DELFI

Documentation

Version 5.00

Report
on the current status of the
DELFI-system implementation

Contents by the Partners of DELFI
All German Federal States, the Federal Ministry of Transport, Germany and
the companies DB AG, HaCon, HBT, IVU, mdv

Edited by Stephan Schnittger
28 June 2006
Updated by Stefan Engelhardt (mdv