsch, alles andere ist englisch.
- Die fertige Arbeit muss folgende Inhalte haben:
  1. Abstract, Management Summary, Aufgabenstellung
  2. Technischer Bericht
  3. Projektdokumentation
  4. Anhänge (Literaturverzeichnis, CD-Inhalt)
- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind.
- Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen.
- Projekttagebuch, Dokumentation des Projektverlaufes, Planung etc.
- Weitere Dokumente (z.B. Kurzbeschreibung, Poster) gemäss www.hsr.ch und gemäss Absprache mit dem Betreuer.

Form der Dokumentation

- Bericht (Struktur gemäss Beschreibung) gebunden (2 Exemplare) und in Ordner (1 Exemplar „kopierfähig“ in losen, gelochten Blättern).
- Alle Dokumente und Quellen der erstellten Software auf CD; CD's sauber angeschrieben (3 Ex.).

Bewertungsschema

Es gelten die üblichen Regelungen zum Ablauf und zur Bewertung der BA-Arbeit des Studiengangs Informatik der HSR mit besonderem Gewicht auf moderne Softwareentwicklung.

Rapperswil, 27. Oktober 2010, S. Keller

Part 1: Technical Report

1 Introduction

1.1 Issue, Vision

The web provides us with numerous services with links to real existing places. Real estate or job portals are a matter of common knowledge. Each of these services has an unmanageable amount of objects in their underlying database. Therefore they provide their visitors with some filtering functionality.
In Switzerland it is common to filter objects by Canton or by ZIP code. Some services allow their customers to search for an object that lies within a certain beeline distance. Because customers of a job portal are not looking for a job at the other side of the country, this is a nice feature for imprecise filtering. Nevertheless, as soon as there are lakes or mountains in the countryside the filter will return a lot of false results. The only possibility to avoid such results is to increase the limits of the filter (i.e. reducing the beeline distance). Unfortunately this approach will also eliminate some true positive results.

A solution to this kind of problem is to replace the beeline filter with a filter that accounts for the topology, the road or the public transport network. A system that uses the road network and could be the basis of a more intelligent filter can be found in (2).

Our vision is to realise the basis for a new filter that makes use of the public transport network. This filter is based on the calculation of isochrones. Isochrones are defined as the set of all points reachable from a specific point of interest given an equal time span.

The system should be consumable from a real estate or job portal website. Visitors of these websites should be able to view at a glance all kinds of objects that could be reached within a certain travel time using public transport.

1.2 Targets and Sub Targets

The target of this project is to develop a service oriented architecture component that calculates isochrones for public transportation networks.
A user should be able to choose an arbitrary coordinate on a map, a time interval for limiting the search scope and the number of isochrones requested.
Primary targets for this project are:

1. Evaluate and implement an algorithm that solves the isochrones problem and develop an OGC compliant WPS and WFS web service for consumption in (4).
2. Preferably, the algorithm should use real world time table data as a basis for computation.
3. The results of this project should be available on a website.

1.3 Basic Parameters, Environment, Definitions, Limitations

1.3.1 Technology

The application and algorithm are developed completely in C# using the .NET Framework 4. The database is PostgreSQL with PostGIS installed. The web service is hosted on an IIS running ASP.NET.
The server component is hosted on a Virtual Machine provided by the HSR University of Applied Sciences.

1.3.2 Time table data

The time table data was provided by the Swiss Federal Railways and the Zurich Transport Network. The data format is known as HAFAS raw data format, see 2.2.2. The version is 5.20.34.

1.3.3 Web Service

The service interface is standard conform to the official WPS and WFS documentation by the OGC. Both WPS and WFS are version 1.0.0.

1.3.4 Limitations

1.3.4.1 Performance

Absolute performance of the presented algorithm is not the primary concern of this paper. As such, this paper does not seek to evaluate, compare and choose a pareto optimal solution algorithm for computing isochrones. The service is meant to be a feasibility study.

1.3.4.2 Static Data

The data provided by the HAFAS raw data format is static and valid only for the 2010 / 2011 period. No effort has been made to provide an synchronisation or real-time query mechanism onto official (web-) data sources.

1.3.4.3 Transfers between Connections

No transfers between connections are explicitly modelled.

1.3.4.4 Caching

The web service does not use any kind of caching mechanism for the results of the algorithm. The possibility of freely choosing the start time and the start point on a map results in a huge range of possible input parameter values for the algorithm. Therefore the cache would need a lot of storage capacity compared to the minimal number of cache hits. Performance gains by adding a cache would not be significant.

1.3.4.5 Modelling of Time table Data

The time table data from SBB and ZVV actually contains exact information about valid dates of a single public transport connection. For example some trains are only in service during the week. Currently, the web service does not interpret this information. Because of this it is not possible to enter a date for the query or another limitation based on the date (e.g. weekend). The choice of data includes only the subset of the available information in the HAFAS raw data format necessary to solve isochrone computation.

1.3.4.6 Mileage / Walking

The remaining time at a station for walking after exiting a public transport medium is modelled as a circular area and is only an approximation. No street network is modelled.

1.3.4.7 Coordinate System

WPS and WFS accept the starting point in WGS and CH1903 coordinates; internally the system bases the calculation only on the CH1903 system. As the usage of this coordinate system is limited to Switzerland, the service would probably not return correct data for countries other than Switzerland.

1.4 Procedure, Setup of Task Project

The work presented in this paper was performed within the scope of a Student Research Project thesis at the HSR University of Applied Sciences.
The project took place over a one semester time period of 14 weeks.
Project methodology is iterative, based on an adapted Scrum template.
For further details of project progress please refer to Part 2.

1.4.1 Project Phases and Iterations

Project time was divided into five iterations and presented in the following.

1.4.1.1 Planning and Infrastructure Iteration

The first Iteration in the project was used to establish a setup of infrastructure software needed for further development work.

1.4.1.2 Architecture Iteration

The second iteration was used to evaluate a suitable graph library, to import the time table data into our database and to implement the WPS Service.

1.4.1.3 Algorithm Iteration

The third iteration was used to evaluate different algorithms to solve the isochrone calculation problem.

1.4.1.4 Implementation Iteration

The fourth iteration was used to implement a suitable algorithm found in the previous iteration.

1.4.1.5 Documentation Iteration

The final iteration was used to produce all paper deliverables.

1.4.2 Risks

Identified risks involve the possibility of not finding a suitable algorithm with the ability to calculate isochrones within a contemporary time limit. A second risk would be not discovering a suitable library for graph representation and WPS service. See chapter 13.4.

1.4.3 Overview of the remaining parts

The remaining parts of this paper are structured as following:

In Chapter 2, we provide an overview of the state of art to date for applications of isochrones and time table systems in public transport. We also present existing standards and show shortfalls of the current approaches.

Chapter 3 presents our criteria for finding a solution in combination with our found conclusions.

In Chapter 4, we detail our realization concept.

Finally, in Chapter 5 we present our results and give an outlook for possible further development.

2 State of the Art to Date

2.1 Existing Approaches

2.1.1 Isochrones

2.1.1.1 Municipality of Bolzano-Bozen

The Municipality of Bolzano-Bozen uses the calculation of isochrones as a planning instrument (1). Given a point of interest (POI), an isochrone is defined as the set of all points from which that POI can be reached in a specific time span. As such, this definition of isochrones is an opposite approach compared to isochrones used in our system.
The planning tool is implemented in Java on top of an Oracle Spatial Network Model.
The presented algorithm is multi-modal, considering bus networks and street networks including transfers, and is a variation of the classical breath-first search on graphs.

2.1.1.2 Broetlikrones

A play on words on bread and isochrones, the application, developed by the University of Applied Sciences Western Switzerland during a workshop, is used to display bakery locations within an isochrone around a user-defined point in Switzerland. Users can get the shortest path to one of the bakeries (2).
The application is implemented using pgRouting, a routing tool used in PostGIS.
No information about the used algorithm is available.

2.1.2 Time table Modelling and Shortest Paths

Time table information systems in public transport systems lend themselves to be modelled as a graph. Current approaches to modelling time table information systems are time-dependent and time-expanded (5).
In the time-expanded approach, every event at a station, e.g. the arrival and departure of a train, is modelled as a node in a graph.
In the time-dependent approach however, the graph contains only one node per station.
Experimental comparisons suggest the time-expanded approach to be more robust for modelling more complex scenarios, whereas the time-dependent approach shows better performance.

The classic problem in route finding is the earliest arrival problem, where an optimal itinerary must be found for a route with starting time A and arriving no later than arrival time B. Current research suggests a variation of the Dijkstra shortest-path algorithm as a solution to this problem.

A combination of a time-expanded time table model and a variation of the shortest-path problem can be found in (6).

2.2 Standards

2.2.1 OGC Web Services

The Open Geospatial Consortium (7) develops publicly available interface standards. OGC standards support interoperable solutions that “geo-enable” the web. The standards are technical documents that detail interfaces and encodings.

2.2.2 HAFAS

The HAFAS raw data file format (3) is a standard representation of public transport time table information developed by HaCon Ingenieur GmbH (8) in Germany. Several European railway companies, e.g. Deutsche Bahn AG, Austrian Federal Railway and the Swiss Federal Railway represent their time table in the HAFAS raw data format.
Time table information is stored in several text files. In order to be used in a database, the data has to be imported first.

2.3 Shortfalls

2.3.1 Municipality of Bolzano-Bozen

The presented algorithm is based on a variation of a breadth-first search. This solution is found to not solve the isochrone calculation problem in public transport networks.

As soon as a node, in this case a station, is explored for the first time, it is marked as visited. If another path is found that comes across the same station previously visited, the exploration stops here.

Taking traveling time into account, it may be the case that the second encounter of the same station was on a route with a faster connection, meaning the remaining rest time is higher as compared to the first encounter and thus, using the faster connection means having the possibility to travel further on the network.

A breadth-first search algorithm fails to notice this distinction between different encounters and as a consequence fails to calculate correct isochrones.

An adaption of a Dijkstra shortest-path algorithm thus was evaluated.

2.3.2 Time-expanded for more Detailed Modelling

The time-dependent approach is considered to have better performance in simple modelling.

2.3.3 Broetlikrones

No algorithm could be found and thus no statement can be made.

3 Valuations

3.1 Criteria

3.1.1 Run-time Performance

3.1.1.1 Theoretical

The algorithm should have a worst case running time of O(N log N).

3.1.1.2 Practical

In general the response time of the service is dependent on maximal travel time and the number of requested isochrones. The following table lists the maximum response time for different combinations of travel time and number of isochrones. In order to exclude the connection speed from the measurement, the measure tool should be run directly on the webserver (localhost). As the measured result also depends on the start point, a measuring with equal parameters should be done several times for different start points.

	Max. travel time		
	30 min	60 min	120 min
1 Isochrone	< 1s	< 1s	< 2s
10 Isochrones	< 1s	< 2s	< 5s
60 Isochrones	< 5s	< 8s	< 20s

Table 1 Admissible response time for isochrone calculation

3.1.2 Correctness

The algorithm must produce correct and valid isochrone calculation results.

3.1.3 Standard Compliance WPS / WFS Web Service

The web service request and replay format must be standard compliant with official OGC standards.

3.1.4 Contentedness

The algorithm should produce isochrones to our satisfaction. This is a subjective measure.

3.2 Results and Conclusion

3.2.1 Theoretical

All data structures used are hash tables, for which a search can run in O(log N). The algorithm itself, which is a Dijkstra, has a worst case running time of O(N log N) (9). Only the search for all stations within a certain time frame, see 4.1.4.1, is O(N). Overall the algorithm performs in O(N log N).

3.2.2 Practical

For each combination of isochrone number and maximum travel time we did 1000 individual tests with a testing client that we ran directly on the webserver. The coordinates for the start point are randomly chosen from a bounding box over Switzerland (WGS 46, 7 – 47.62, 10). The raw data we used for calculating the means in the table below can be found on the CD (90_Performance_Messung/Messresultate).

Measure-Matrix	Max. travel time		
	30 min	60 min	120 min
1 Isochrone	0.01s	0.10s	2.30s
10 Isochrones	0.04s	0.33s	5.74s
60 Isochrones	0.28s	1.13s	23.29s

Table 2 Mean time required for isochrone calculation

The times we have measured show a big distribution. This comes from the fact that different areas in Switzerland have a varying density of stations. As soon as a start point near a larger city is chosen, the resulting possible travel tree increases heavily. This is especially the case in the greater Zurich area. Due to the additional ZVV data we store in our database for this region, station coverage there is much higher. Picture 1 in Appendix A shows this fact. The results verify that queries with 30 and 60 maximum travel time do fulfil our time boundary criteria from paragraph 3.1.1.2 whereas the criteria for a 120 minute travel cannot be maintained.

4 Realisation Concept

4.1.1 Time-table Import

The time table information we received from SBB and ZVV was in the HAFAS raw data format. In this standard the information is stored in several text files and therefore first has to be imported to the database. For this purpose we created a small application "Fahrplan2PostgreSql". After the start of the program, a database connection and a text file can be chosen and the import process can be started. For all SBB data this process will take around an hour. Currently we are using just two text files.

HAFAS Standard	SBB terminology	ZVV terminology	Table name in database
FPLAN	ZUGDAT	L00XXX (X standing for an integer value)	fplan
BFKOORD	KOORD	KOORD_WGS84	bfkoord

Table 3 Different glossary for text files used by SBB, ZVV and in CIPT database

The data in fplan is not completely in the third normal form (3NF). The same combination of fk_bahnhof and fk_nextbahnhof is appears very often in the table because usually more than one train or bus connects the same stations per day. We do not perform this normalization because it would just lead to additional work for the import and later also for loading of the data into memory while the application is starting. The application accesses the data in the database with the help of a view (zugdat_view). This view is actually the fplan table but with some additional columns providing detailed information for the stations from the bfkoord table.
If the third normal form for the data is required, a SQL script which does exactly that can be found on the CD (80_Daten-Aufbereitung/SQL-Skripte /3. Normalform).

4.1.2 Representation of Time Table as Time-expanded Graph

The decision to model the time table as a time-expanded graph was done primarily for performance reasons, as 2.1.2 describes.
Filling the graph is done only once at application start-up and all stations and train line sections are held in memory for as long as the web service is running. As such, service of requests does not query the database.
The graph structure is an adjacency-list, where each station ID holds a list of line sections that connect this station.
Loading can be done efficiently since the graph structure is mirrored in the database schema, as we decided to de-normalise the database design, see 0.

4.1.3 Station Search

Each call to the CIPT web service that requires the calculation of isochrones has to provide a coordinate within Switzerland. This coordinate represents the starting point of the algorithm and can be chosen freely. But the modified Dijkstra algorithm, which calculates a route through the public transport network, needs a set of stations as input. Therefore, before this algorithm can be started, the set of stations has to be created depending on the start coordinate. This set should contain all stations reachable by foot within the maximum travel time. The arrival time of the stations in the set also needs to be calculated.

The construction of this set is done in two steps. The first step runs a query on the table of all stations with a filter on the latitude and longitude. This query returns stations that are located in a virtual square box around the start coordinate. In the figure below stations B, D, E and F are returned.

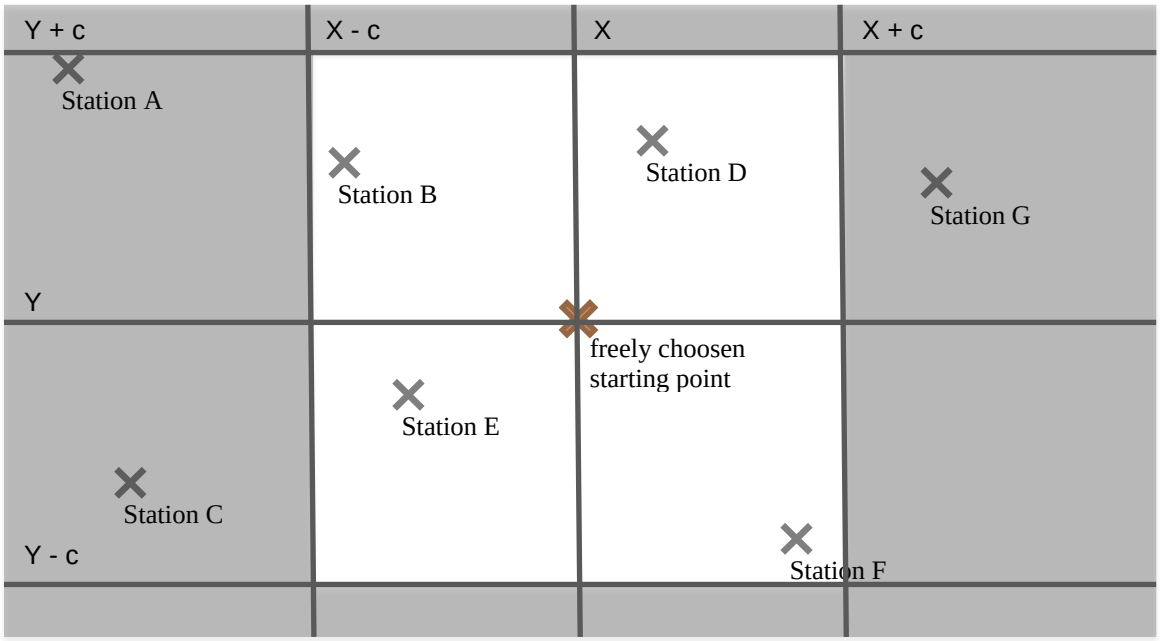


Figure 3 A virtual square box around the starting point determines which stations can potentially be reached by foot. c stands for a constant distance representing the walking speed multiplied by the maximum travel time.

The second step calculates the distance from the starting point to each station of the result set, which were found in the first step. Based on the distance, the arriving time and the remaining travel time can be calculated for each station. Stations that have negative remaining time are removed from the result set. Negative remaining time means that the station cannot be reached by foot within the time boundary. In the figure below the second step eliminates station B and F from the result set. Only station D and E can be reached by foot and act as origin for the modified Dijkstra algorithm.

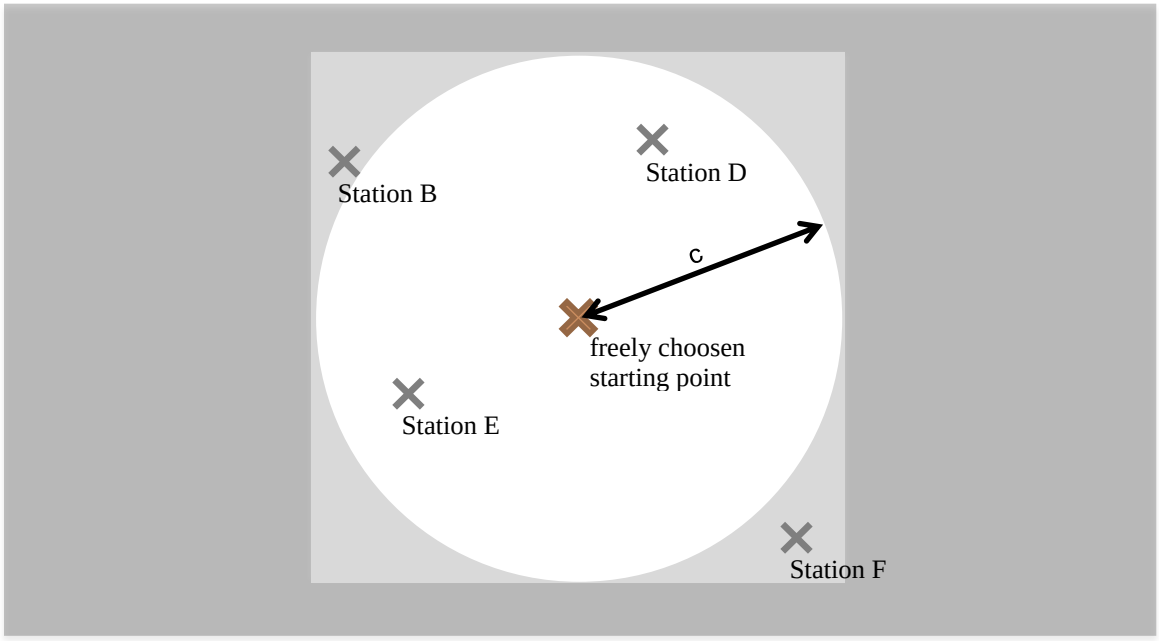


Figure 4 Only stations inside the white circle can be reached by foot within the time boundary and are therefore candidates for the modified Dijkstra algorithm.

It is necessary to mention that the first step is only a performance improvement and not required by the logic. The system would be able to find the correct station candidates equally by applying the operation from the second step to all stations in the database. What makes the difference is that the first step is much quicker than the second.

4.1.4 Algorithm: Modified Dijkstra

The core of the CIPT web service is an algorithm that traverses the network in order to find possible travel routes. We have implemented two different algorithms; the first one is a breadth-first search (BFS) and the second one is a slightly modified Dijkstra (9) algorithm (10 p. 580). A deeper inspection of the standard BFS algorithm revealed that this approach does not always deliver a correct solution, see 2.3.1. Therefore we decided to concentrate our implementation on a Dijkstra algorithm.
A difference of ours to a standard Dijkstra implementation is that we do not modify the items in the container that stores the graph. A standard Dijkstra first applies a distance for all vertices of infinite and while the algorithm is running sets the distance values on the visited vertices. Our implementation only reads from the container containing the graph and stores copies of the vertices, which do not have an infinite distance any more, in a new container called *minHeap*. This is a necessary modification in order to run the algorithm in quick succession without having the need of copying the full graph.
Also the break condition is specially adapted to our scope. The usual purpose of the Dijkstra algorithm is to find a shortest path between two vertices. It is the nature of the beast that the cost for the final solution of the problem (shortest path between the two vertices) is not known at the beginning. For our problem the question has to be put differently. We already know the cost at the beginning of the algorithm, what we would like to know is which destinations can be reached. Therefore the break condition is directly dependent on the cost – which is in our case the travel time.

For the sake of clarity we have split the implementation of the algorithm. On the one side our *minHeap* gets a lot more abilities than a usual implementation, on the other side the loop of the algorithm gets simpler. The *minHeap* will be called from the algorithm loop only by the two methods *AddIfNearer()* and *GetANearest()*.

4.1.4.1 Algorithm-loop

The read-only graph container mentioned above is actually a list of connections between two stations, called line section in the domain model. Such a connection is defined by a start and end station and also by a departure and an arriving time. In the pseudo code below this container is called *lineSectionList* and acts as static input for the algorithm. New objects are received from the list by calling the *getPossibleLineSection()* method of this container. This method receives a station and an upper time limit. The return value is a filtered list of line sections (connections) starting from the provided station. Only those connections are returned which have an earlier arriving time than the provided upper time limit and which have a later departure time than the arriving time at the start station.

Dynamic input parameters for the algorithm are a station (*startStation*) representing the vertex from which the travelled tree will grow and an upper time boundary (*timeLimit*). The arrival time of the start station needs also be set. The arrival time for the first station is the lower time boundary for the algorithm.

```
1:  Static Input: lineSectionList list of connections
2:  Dynamic Input: startStation source node, timeLimit upper time limit
3:  minHeap.AddIfNearer(startStation)
4:  v = startStation
5:  while v ≠ null do
6:      alreadyVisitedSet.Add(v)
7:      sectionList = lineSectionList.getPossibleLineSection(v, timeLimit)
8:      for each lineSection in sectionList ∉ alreadyVisitedSet do
9:          minHeap.AddIfNearer(lineSection.destinationStation)
10:     end for
11:     v = minHeap.GetANearest()
12: end while
```

Before the algorithm enters the loop the initial station is inserted into the empty minheap (line 3). The *startStation* is assigned to a placeholder variable *v* (line 4). This is important, as otherwise the break condition (*v* is null) would already be fulfilled at the beginning and the while loop would not even be entered a single time.
The loop itself puts the current station in the placeholder variable *v* in the *alreadyVisitedSet* (line 6). In the next line of code a list of line section having neighbour stations from the current station *v* that can be reached within the time boundary is loaded. All the destination stations from the connections in this list are potential candidates for the *minHeap*. They will be added only if they are not yet in the *alreadyVisitedSet* (line 9). The last task in the loop is getting the next station from the minHeap and assign it to the placeholder variable *v* (line 11). As long as the minheap is not empty *v* will not be equals *null* and the while loop starts over again.

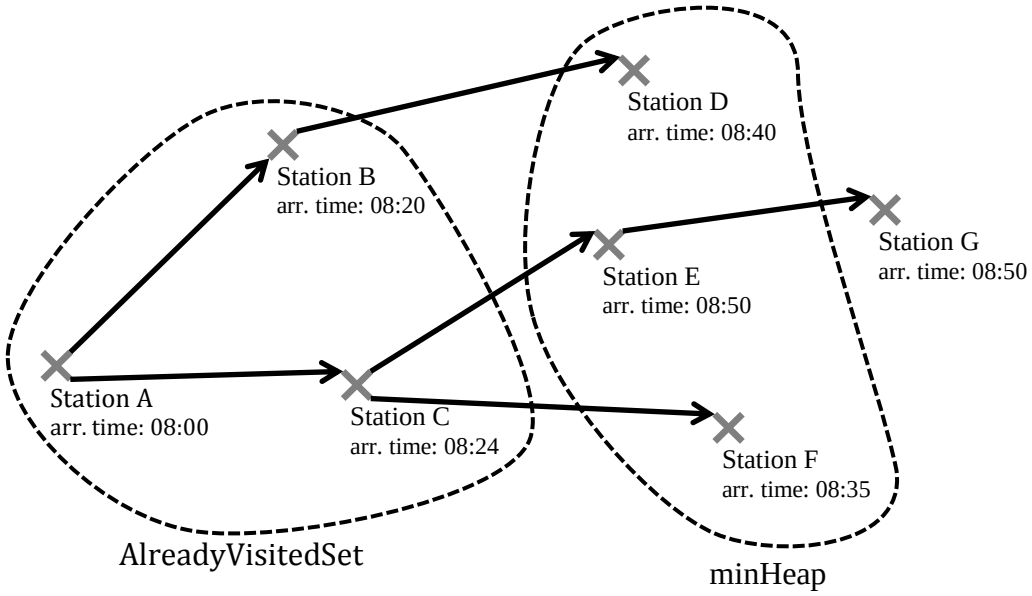


Figure 5 Discovery of stations

4.1.4.2 minHeap

The modified Dijkstra algorithm maintains a set of already visited stations. Each of these stations in the set has probably one or more neighbour stations that are not already in the set. All those neighbour stations are collected in the minheap. If a new station is added to the already visited list, a lookup for the neighbours of the new station will be done and these neighbour stations will then be added to the minheap. The minheap will also be needed if a new station is added to the already visited list. By calling GetANearest(), minheap delivers the new station.

4.1.4.2.1 AddIfNearer

This method receives a station domain object. Based on the id and the arrival time at the station the AddIfNearer method decides if the new object will be added to the heap or simply ignored. As it is not required by the modified Dijkstra algorithm the method does not provide feedback whether the station was added or ignored. If the method cannot find a station in the list that has the same id like the new candidate, the candidate will immediately be added to the heap. If there exists already a station in the heap with the equal id as the candidate, they will be compared based on the arrival time. The one with the earlier arrival time wins and will be added or remains in the heap.

4.1.4.2.2 GetANearest

This method returns a station object or null when the heap is empty. The latter is also the signal for the modified Dijkstra that the algorithm has terminated. But as long as the heap is not empty this method always returns the station with the earliest arriving time and removes it from the heap. If there is more than one station with the same earliest arriving time, one of them will be picked out and the rest remains on the heap.

4.1.5 Isochrone Construction

The result of the modified Dijkstra algorithm is a list of stations (result list). Each station in the list has an individual arrival time and remaining time span.

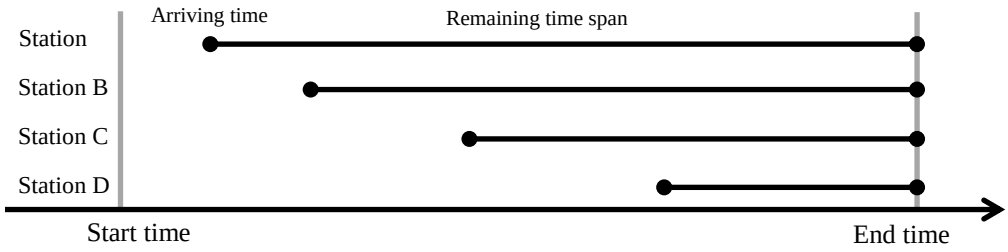


Figure 6 Symbolisation of the result station list

But what the CIPT web service returns is not any kind of station list; it is a set of points that will form polygons representing isochrone areas. Therefore some program logic was developed that does this conversion.
The first step is to determine the different times for isochrones. This is done by dividing the time span between start and end time into N equal time slots whereby N stands for the number of requested isochrones. The end time of each time slot will then be the time of an isochrone.

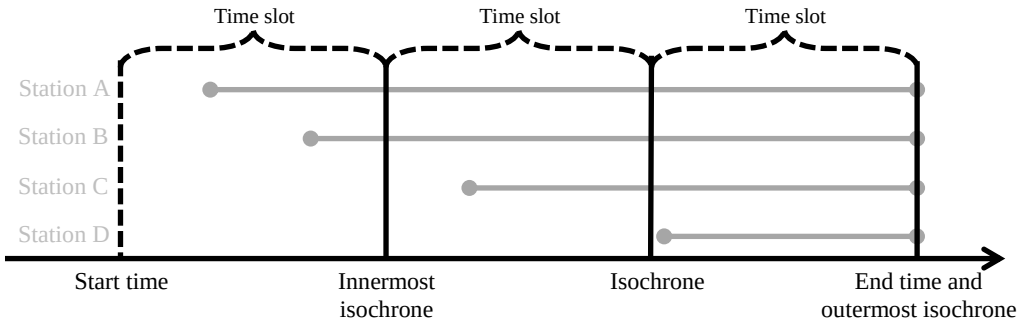


Figure 7 Symbolisation of the calculation of the different isochrones times

Every isochrone will now get its own list of stations (isochrone list). These N isochrone lists will be filled with copies of stations from the result list. A specific isochrone list will receive a copy of a station only if the arriving time of the station is smaller/sooner than the time of the isochrone. If the arrival time plus the remaining time of the copy is bigger / later than the time of the isochrone, the remaining time will be shortened so that arrival time plus remaining time is equal to the isochrone time.

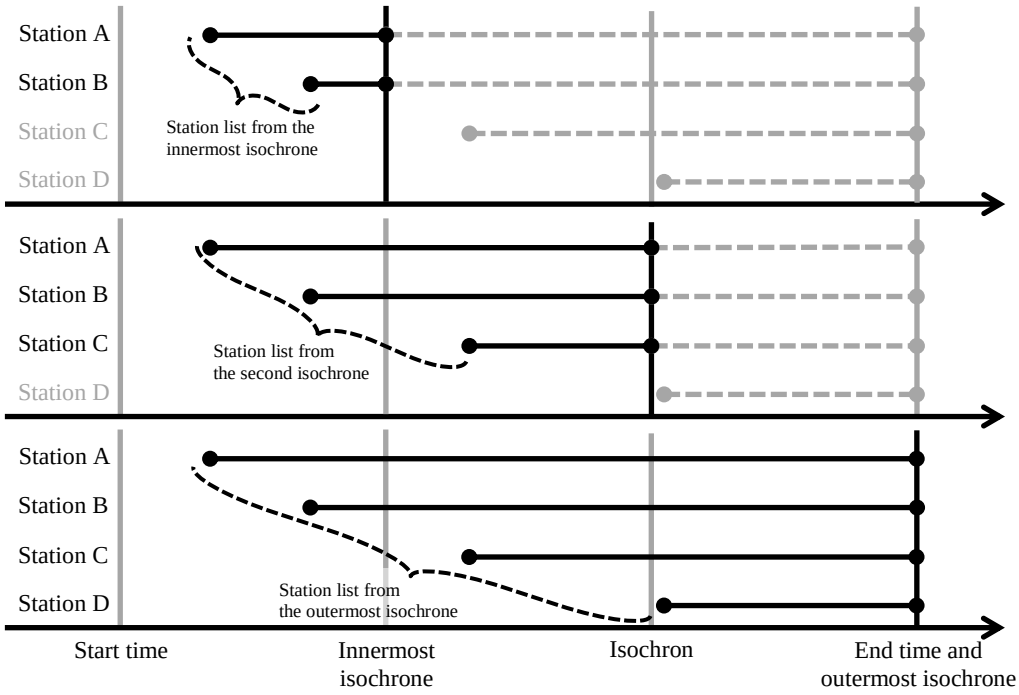


Figure 8 The figure shows how the station lists for the isochrones are built and how the remaining time of the stations are truncated

At the end of this copy procedure each isochrone list contains stations where the arrival time plus the remaining time is smaller or equal than the isochrone time.
With the help of a library from computational geometry each isochrone list will now be transformed to a multipolygon (11). The first step is to draw a polygon like a circle around every station in the list. This can be achieved with a buffer method using the remaining time of the station multiplied by a static walking speed factor as radius of the approximated circle.

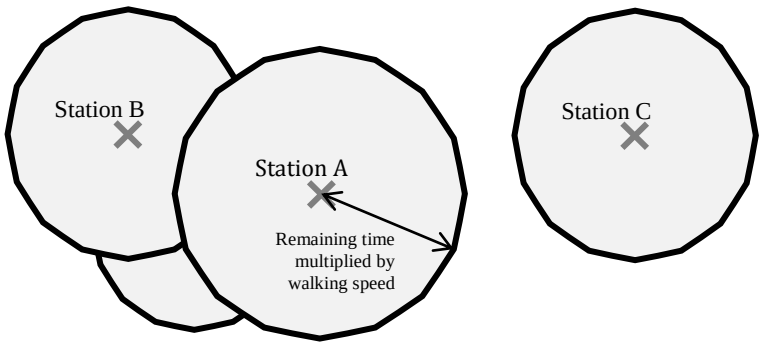


Figure 9 Remaining walking time at each station shown as overlapping approximated circles

The next step is to merge all the approximated circles to one multipolygon with the help of a union method. The resulting multipolygon can be composed of several areas that do not touch each other. It is also possible that an area has holes in it.

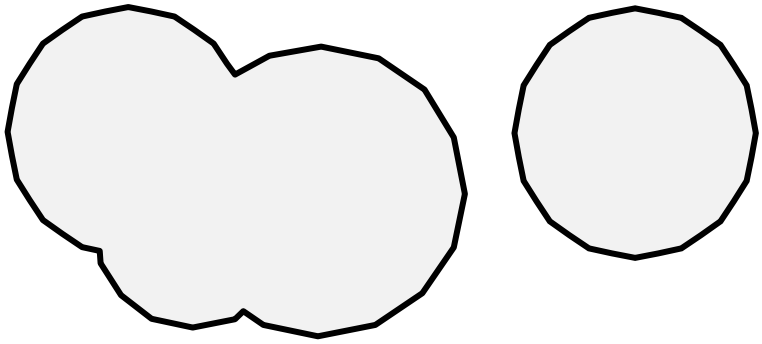
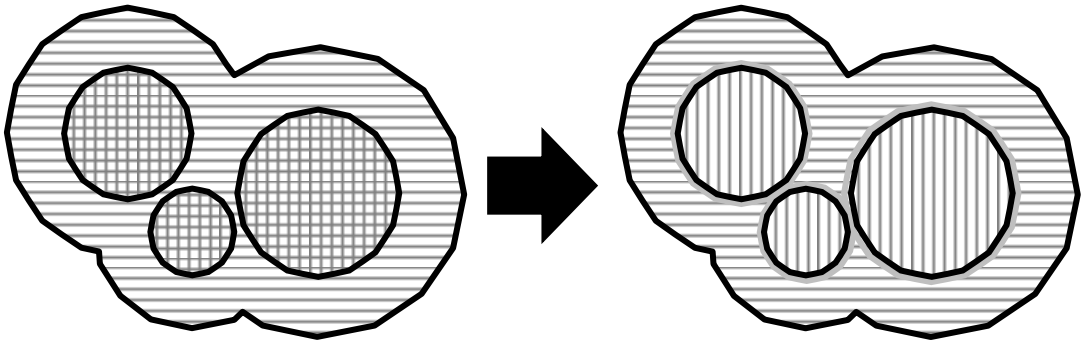


Figure 10 Multipolygon construction as result of a union

We now have a multipolygon for each isochrone. The individual multipolygons not only touch each other, they are completely embedded by the one that belongs to an isochrone having a bigger / later time. As tests with clients have shown this does not yield a satisfying result. Multipolygons overlapping each other are not rendered nicely by clients. Therefore the overlapping has to be removed. This is done by removing the overlapping area from the larger multipolygon.



Two multipolygons belonging to different isochrones are overlapping

Removed overlapped area from the bigger multipolygon

Figure 11 Inner isochrones are cut-out from enveloping isochrone; therefore do not lie on top of each other

The outer boundary of the multipolygon is now the isochrone itself.
A difference between KML (12) and GML (13) is that KML is not only able to describe the geometric form of an object; KML can also contain information about how the object has to be presented. In our test client we need this feature to provide each isochrone area with an individual colour.

4.1.6 WPS, WFS Web Services

Both OGC interface standards are implemented as far as required by the CIPT web service, including error messages. An implementation of all features is out of the scope for this project.

5 Results, Validation and Outlook

5.1 Target Achievement

As a result of our project work, we were successfully able to implement a service that computes isochrones within the described limitations. Our algorithm shows that calculation and presentation of isochrones for multi-modal public transport systems can be achieved in a contemporary matter.

5.2 Benefits, possible Scenario of Utilization

In addition to our vision presented in 1.1, the following scenarios indicate a possible use.

- Decision-Support Systems
- Planning Systems
- Location Based Services

5.3 Outlook: Further Development

5.3.1 Website Integration

In order to present our work to the general public, a website with a mapping service could be developed.

5.3.2 Performance Improvement

As the result of this service is a feasibility study, the following improvements may point the way to a service consumable by the general public through a website.

5.3.2.1 Algorithm

As mentioned in 1.3.4.1, absolute algorithm performance was not a target for this project. An improved algorithm could be discovered.

5.3.2.2 Parallelisation

The current implementation doesn't support parallel requests. In addition, the algorithm itself may benefit from future parallelisation advancements provided by the run-time or language.

5.3.2.3 Pre-processing

Since the time table is static data valid for a years' period, an alternative to … the algorithm for each request, one could pre-process the results of an isochrone calculation for every station for all possible time intervals.

5.3.2.4 Caching

The most frequently or last few runs of the algorithm could be stored in a cache for accelerated equal future requests.

5.3.3 Graph Database

Time table data currently resides in a relational database management system. An alternative approach to persist the graph directly could be achieved by using a graph database. For the .NET Framework, Sones could be a possible choice (14).

5.3.4 Join with Other Data Sources

The underlying time table data and diverse other data source could be included as a filter into algorithm processing, allowing the user to query the service in more customized ways with enhanced results.

5.3.5 Update Mechanism for Time Table Information

Currently, time table data is parsed from text files and loaded into the database which remains valid for a time period. Synchronisation mechanisms onto official data sources could be developed.

5.3.6 Display of Remaining Walking Time on a Street Network

Instead of visualising the remaining walking time as a circle around a station, the algorithm could use as a basis for further exploration a street network, like OpenStreetMap (15). An intersection is a node and a street is a vertex in the graph.

5.4 Acknowledgements

We would like to thank Prof. Keller for his support. We also take this opportunity to thank SBB and ZVV for providing us with the HAFAS data.

Part 2: SW-Project Documentation

7 Vision

Siehe Kapitel 1.1.

8 Anforderungsspezifikation

8.1 Anforderung an die Arbeit

1. Es sollten zuerst Algorithmen zur Berechnung der Isochronen evaluiert werden. Darauf aufbauend soll ein Algorithmus ausgewählt, ggf. verbessert und umgesetzt werden.
2. Die Applikation soll einen Webservice (sog. WPS) anbieten (mit einem passenden Format, z.B. GeoJSON oder GML sowie ggf. ein geläufiges Geo-Rasterformat).
3. Die Resultate der Arbeit sollen auf einer eigenen Webseite präsentiert werden.

8.2 Use Cases

8.2.1 UC1 Shows Capabilities

Use Case 01	**Show Capabilities**
Anwendungsbereich	CIPT
Level	Benutzer Ziel
Primärer Akteur	OGC WFS/WPS User
Stakeholder und Interessen	OGC WFS/WPS User: will eine Beschreibung der gültigen Parameter der angebotenen Dienste.
Voraussetzungen	User verwendet ein System, welches OGC standardkompatibel ist.
Erfolgsgarantie	User erhält eine Beschreibung der gültigen Parameter.
Hauptszenario	1. User startet neue Anfrage. 2. System zeigt Liste mit Service-Beschreibungen und deren Verwendungs-Parametern an.
Erweiterungen	2a. Anfrage ist falsch formuliert. 1. System meldet einen Fehler und bricht Verarbeitung ab. *a. Jederzeit: das System stürzt ab. 1. System meldet einen Fehler und bricht Verarbeitung ab.
Spezielle Anforderungen	
Technologie und Daten Format Liste	Anfragen des Users entsprechen offiziellem OGC Standard.
Auftretenshäufigkeit	Nicht beachtet.
Sonstiges	

8.2.2 UC2 Show Description

Use Case 02	**Show Description**

Anwendungsbereich	CIPT
Level	Benutzer Ziel
Primärer Akteur	OGC WFS/WPS User
Stakeholder und Interessen	OGC WFS/WPS User: will eine Beschreibung der angebotenen Dienste sehen.
Voraussetzungen	User verwendet ein System, welches OGC standardkompatibel ist.
Erfolgsgarantie	User erhält eine Beschreibung der Dienste
Hauptszenario	1. User startet neue Anfrage. 2. System zeigt Liste mit Service-Beschreibungen an.
Erweiterungen	2a. Anfrage ist falsch formuliert. 1. System meldet einen Fehler und bricht Verarbeitung ab. *a. Jederzeit: das System stürzt ab. 1. System meldet einen Fehler und bricht Verarbeitung ab.
Spezielle Anforderungen	
Technologie und Daten Format Liste	Anfragen des Users entsprechen offiziellem OGC Standard.
Auftretenshäufigkeit	Nicht beachtet.
Sonstiges	

8.2.3 UC3 Get Isochrones

Use Case UC3	**Get Isochrones**
Anwendungsbereich	CIPT
Level	Benutzer Ziel
Primärer Akteur	OGC WFS/WPS User
Stakeholder und Interessen	OGC WFS/WPS User: will zeitnahe und exakte Isochrone als Antwort auf seine Anfrage.
Voraussetzungen	User verwendet ein System, welches OGC standardkompatibel ist. Zeitintervall ist am gleichen Tag, d.h. Endzeit ist grösser als Startzeit. Koordinate liegt innerhalb der Schweiz Anzahl Isochronen-Intervalle ist kleiner als 60.
Erfolgsgarantie	Angeforderte Anzahl Isochronen wurde generiert.
Haupterfolgs Scenario	1. User startet neue Anfrage. 2. User gibt Koordinate ein. 3. User gibt Startzeit ein. 4. User gibt Endzeit ein. 5. User wählt Anzahl Isochronen-

	Intervalle. 6. System liefert Isochrone.
Erweiterungen	2a. Koordinate im falschen Format. 1. System meldet einen Fehler und bricht Verarbeitung ab. 2b. Koordinate liegt ausserhalb der Schweiz 1. System meldet einen Fehler und bricht Verarbeitung ab. 4a. Endzeit ist kleiner als Startzeit. 1. System meldet einen Fehler und bricht Verarbeitung ab. 5a. Anzahl Isochronen-Intervalle ist grösser als 60. 1. System meldet einen Fehler und bricht Verarbeitung ab. *a. Jederzeit: das System stürzt ab. 1. System meldet einen Fehler und bricht Verarbeitung ab.
Spezielle Anforderungen	Die Antwortzeit des Systems ist kleiner als 1 Minute.
Technologie und Daten Format Liste	Anfragen des Users entsprechen offiziellem OGC Standard. 2a. Koordinaten sind im WGS oder CH1903 Format. 3-4a. Zeitangaben sind im Format [hhmm].
Auftretenhäufigkeit	Nicht beachtet.
Sonstiges	

8.2.4 Use Case Diagramm

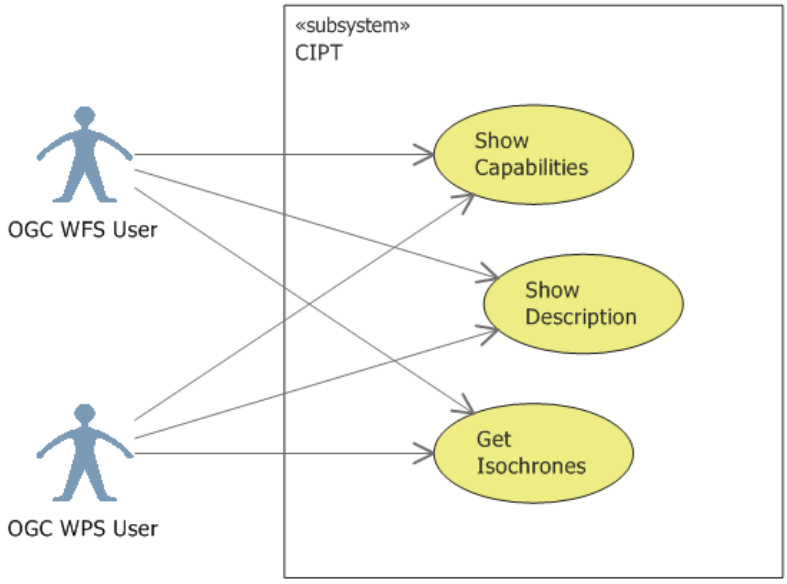


Figure 12 Use Case Diagramm

8.3 System-Sequenzdiagramme

Die Systemoperationen sind durch die OGC Standards genau vorgegeben. UC 01 bis UC 03 sind in einem SSD zusammengefasst und pro Actor dargestellt.

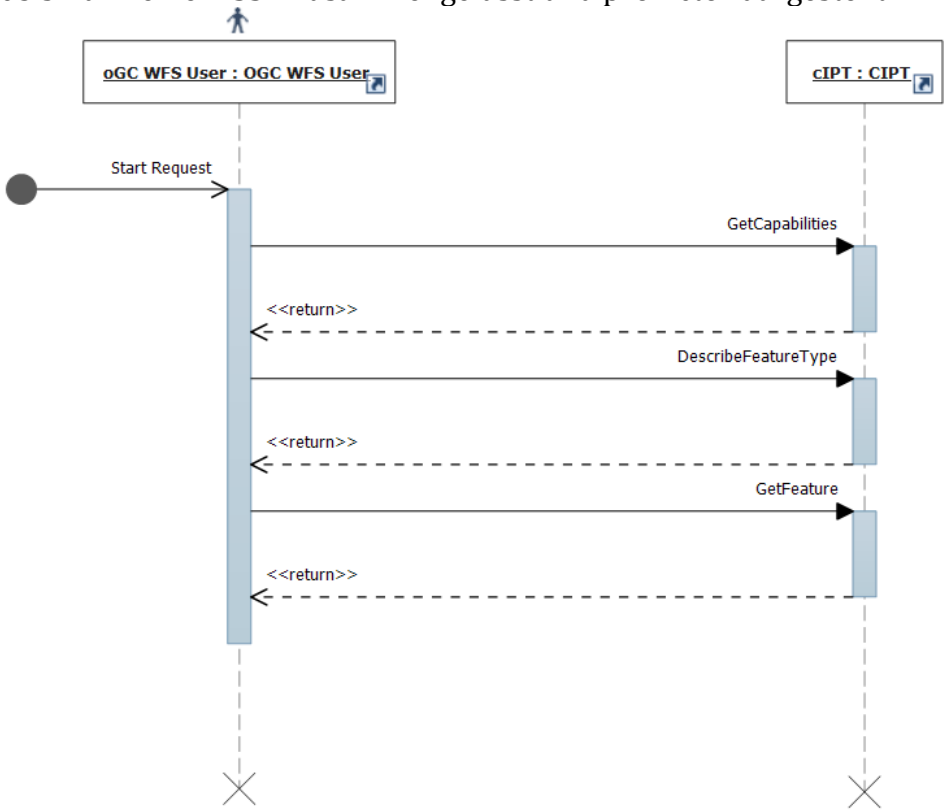


Figure 13 SSD für WFS

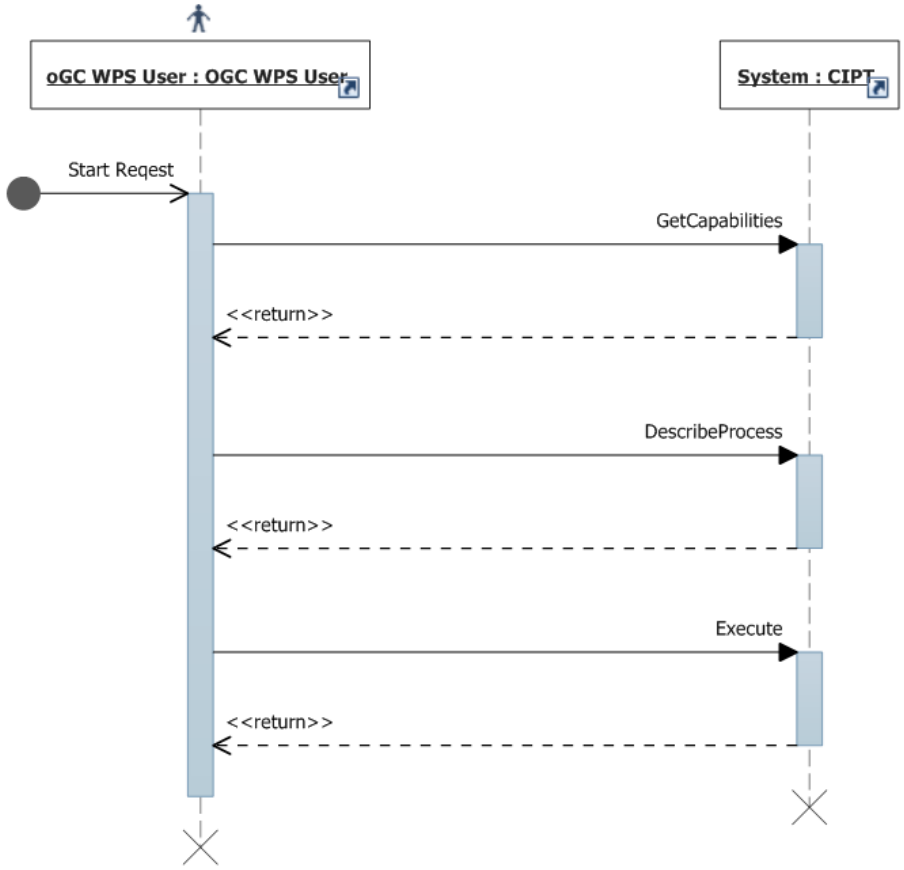


Figure 14 SSD für WPS

8.4 Nicht-funktionale Anforderungen

8.4.1 Antwortzeit

Der Web Service soll 80% der Anfragen innert 5 Sekunden beantworten. Eine genauere Übersicht über unsere Anforderungen steht in 3.1.1.2. Die Messungen in 3.2.2 zeigen, dass der Dienst diese Vorgabe erfüllt.

8.4.2 Usability

Der Web Service soll Isochronen liefern, die der geläufigen Usability unterliegen, z.B. bezüglich Farben und Form. Für den User soll klar erkennbar sein, welche Zeitzone zu welcher Isochrone gehört.

8.4.3 Verfügbarkeit

Der Web Service muss keine Verfügbarkeits-Anforderung erfüllen, da es sich um eine Machbarkeits-Analyse handelt.

9 Analyse

9.1 Domain Modell

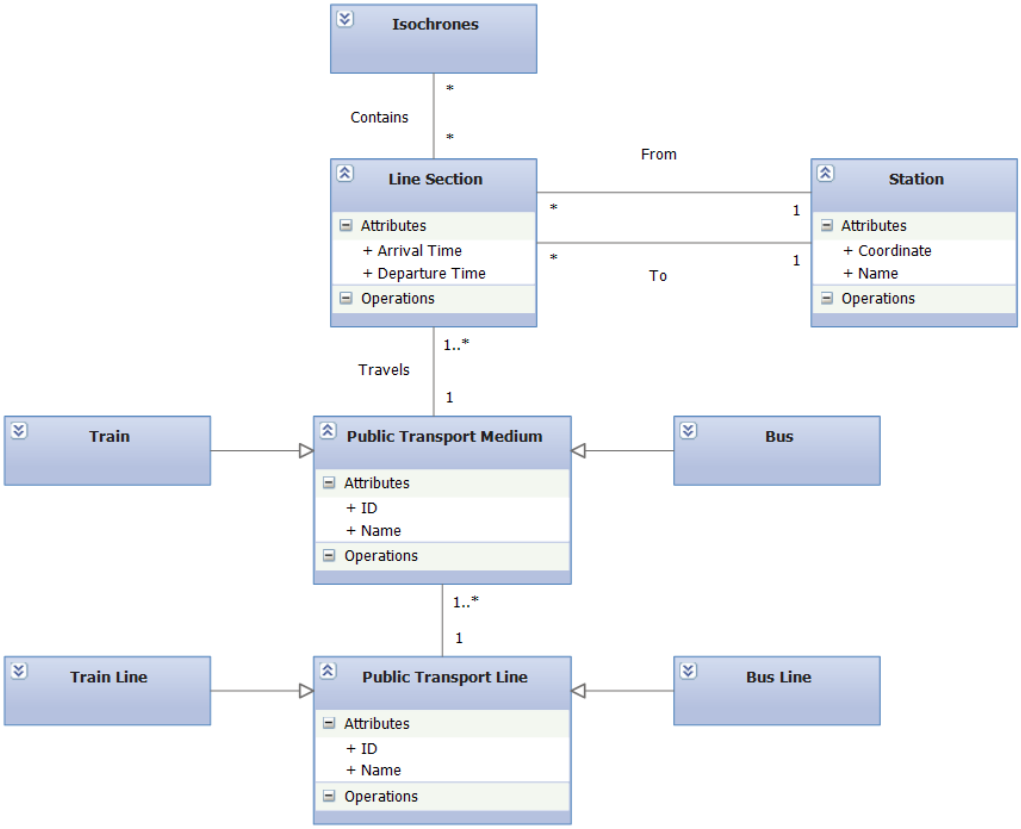


Figure 15 Domain Modell

9.2 Datenbankschema

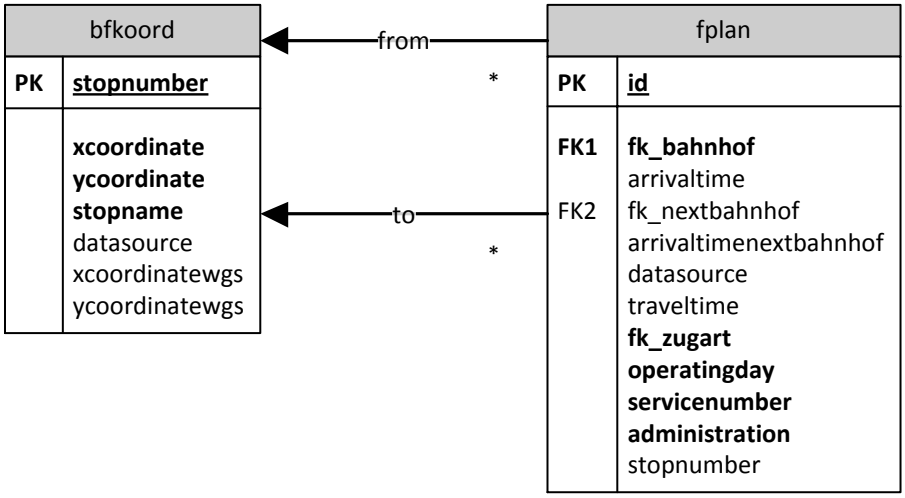


Figure 16 Datenbank Schema

Auf eine Normalisierung hin zur 3. Normalform (NF) wurde aus Performance-Gründen verzichtet. Die Speicherung in der 3. NF bringt keinen nennenswerten Speicherplatz-Vorteil. Das denormalisierte Schema hingegen vereinfacht das Laden des Grafen, siehe 4.1.2.

9.3 Algorithmen

9.3.1 BFS

Siehe 2.3.1.

9.3.2 Dijkstra

Siehe 4.1.4.

9.4 Evaluation von Tools und Bibliotheken

9.4.1 Graphen Bibliotheken

Für die Abbildung des öffentlichen Verkehrsnetzes als Graph haben wir verschiedene Graphen Bibliotheken evaluiert. Aufgrund der Anforderungen in der Aufgabenstellung muss die Bibliothek in C# geschrieben sein und idealerweise das .NET Framework 4 unterstützen.
Vorwegnehmend lässt sich erwähnen, dass wir keine geeignete Bibliothek gefunden haben, insbesondere aufgrund der Eignung für unsere Problemstellung und fehlender/lückenhafter Dokumentation im open source Umfeld. Die Evaluation hat aber viel Projektzeit in Anspruch genommen, siehe 14.2.
Folgende Kriterien wurden beachtet:

- Vorhandensein von Standard Algorithmen (Dijkstra)
- Vorhandensein einer Priority Queue
- Umfang/Komplexität allgemein
- Einbindbarkeit in unsere Problemstellung

9.4.1.1 QuickGraph

QuickGraph (16) ist ein Port der Boost Graph Library (17) nach C#. Die Bibliothek ist sehr umfangreich und eignet sich aufgrund der Komplexität nicht, um einen eigenen Algorithmus einzubringen. Ein ‚hello-world' Beispiel mit einem Dijkstra konnte getestet werden.

9.4.1.2 Graphen Bibliothek aus Programmieren 2

Das Modul Programmieren 2 verwendet als Grundlage das Buch Algorithms & Datastructures in Java (18). Da die Bibliothek in Java geschrieben wurde, probierten wir einen Port nach C# zu realisieren. Da das Buch nicht auf den .NET Datenstrukturen aufbaut, sondern jede benötigte Datenstruktur selber implementiert, musste auch die Portierung dieser Strukturen in Betracht gezogen.
Auch diese Library eignete sich nicht für unsere Problemstellung Sie ist teils zu simpel und an anderen Stellen zu komplex, vor allem aufgrund vieler Abhängigkeiten.

9.4.1.3 GraphLib

GraphLib wurde nach kurzer Evaluation als zu trivial befunden. Einen Dijkstra-Algorithmus implementiert die Library nicht.

9.4.2 WPS Implementierung

9.4.2.1 plWPS

plWPS sollte laut eigenen Angaben einen WPS-Dienst in der Version 1.0.0 in ASP.NET implementieren. Die XML-Schema Definitionen sind aber veraltet, sodass wir uns entschieden, den WPS-Dienst selber zu implementieren.

9.4.3 Geometrie-Bibliotheken

Die Generierung der Isochronen aus den Zielbahnhöfen erfordert einige komplexe geometrische Operationen (4.1.5). Es war für uns klar, dass wir diese Operationen nicht selber implementieren wollten. Deshalb wollten wir zu Beginn des Projektes diese Aufgabe an PostGIS übergeben (19). Nachdem wir die beiden untenstehenden Bibliotheken gefunden hatten, war es offensichtlich, dass PostGIS keine Alternative darstellt, da wir bei dessen Verwendung unnötige Datenverschiebungen und somit Performanceeinbussen hätten in Kauf nehmen müssen.

9.4.3.1 NetTopologySuite

NetTopologySuite (20) ist ein direkter Port der bekannten JTS-Geometrie-Bibliothek (Java) nach C#. Da die Portierung noch nicht ganz abgeschlossen ist, einige Funktionsnamen nicht dem Original entsprechen und fast keine Dokumentation vorhanden ist, gestaltete sich die Implementierung dieser Bibliothek als Ratespiel. Grundsätzlich erfüllte diese Bibliothek jedoch unsere Anforderungen und wir konnten damit eine lauffähige Implementation erstellen.

9.4.3.2 SQL Server 2008 Spatial Data Type

Die SQL Server 2008 spatial data typen (21) sind Teil des SQL Server 2008. Die Datentypen können jedoch auch unabhängig vom Datenbankserver kostenlos bei Microsoft heruntergeladen werden und auf einem System installiert werden. Da die Funktionalität dieser Bibliothek wie auch die der NetTopologySuite grundsätzlich OGC

konform sind, konnten wir NetTopologySuite einfach mit den SQL Server 2008 – Datentypen ersetzen.
Wir haben uns schliesslich für die definitive Verwendung dieser zweiten Möglichkeit entschieden, da uns die NetTopologySuite zu fehleranfällig erschien und im Unterschied zur Microsoft-Lösung keine Dokumentation vorhanden ist.

9.4.4 Desktop/Client Applikationen

Gemäss Anforderungen sollte unser WPS-Service mit einer Desktop/Client Applikation getestet werden können. Alle evaluierten Applikationen stellen diese Funktionalität als Plug-In zur Verfügung. Als Kriterium untersuchten wir die korrekte Funktionsweise mit einem öffentlichen WPS-Dienst (22).
Da keine der evaluierten Applikationen einen WPS-Dienst korrekt verarbeiten konnte, musste im Projekt als zusätzliche Anforderung ein WFS-Dienst implementiert werden, siehe auch 11.2.2. Als Akzeptanztest diente uns danach wieder Quantum GIS.
Für die Evaluation aller Clients wurde viel Projektzeit verwendet, siehe 14.1.

9.4.4.1 Quantum GIS

Das Plug-In von kappsys.ch (23) sollte laut eigenen Angaben einen WPS-Dienst verarbeiten können. Leider konnte kein einziger der öffentlichen WPS-Services damit getestet werden.

9.4.4.2 OpenJump

Für OpenJump existiert ein WPS Plug-In (24). Dieses liefert keine korrekten Resultate.

9.4.4.3 MapWindow

MapWindow ist eine Open Source Desktop GIS Applikation für die Windows-Plattform (25). Die Entwickler beschreiben in ihrem Artikel eine ActiveX Erweiterung für WPS (26). Diese lässt sich leider nicht auffinden.

9.4.4.4 uDig

uDig wurde nur kurz als Alternative evaluiert (27). Der WPS Assistent funktioniert leider nicht korrekt. Es werden Fehlermeldungen angezeigt und es kann kein Import der Daten durchgeführt werden.

10 Design

10.1 Architektur

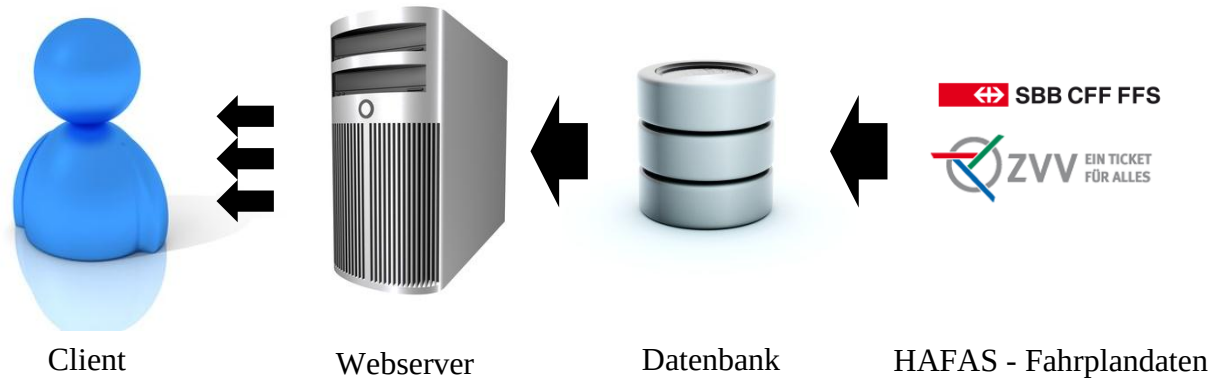
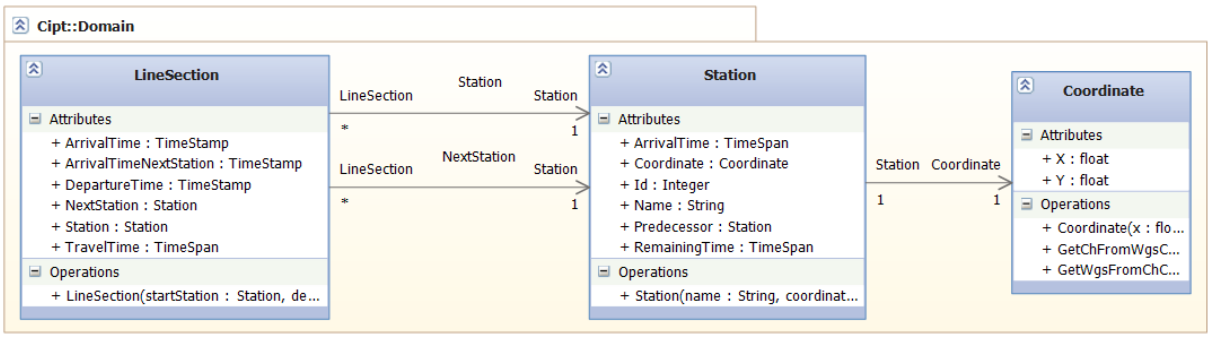


Figure 17 Physische Architektur

Figure 17 zeigt den Fluss der Daten. Die HAFAS-Daten müssen bei einer Neuerscheinung eines Fahrplans (normalerweise einmal pro Jahr) in die Datenbank geladen werden. Der Webserver lädt diese Daten beim Starten von der Datenbank. Vom Webserver gehen jeweils kleinere Mengen der Daten bei jeder Abfrage zum Client.

10.2 Objektkatalog



10.3 Package- und Klassendiagramme

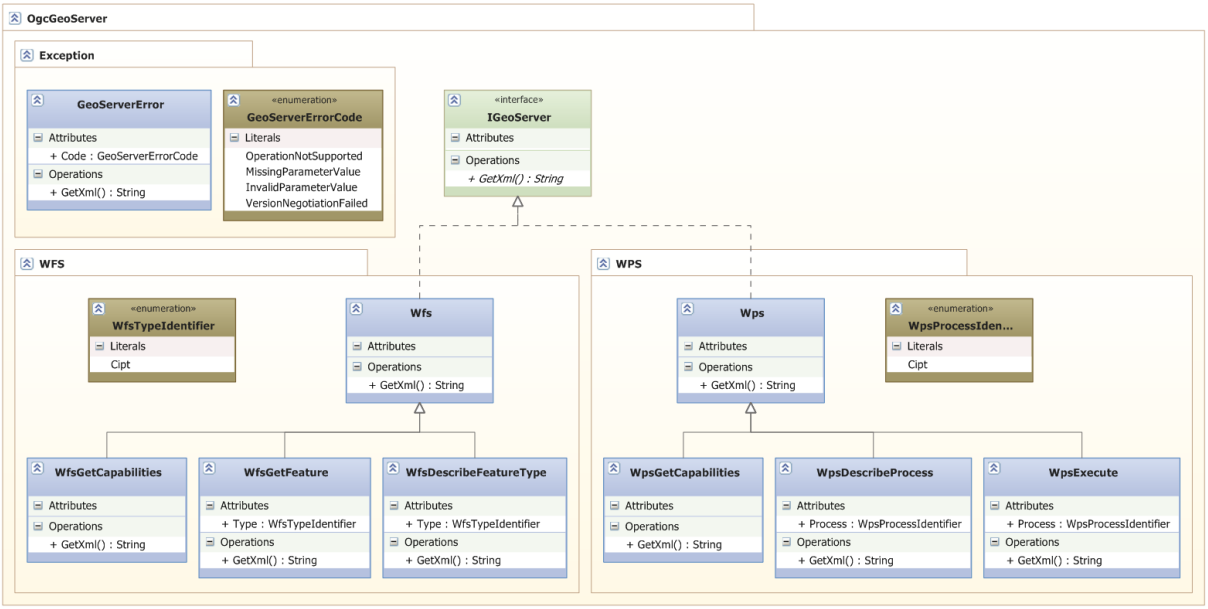


Figure 18 Package Diagramm OgcGeoServer

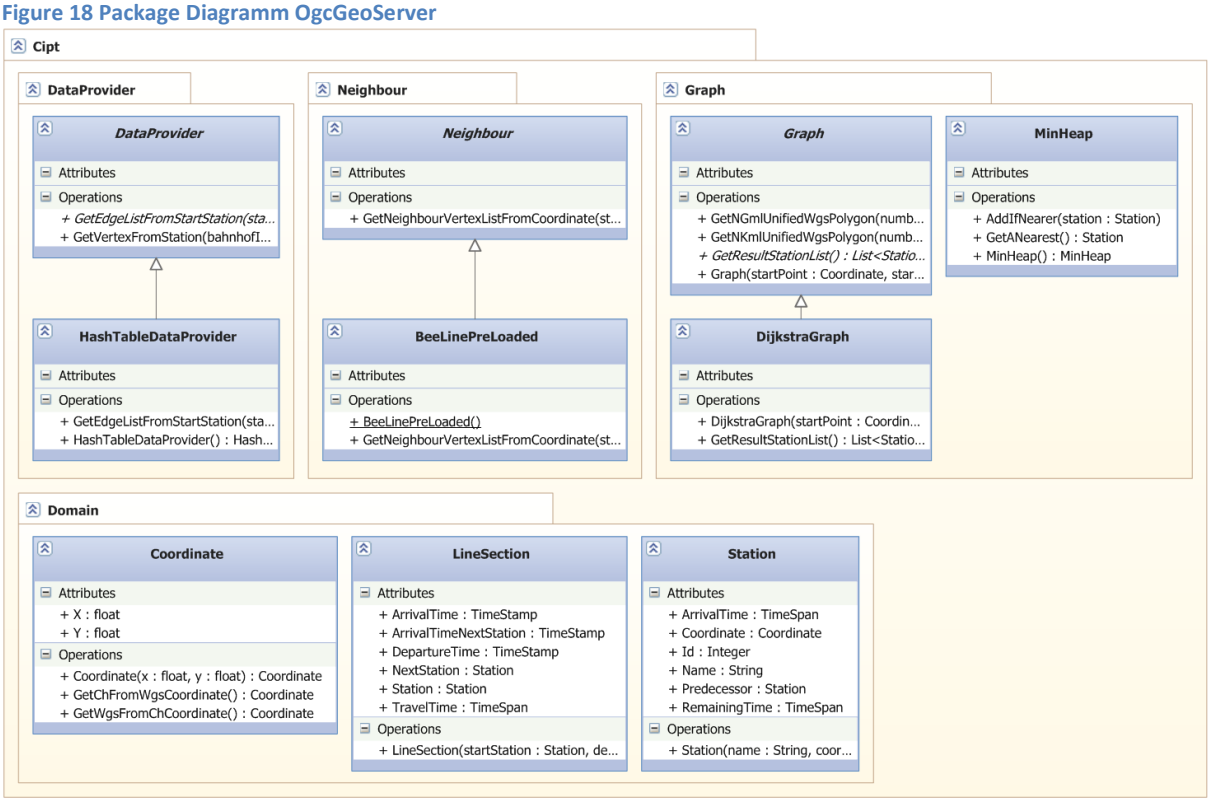


Figure 19 Package Diagramm Cipt

10.4 Sequenzdiagramme

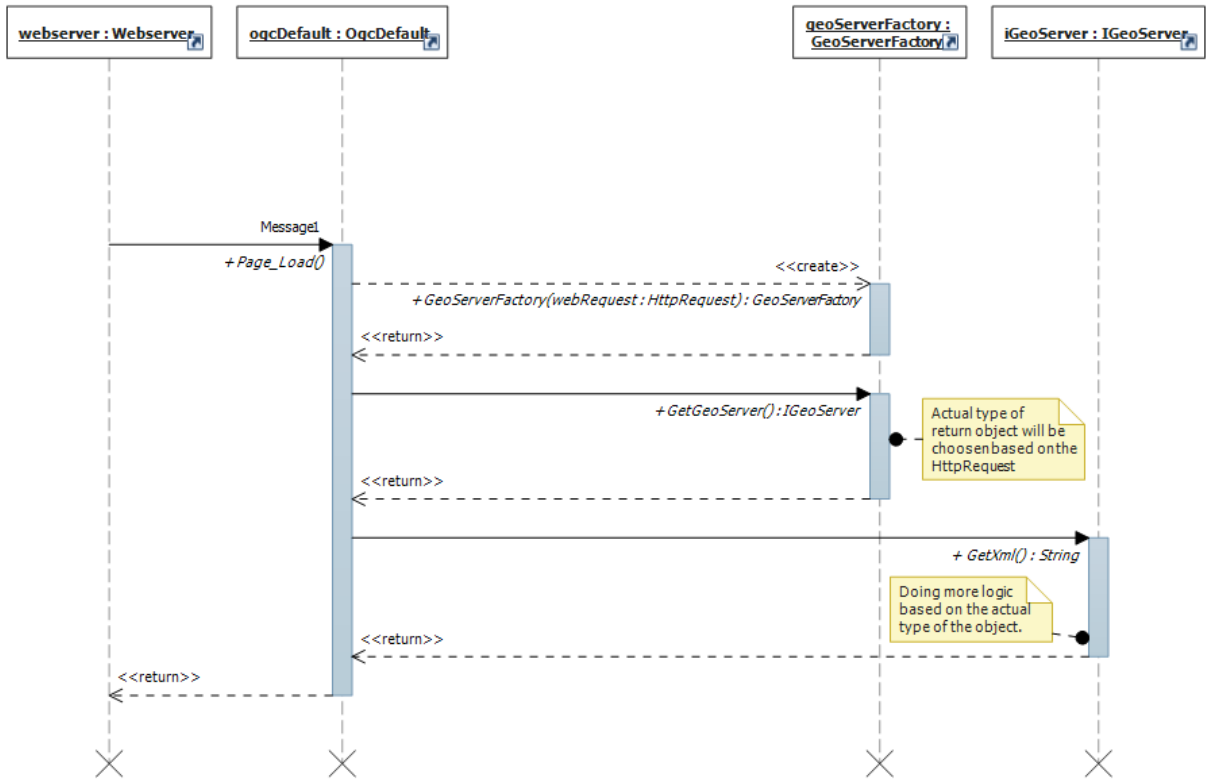


Figure 20 Aufruf OgcGeoServer

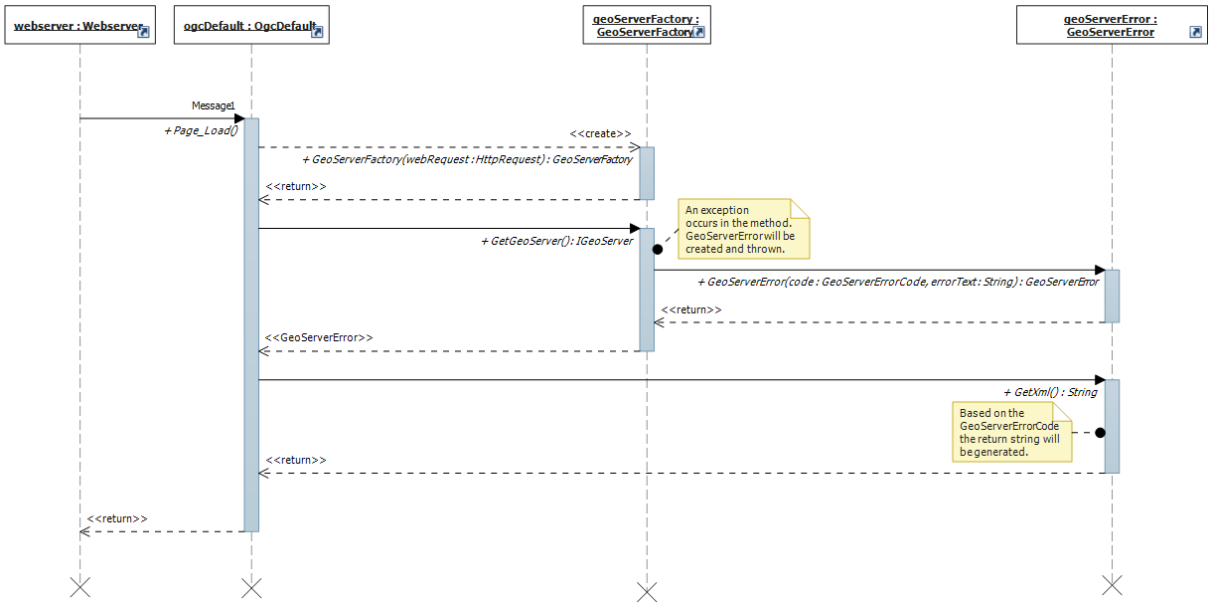


Figure 21 Exception Handling

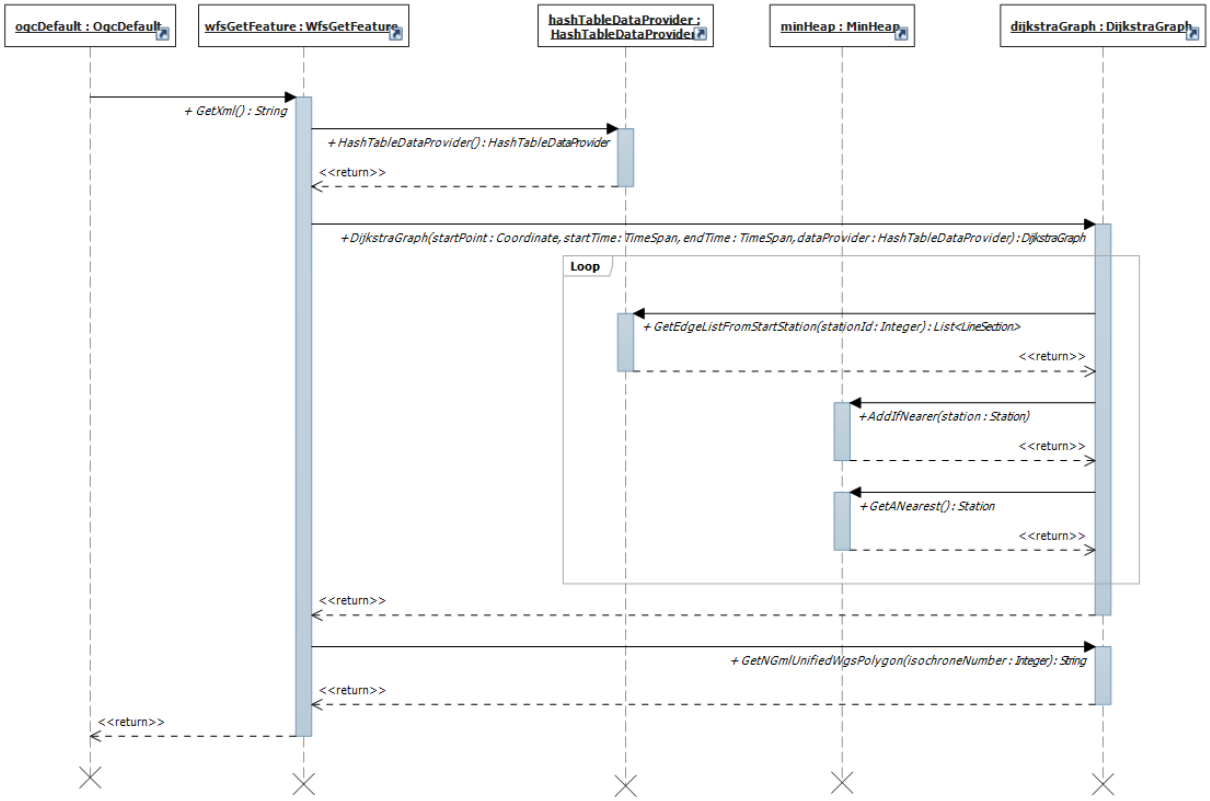


Figure 22 WPS Execute oder WFS GetFeature Aufruf

11 Implementation und Test

11.1 Implementation: Erläuterungen wichtiger konkreter Klassen

Die Klassen HashTableDataProvider und BeeLinePreLoaded haben beide einen statischen Konstruktor. Diese Konstruktoren werden nur einmal beim Starten des Programmes, respektive beim Hochfahren des Webservers ausgeführt. Beide Konstruktoren haben die Aufgabe, Daten von der Datenbank zu laden und in einem statischen Feld der jeweiligen Klasse abzulegen. Dieser Vorgang kann sehr lange dauern – erhöht dafür aber die Performance des CIPT Webservices im Betrieb stark.

Die statische Klasse ApproxSwissProj (in der Datei wgs84_ch1903.cs) konnten wir bei (28) beziehen. Mit dieser Klasse können Koordinatenpunkte einfach zwischen den beiden Koordinatensystemen WGS84 und CH1903 umgerechnet werden.

11.2 Implementation der OpenGIS Web Services (OWS)

Mit WPS können die Query-Argumente frei definiert werden. Dadurch ist dieser Service-Typ relativ flexibel einsetzbar. Die ursprüngliche Idee der Studienarbeit war deshalb, lediglich einen WPS anzubieten. Im Laufe der Arbeit konnte jedoch kein zuverlässig funktionierender WPS-Client für das Testen unseres Services gefunden werden. Deshalb haben wir uns entschlossen, ebenfalls einen WFS anzubieten. Im zweiten Teil der Studienarbeit wurde deshalb der Schwerpunkt auf die Realisierung eines WFS gelegt.

11.2.1 WPS

Wie oben beschrieben, konnten wir den WPS mangels eines funktionierenden Clients nicht testen. Lediglich die korrekte Funktionsweise der beiden Operationen „GetCapabilities“ und „DescribeProcess“ konnte mit Quantum GIS überprüft werden. „Execute" als dritte und eigentlich wichtigste Operation (mit „Execute“ werden die Daten abgefragt) funktionierte leider nicht. Mit der Hilfe des WPS Standards konnten wir uns jedoch einen möglichen GET-Request für die Execute-Operation zusammenstellen (29 p. 39) und dies mit einem Webbrowser testen.

```
http://foo.bar/foo?request=Execute&service=WPS&version=1.0.0&language=en&Identifier=cipt&DataInputs=IsochroneNumber=3;StartTime=0800;EndTime=0845;StartCoordinate=8.836,47.23
```

Table 4 Beispiel einer korrekten GET-Requests für eine WPS-Execute Operation

Obiger Request liefert mehrere Multipolygone im GML-Format. Der Value-String für den DataInputs-Parameter ist gleich aufgebaut und unterliegt den gleichen Restriktionen wie der Filter-String für den WFS (11.2.2).

11.2.2 WFS

Im Gegensatz zu WPS- wird der WFS-Standard von Quantum GIS sogar ohne die Installation von zusätzlichen Plug-Ins unterstützt. Leider sieht der WFS-Standard jedoch die Verwendung von servicespezifischen Argumenten nicht vor. Der Standard beschreibt lediglich, dass optional eine Bounding-Box (30) angegeben werden kann um

einen Ausschnitt einer Karte zu definieren. Falls der CIPT-WFS erkennt, dass eine Bounding-Box mitgeliefert wird, dient das Zentrum der Box automatisch als Startpunkt für den Algorithmus.
In der aktuellen Quantum GIS-Entwicklerversion kann beim Aufruf eines WFS ein zusätzlicher Filter-String angegeben werden. Dieser Filterstring wird vom CIPT-WFS ausgewertet. Untenstehende Tabelle zeigt die unterstützten Parameter.

Parameter-Name	Wertebereich	Standardwert
IsochroneNumber	Natürliche Zahl von 1 bis 60	3
StartTime	0000 bis 4759	0800
EndTime	0000 bis 4759	0900
StartCoordinate	5.0,45.0 bis 12.0,48.0 (WGS) oder 400000,10000 bis 900000,300000 (CH1903)	Mittelpunkt der Bounding-Box

Table 5 Mögliche Parameter für den Filter-String des CIPT-WFS

Parameter und Wert werden durch ein Gleichheitssymbol (=) voneinander getrennt. Mehrere Parameter-Wertkombinationen werden durch ein Semikolon (;) getrennt. Die Zeitangabe für StartTime und EndTime soll die Uhrzeit im 24-Stunden-Format, jedoch ohne Trennzeichen sein. Beispielsweise wäre 0855 die Darstellung für 8.55 Uhr. 0875 wäre eine ungültige Zeitangabe, da der Minutenteil grösser als 59 ist. Das HAFAS Rohdatenformat (3) unterstützt auch Verkehrsverbindungen, die über den aktuellen Tag hinaus reichen. Beispielsweise könnte ein Zug um 23.45 Uhr in Rapperswil abfahren und zwei Stunden später um 1.45 Uhr am nächsten Tag in Bern ankommen. Die Ankunftszeit wird dann als 2545 im Filterstring eingetragen. StartCoordinate erwartet zwei kommagetrennte Zahlen, die eine Koordinate innerhalb der Schweiz beschreiben. Die Koordinate kann entweder in Schweizer Landeskoordinaten (CH1903) oder im World Geodetic System (WGS) angegeben werden. Da der CIPT-Webservice nur Koordinaten innerhalb der Schweiz erwartet, kann er diese beiden Systeme automatisch unterscheiden. Falls im Filterstring eine Startkoordinate angegeben und eine Bounding-Box übergeben wird, verwendet der Service die Koordinate aus dem Filterstring und ignoriert die Bounding-Box. Die Verwendung der einzelnen Parameter ist optional. Wird ein Parameter nicht angegeben, wird ein Standardwert verwendet.

```
IsochroneNumber=3;StartTime=0800;EndTime=0845;StartCoordinate=8.836,47.23
```

Table 6 Beispiel eines korrekt formatierten Filterstrings für den Abruf von drei Isochronen zwischen 8.00 Uhr und 8.45 Uhr ab einem Startpunkt in der Nähe von Rapperswil

11.2.3 Fehlermeldungen des CIPT-Web service

Falls eine Exception bei der Verarbeitung einer Web-Anfrage vom Code auf dem Server ausgelöst wird, liefert der Server einen sogenannten Exception-Report zurück. Dieser Report ist ebenfalls vom OGC standardisiert. Jeder Report ist von einem bestimmten Typ und kann einen frei definierbaren Text enthalten, der weitere Informationen zum Fehler liefert. Der CIPT-Webservice benützt die folgenden vier Typen:

- OperationNotSupported
- MissingParameterValue
- InvalidParameterValue
- VersionNegotiationFailed

OperationNotSupported wird erstellt, wenn der Request-Parameter einen unpassenden Wert enthält. WPS und WFS unterstützen andere Werte.
MissingParameterValue zeigt an, dass ein obligatorischer Parameter gar nicht angegeben wurde. Welche Parameter tatsächlich angegeben werden müssen, ist vor allem vom Operationen-Typ (Request-Parameter) abhängig.
InvalidParameterValue zeigt an, dass irgendein Parameter (mit Ausnahme des Request-Parameters) einen unpassenden Wert hat. Insbesondere wird auch ein solcher Fehler geworfen, wenn der Filterstring beim WFS oder der DataInputs-Parameter beim WPS einen ungültigen Wert enthält (z.B. mehr als 60 Isochronen).
VersionNegotiationFailed wird erstellt, wenn eine nicht unterstützte Version angegeben wird.

11.3 Konfiguration

Einige Parameter des CIPT-Webservices lassen sich ganz einfach in der Web.config-Datei ändern ohne dass die ganze Anwendung neu kompiliert werden muss. Die Konfigurationsdatei befindet sich im Root-Verzeichnis des Webservers. Die untenstehende Tabelle zeigt die möglichen Parameter.

Parameter-Name	Wertebereich	Obligatorisch
dbConnectionString	Connectionstring (z.B. Server=localhost;Port=443;User Id=postgr;Password=secretPw;Database=postgres;)	Ja
walkingSpeedKmH	5	Nein
loadLineSectionQuery	SQL-Abfrage auf zugdat_view	Nein

Table 7 Mögliche Parameter für die Web.config-Konfigurationsdatei

11.4 Automatische Testverfahren

Unsere Unit-Tests sind auf drei Projekte aufgeteilt. Die beiden Projekte „TestCipt“ und „TestOgcGeoServer“ enthalten Unit-Tests im herkömmlichen Sinn und testen die entsprechenden Programmcode-Projekte durch direktes Einbinden und Ausführen ihrer Libraries.
Das dritte Testprojekt „TestWpsWfsSchemaValidation“ gibt sich als Client aus und testet direkt den Webserver, in dem es WPS- und WFS-Konforme Anfragen an den Server sendet. In den meisten Fällen antwortet der Server mit XML-Daten. Diese werden von den Testroutinen mit Hilfe von XSL auf ihre Gültigkeit überprüft. Dieses Testprojekt betrachtet den CIPT-Webservice somit als reine Blackbox.

11.5 Akzeptanz-Tests mit Quantum GIS

Die nachfolgenden Abbildungen zeigen den Ablauf eines Aufrufs des WFS-Services in Quantum GIS (Version 1.8.0). Für die Startkoordinate in der Umgebung von Rapperswil werden alle Orte gesucht, die man zwischen 07:55 Uhr und 08:45 Uhr erreichen kann, dargestellt mit 3 Isochronen.

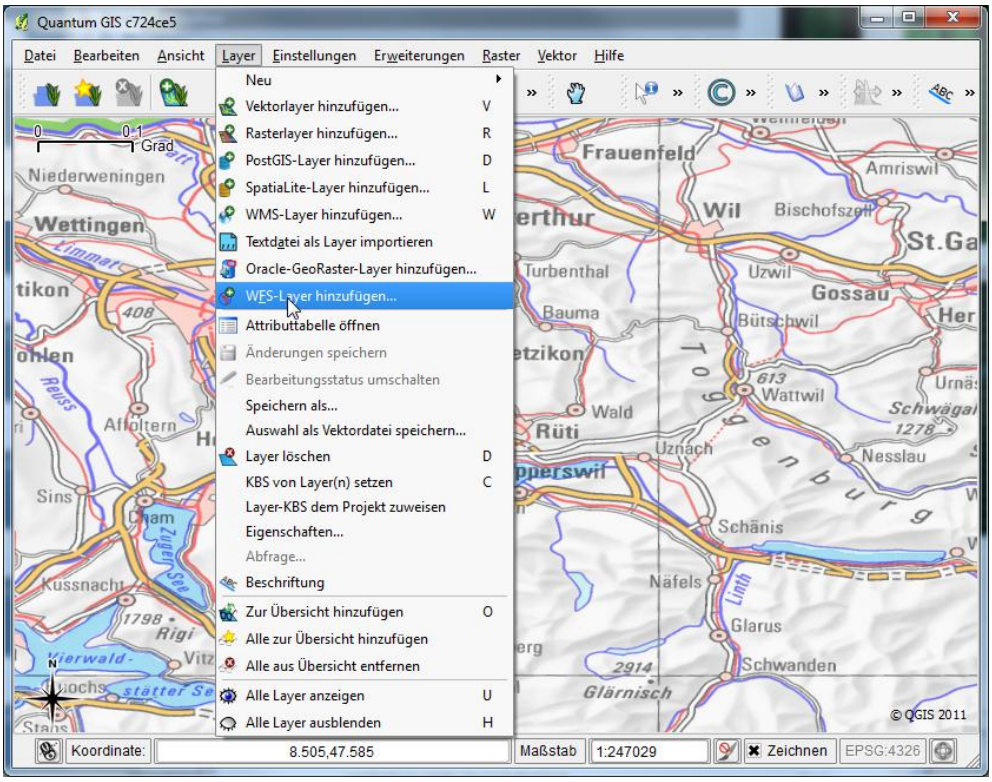


Figure 23 WFS Layer hinzufügen

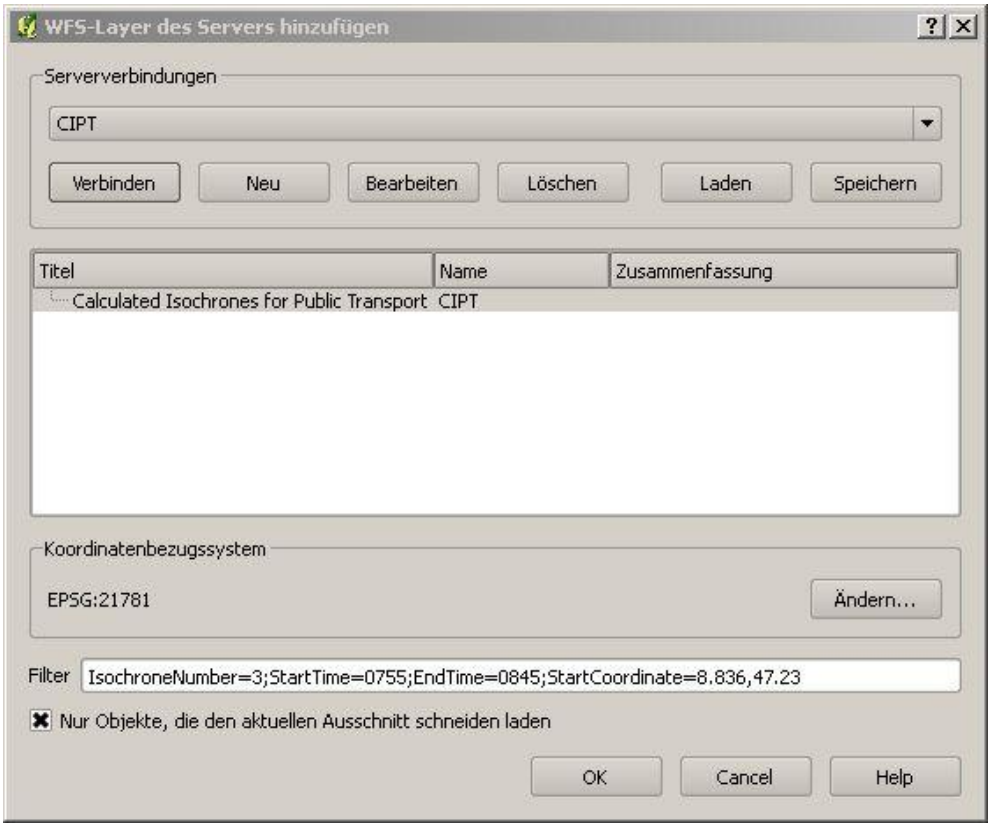


Figure 24 WFS Layer Dialogbox mit Anfrage an CIPT Web Service

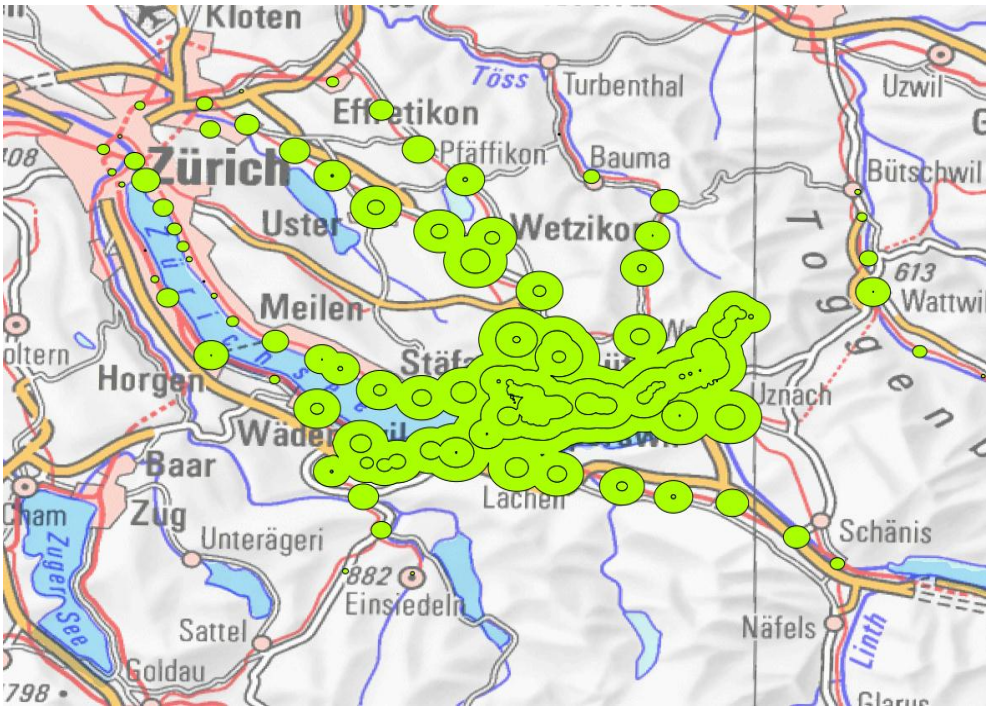


Figure 25 Isochronen als Antwort

12 Resultate und Weiterentwicklungen

12.1 Resultate

Siehe Kapitel 5.

12.2 Möglichkeiten der Weiterentwicklung

Eine Übersicht über mögliche Weiterentwicklungen ist im Kapitel 5.3 zu finden. Auf einige mögliche Weiterentwicklungen gehen folgende Unterkapitel ein. Dabei geht es insbesondere darum, aus der Machbarkeits-Analyse einen öffentlichen Service zur Verfügung zu stellen.

12.2.1 Integration in Website

Um die Darstellung der Isochronen unabhängig von Quantum GIS zu visualisieren, könnte eine Website erstellt werden, die eine GIS Funktionalität im Browser hat. Unsere Implementation bietet schon eine Ausgabe für Google Maps. Erweiterte Möglichkeiten, z.B. mit standardisierten Abfragen, sind denkbar.
Um den Service beispielsweise einem Job-Portal oder einer Immobilien-Plattform zugänglich zu machen, sind verschiedene weitere Abklärungen nötig.

12.2.2 Performance

12.2.2.1 Algorithmus

Im Rahmen einer Machbarkeits-Analyse und gemäss den Anforderungen soll die absolute Performance als sekundär betrachtet werden.
Weiterentwicklung, eine tiefergehende Untersuchung und Gegenüberstellung verschiedenster Algorithmen und neue Ansätze (z.B. mittels Heuristik) könnten entscheidende Leistungssteigerungen zu Tage fördern.

12.2.2.2 Parallelisierung

Pro Anfrage sollte ein eigener Thread gestartet werden, damit der Server nicht für neue Anfragen blockiert ist.
Es könnte untersucht werden, ob der Algorithmus an sich parallelisierbar ist. Neue Möglichkeiten zur Parallelisierung könnte eine andere Programmiersprache oder ein anderes Laufzeitsystem bringen.

12.2.2.3 Cloud-Computing

Eine bare-bone Leistungssteigerung mittels schnellerer Hardware könnte Cloud-Computing bringen. Die verfügbare Bandbreite des Netzwerks, auch allgemein, sollte nicht ausser Acht gelassen werden.

12.2.2.4 Pre-Processing

Der Fahrplan ist für eine Periode von einem Jahr gültig. Zudem ist er statisch. Man könnte alle erdenklichen Abfragen vorberechnen, so dass gar kein Algorithmus nötig ist. Um den nötigen Speicherverbrauch abzuschätzen, müssten weitere Untersuchungen angestellt werden.

12.2.2.5 Caching

Gleiche oder ähnliche Anfragen, die schon einmal gestellt wurden, könnten in einem Cache zwischengespeichert werden. Der Algorithmus müsste so nicht nochmals ausgeführt werden.

12.2.3 Datenbank

Anstelle einer relationalen Datenbank könnte man eine Graphen-Datenbank, z.B. Sones im .NET Umfeld verwenden.

12.2.4 WPS / WFS Server Erweiterung

Die OGC Dokumentation beschreibt noch weitere Möglichkeiten der Datenübergabe, z.B. mittels WSDL. Der Server könnte noch zusätzliche Protokolle implementieren.

12.3 Vorgehen

Folgende Weiterentwicklungen könnten bei einer Fortsetzungsarbeit in Erwägung gezogen werden:

- Parallelisierung
- Graphen Datenbank
- Einbau in Website
- WPS/WFS Server Erweiterung

13 Projektmanagement

13.1 Meilensteine

Folgende Meilensteine haben wir für das Projekt gesetzt:

Nr.	Woche-Nr.	Iteration	Inhalt
1	4	Planung & Infrastruktur	Einrichten virtueller Server, Datenbank, Entwicklungsumgebung, Zeitplan, Dokumentation
2	6	Architektur	Graph Library, Daten in Datenbank importiert, WPS implementiert
3	8	Algorithmus	Verschiedene Algorithmen evaluiert
4	10	Implementation	Ausgewählter Algorithmus implementiert
5	12	Optimierung	Algorithmus optimiert
6	14	Dokumentation	Abgabe Dokumentation

Table 8 Meilensteine

13.2 Team, Rollen und Verantwortlichkeiten

13.2.1 Team-Mitglieder

Name	E-Mail
Marco Birchler	m1birchl@hsr.ch
Kevin Lynn	klynn@hsr.ch

Table 9 Team-Mitglieder

13.2.2 Betreuung

Prof. Stefan Keller betreute das Projekt.

13.2.3 Verantwortlichkeiten

13.2.3.1 Planung & Infrastruktur Iteration

Marco Birchler	Kevin Lynn
Virtueller Server Webserver	Datenbank

13.2.3.2 Architektur Iteration

Marco Birchler	Kevin Lynn
WPS Implementation Import HAFAS	Graphen Bibliothek Test WPS

13.2.3.3 Algorithmus Iteration

Marco Birchler	Kevin Lynn
Evaluation Algorithmus	Evaluation Algorithmus

13.2.3.4 Implementation

Marco Birchler	Kevin Lynn
Algorithmus WFS und Test des WFS Polygon Generierung	Graphen Bibliothek Algorithmus

13.2.3.5 Optimierung Iteration

Marco Birchler	Kevin Lynn
Optimierung WPS	Refactoring

13.2.3.6 Dokumentation

Marco Birchler	Kevin Lynn
Arbeit am Bericht Entwicklung Test-Client	Arbeit am Bericht

13.3 Aufwandschätzung, Zeitplan, Projektplan

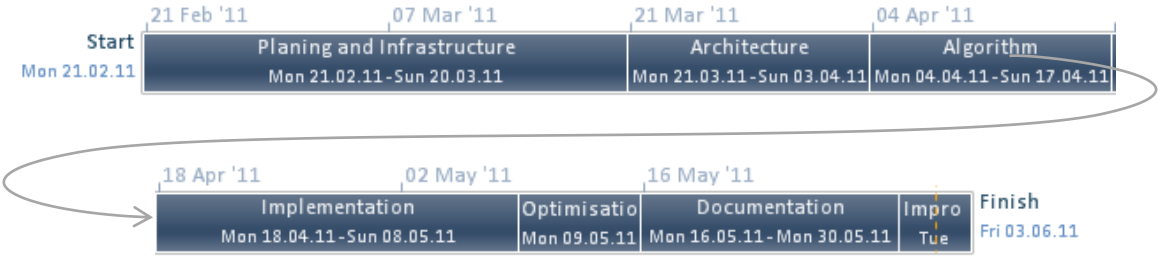


Figure 26 Übersicht Projektplan

13.4 Risiken

Risiko Analyse: Da das aktuelle Projekt keine Kosten im herkömmlichen Sinn verursacht, werden Arbeitstage als Kosten betrachtet.

Projektname: Calculate Isochrones for Public Transport (CIPT)

Projektmanager: Kevin Lynn / Marco Birchler

Datum letztes Update: 19.04.2011

Risiko Bewertungen									
Risk ID	Risiko	Auswirkung	Massnahme	Max. Schaden in Tagen	W'keit des Eintreffens	Gewichteter Schaden in Tagen	Priorität	W'keit des Eintreffens (aktuell)	Gewichteter Schaden effektiv in Tagen
R06	Es kann keine WPS Library für C# gefunden werden.	WPS-Dienst muss selber implementiert werden.	frühzeitiges Suchen nach Libraries	5	80%	4.0	1	100%	5.0
R04	Performance des optimalen Algorithmus ist zu schlecht für eine sinnvolle Nutzung des Dienstes.	Der WPS-Dienst kann nicht benutzt werden. Es muss das Polling-Feature von WPS implementiert werden.	frühzeitige Abschätzung der Parameter einer Abfrage: #DB-Zugriffe, Dauer eines Zugriffs usw.	5	40%	2.0	2	40%	2.0
R01	SBB/ZVV liefern keine Fahrplan-Daten	Es müssen manuell Daten erstellt werden. Attraktivität der ganzen Applikation sinkt erheblich	frühzeitige Kontaktaufnahme mit SBB und ZVV	5	40%	2.0	3	0%	0.0
R03	Fahrplandaten brauchen extrem viel Speicherplatz.	Virtueller Server genügt den Anforderungen nicht und es muss eine andere Lösung gefunden werden (evt.	frühzeitige Kontaktaufnahme mit SBB und ZVV	5	40%	2.0	4	0%	0.0
R08	Build-Lösung bereitet Probleme.	Eine reibungslose Entwicklung des Codes ist nicht mehr gewährleistet.	Ganzes Server-Image jeweils backupen.	3	60%	1.8	5	20%	0.6
R02	Gelieferte Daten sind in einem sehr unhandlichen Format	Es muss ein Importer programmiert werden, der die Daten nach Postgres kopiert.	frühzeitige Kontaktaufnahme mit SBB und ZVV	2	80%	1.6	6	80%	1.6
R07	Nur 3 offene Ports zum vServer in der HSR genügen nicht mehr.	Es kann nicht problemlos / auf direktem Weg auf den Server zugegriffen werden.	Lösung: Implementation eines SSH-Tunels	3	20%	0.6	7	20%	0.6
R05	Es kann kein funktionstüchtiger WPS-Client gefunden werden.	Das Testen des WPS wird erheblich erschwert.	Lösung: eigener Client programmieren.	4	10%	0.4	8	90%	3.6
	Total Rückstellungen					14			13

13.5 Prozessmodell

Der Entwicklungsprozess war iterativ, als Vorlage nahmen wir eine Adaption eines Scrum Templates.

14 Projektmonitoring

14.1 Soll-Ist-Zeit Vergleich

Für eine Studienarbeit sollte folgende Anzahl Stunden aufgewendet werden:

ECTS Punkte	Zeitaufwand
1 ECTS	25 – 30 h
8 ECTS Studienarbeit	240 h
16 ECTS für 2 Personen	480 h

Table 10 Zeitaufwand

Zeitvergleich:

Iteration / Artefakt	KW	Soll	Ist
Infrastruktur, Planung		**128 h**	**106 h**
	8		20 h
	9		30 h
	10		28 h
	11		28 h
Architektur		**64 h**	**60 h**
	12		30 h
	13		30 h
Algorithmus		**64 h**	**58 h**
	14		34 h
	15		24 h
Implementation		**64 h**	**74 h**
	16		38 h
	17		36 h
Optimierung		**64 h**	**74 h**
	18		36 h
	19		28 h
Dokumentation		**96 h**	**122h**
	20		32 h
	21		42 h
	22		48 h
Total		**480 h**	**484 h**

Table 11 Soll/Ist-Zeitvergleich

14.2 Probleme

14.2.1 Evaluation Graphen Bibliothek

In der Architektur Iteration begannen die Team-Mitglieder, diverse Graphen Bibliotheken zu evaluieren. Eine Auflistung evaluierter Bibliotheken ist in Kapitel 9.4.1 zu finden. Die Implementation und die darauffolgenden Tests, ob der Algorithmus die Problemstellung zufriedenstellend lösen kann, sind sehr stark abhängig von der gewählten Modellierung des Graphen.

Da beide Teammitglieder von anderen Standpunkten aus gingen, wurden Graphen und Algorithmen-Tests parallel entwickelt, in der Absicht, eine bestmögliche Lösung zu finden. M. Birchler setzte dabei auf eine Eigenentwicklung, K. Lynn versuchte einen Ansatz mittels bereits bestehender Graphen Bibliotheken und Algorithmen aus diversen Papers. K. Lynn konsultierte dazu auch die Meinung von Herrn Prof. Joller, der das Datenstrukturen- und Algorithmen-Modul an der HSR lehrt.
Die Parallelentwicklung dauerte bis Mitte Implementationsphase und nahm folglich viel Projektzeit in Anspruch.
Anlässlich des Meetings vom 4. Mai 2011 entschied man, die Optimierungs-Iteration zu kürzen und die verschiedenen Code-Teile anzupassen. Die lauffähige und zufriedenstellende Graph- und Algorithmus-Implementierung von M. Birchler wurde ab diesem Zeitpunkt weiterverfolgt. Alle Teile, die nicht zu einem Resultat geführt hatten, wurden fallengelassen und aus dem Repository gelöscht.

14.2.2 WPS mit Quantum GIS

Gemäss Aufgabenstellung sollte unser WPS Dienst standardkonform implementiert sein. Als Akzeptanztest sollte Quantum GIS benützt werden, das ein Plug-In zur Abfrage eines WPS Dienstes verwendet. Nach unserer Evaluation, siehe 9.4.4.1, lieferte das Plug-In für keinen der öffentlich verfügbaren WPS-Dienste ein Resultat. In der Annahme, dass diese auch standardkonform implementiert sind, wurde das Plug-In als unbrauchbar beurteilt. Dies führte zu einer Suche nach weiteren Desktop Client GIS Applikationen und weiteren Evaluationen, siehe 9.4.4.
Bis ein passender Client gefunden wurde, implementierte M. Birchler einen KML-Output für Google Maps, gewissermassen als Nebenprodukt, um unseren Dienst testen zu können und um die Resultate zu visualisieren.
Am 28. April 2011, in der Implementations-Iteration, entschieden der Betreuer und das Team, als zusätzliche Projekt-Anforderung einen WFS-Dienst zu implementieren um den Akzeptanztest wieder mit Quantum GIS durchführen zu können.

14.3 Codestatistik

Die untenstehende Tabelle gibt einen Überblick über einige statistische Werte des von uns erstellten Codes. Detaillierte Angaben können in einem Excel-Dokument auf der CD gefunden werden (/70_Codestatistik).

Project	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
Cipt	75	157	2	39	418
OgcGeoServer	81	142	2	33	250
TestCipt	67	25	3	18	150
TestOgcGeoServer	54	26	1	8	160
TestWpsWfsSchemaValidation	72	24	1	7	65

Table 12 Codestatistik ermittelt mit Visual Studio 2010, Stand 02.06.2011

Die Tests in TestCipt und TestOgcGeoServer decken ungefähr 90% des gesamten Codes von Cipt und OgcGeoServer ab.

14.4 Beschlussprotokolle

Es wird darauf verzichtet, die Beschlussprotokolle hier einzufügen. Die Sitzungsprotokolle sind auf der CD enthalten (CD/60_Projektplanung/Sitzungsprotokolle).

15 Softwaredokumentation

Die Softwaredokumentation ist auf der CD enthalten, zu finden unter (CD/40_Code-Dokumentation)

Appendix A: Figures

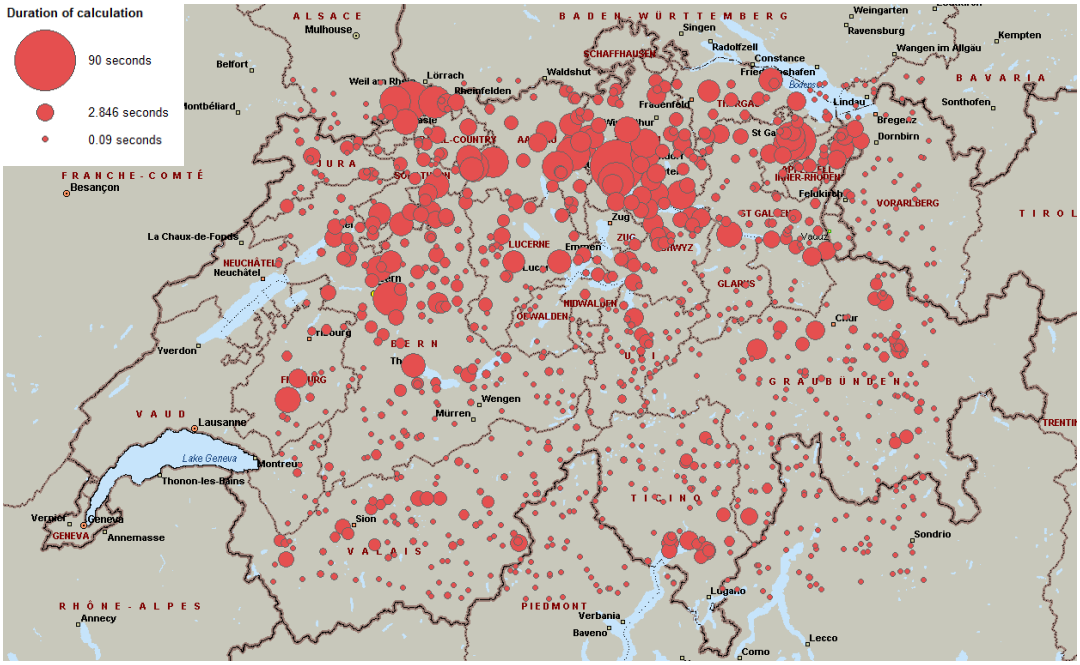


Figure 1 Grafische Darstellung der Algorithmus-Laufzeiten für eine Reisezeit von 60 Minuten und 60 Isochronen an verschiedenen zufällig gewählten Koordinatenpunkten. Durchmesser der Kreise ist logarithmisch abhängig von der Laufzeit. Quelle der Kartendaten: (31)

Appendix B: CD Content

Inhalt	Pfad
Dieses Dokument	10_Cipt.pdf
Poster	20_Poster_CIPT.pdf
Code (VS 2010 Solution)	30_Code
Code-Dokumentation	40_Code-Dokumentation
Kompiliertes Web-Projekt	50_CIPT-Webservice
Projektplanung	60_Projektplanung
Sitzungsprotokolle	60_Projektplanung/Sitzungsprotokolle
Detaillierte Code-Statistiken	70_Codestatistik
HAFAS-Datenimport	80_Daten-Aufbereitung
Fahrplan-Rohdaten	80_Daten-Aufbereitung/Fahrplandaten
Datenbank-Skripte	80_Daten-Aufbereitung/SQL-Skripte
Daten-Import-Tool	80_Daten-Aufbereitung/Fahrplan2PostgreSql Import-Tool
HAFAS Rohdatenbeschrieb	80_Daten_Aufbereitung/HAFAS-5.20.34_d.pdf
Performance Messung Auswertung	90_Performance_Messung/Auswertung.xlsx
Performance Messresultate	90_Performance_Messung/Messresultate
Performance Mess-Tool	90_Performance_Messung/Messtool
Performance Messdaten	90_Performance_Messung/Messresultate
Webseite	100_Webseite

Appendix D: Bibliography

1. *Computing Isochrones in Multi-Modal, Schedule-Based Transport Networks.* **Gamper, Johann.**
2. Broetlikrones. [Online] http://geosysin.iict.ch/trac/wiki/BroetliKrones.
3. HAFAS. [Online] [Cited: Mai 19, 2011.] http://www.hacon.de/hafas.
4. Quantum GIS. [Online] http://www.qgis.org.
5. **Pryga, Evangelina, et al., et al.** *Efficient Models for Timetable Information in Public Transport Systems.*
6. **Gerth Stølting, Brodal and Riko, Jakob.** *Time-dependent networks as models to achieve fast exact time-table queries.* 2002.
7. Open Geospatial Consortium, Inc. [Online] http://www.opengeospatial.org/.
8. HaCon Ingenieur GmbH. [Online] http://www.hacon.de/hafas.
9. **Dijkstra, Edsger W.** A note on the two problems in connections with graphs. *Nummerische Mathematik 1.* 1959.
10. **Prof. Dr. Ottmann, Thomas und Prof. Dr. Widmayer, Peter.** *Algorithmen und Datenstrukturen.* Heidelberg : Skeptrum, 2002. 3-8274-1029-0.
11. Relation:multipolygon. [Online] 28. 03 2011. [Zitat vom: 22. 05 2011.] http://wiki.openstreetmap.org/wiki/Relations/Multipolygon.
12. KML. [Online] http://en.wikipedia.org/wiki/Kml.
13. GML. [Online] http://en.wikipedia.org/wiki/Geography_Markup_Language.
14. Sones. [Online] http://www.sones.com.
15. OpenStreetMap. [Online] http://www.openstreetmap.org.
16. QuickGraph, Graph Data Structures And Algorithms for .Net. [Online] http://quickgraph.codeplex.com/.
17. The Boost Graph Library. [Online] http://www.boost.org/doc/libs/1_46_1/libs/graph/doc/index.html.
18. **Goodrich, Michael T. und Tamassia, Roberto.** *Data Structures & Algorithms in Java.* 2006. 0-471-73884-0.
19. [Online] [Zitat vom: 29. Mai 2011.] http://postgis.org.
20. nettopologysuite. [Online] [Zitat vom: 20. Mai 2011.] http://code.google.com/p/nettopologysuite/.
21. Microsoft SQL Server 2008 - Spatial Data. [Online] [Zitat vom: 30. Mai 2011.] http://www.microsoft.com/sqlserver/2008/en/us/spatial-data.aspx.
22. WPS auf GISPunkt. [Online] http://gis.hsr.ch/wiki/WPS.
23. Kappasys. [Online] http://www.kappasys.ch/cms/index.php?id=10.
24. OpenJUMP. [Online] http://www.openjump.org/.
25. MapWindow. [Online] http://www.mapwindow.org/.
26. **Michaelis, Christopher D., Ames, Daniel P.** Evaluation and Implementation of the OGC Web Processing Service for Use in Client-Side GIS. *Geoinformatica (2009).* 2007.
27. uDig. [Online] http://udig.refractions.net/.
28. **EPFL, U. Marti swisstopo / H. Dupraz.** Skripts für WGS84<->CH1903. [Online] Bundesamt für Landestopografie swisstopo. [Zitat vom: 25. April 2011.] http://www.swisstopo.admin.ch/internet/swisstopo/de/home/products/software/products/skripts.html.
29. OGC Web Processing Service Standard. [Online] http://www.opengeospatial.org/standards/wps.

30. **Caldwell, Douglas R.** Unlocking the Mysteries of the Bounding Box. [Online] 29 August 2005. [Cited: 19 Mai 2011.] http://purl.oclc.org/coordinates/a2.htm.
31. **1988–2009 Microsoft Corporation and/or its suppliers.** [Microsoft MapPoint 2010]
32. OGC Web Feature Service Standard. [Online] http://www.opengeospatial.org/standards/wfs.
33. SBB. [Online] http://www.sbb.ch.
34. ZVV. [Online] http://www.zvv.ch.
35. .NET Graph Library. [Online] http://graphlib.codeplex.com/.
36. plWPS. [Online] http://sourceforge.net/projects/plwps/.

Appendix E: Personal Reports

Marco Birchler

Die Arbeit an diesem Projekt hat mir im Grossen und Ganzen Spass gemacht. Vor allem hat es mich gefreut, dass wir das Projekt praktisch auf der grünen Wiese beginnen konnten und nicht an einem bereits bestehenden Gebilde etwas herumschrauben mussten. Somit waren immer wieder innovative Ideen gefragt.

Ein Problem einer solch offenen Ausgangslage kann natürlich auch sein, dass man zu wenig recherchiert und bestehendes ignoriert. In diesem Punkt war ich sehr froh um die Mitarbeit von Kevin, der sich stark mit den Grundlagen beschäftigte. Ich denke, dass wir uns deshalb gut ergänzt haben. Denn ohne Kevin hätte ich wohl einfach mal drauf los programmiert und das Ergebnis wäre nicht optimal herausgekommen.

Kevin Lynn

Das Thema GIS war für mich ganz neu und die Idee von Marco fand ich innovativ. Persönlich bin ich kein Freund des ÖV; die Möglichkeiten, so einen Dienst auf Basis des Fahrplans der SBB in ein Portal einbinden zu können, fand ich spannend.

Auf der Algorithmen-Seite hätte ich mir gerne etwas mehr Unterstützung gewünscht, die Suche nach einer Graphen Bibliothek war leider nicht genutzte Zeit. Allgemein ist zu sagen dass Open Source Projekte immer sehr schlecht dokumentiert sind, ohne die Unterstützung der HSR mit mindestens jemandem an der Schule, der sich damit auskennt und Fragen beantworten könnte, sind solche Projekte nicht innerhalb nützlicher Frist nutzbar.

Die GIS-Welt empfinde ich als Durcheinander von verschiedenen OSS Projekten, die alle ihre Stärken haben, aber unsauber abgegrenzt sind und bei der Suche nach Lösungen viel Zeit in Anspruch nehmen. Im Rahmen eines Semesters kann man nicht alle Projekte anschauen. Beim WPS Plug-In mit nur einem Entwickler haben wird die Abhängigkeit unterschätzt und mussten eine zusätzliche Anforderung einbringen, um unseren Dienst überhaupt nutzbar zu machen.

Es wäre gut gewesen, die Aufgabenstellung mit einer Lösungsstrategie schon zu Beginn zu haben; die ersten Wochen verbrachte ich mit PostGIS lernen, was wir gar nicht gebraucht haben.

Ich bin froh konnte ich mit Marco zusammenarbeiten, sein pragmatischer Ansatz und Einsatz haben einem ganz wesentlichen Teil zum Gelingen der Arbeit beigetragen. Ich fände es fair wenn Marco eine halbe Note zusätzlich zur Gesamtnote erhielte.

Appendix F: Declaration of Authorship

Erklärung

Ich erkläre hiermit,
- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.

Ort, Datum: Einsiedeln, 03.06.2011

Name, Unterschrift: Marco Birchler

Erklärung

Ich erkläre hiermit,
- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.

Ort, Datum: Jona, 03.06.2011

Name, Unterschrift: Kevin Lynn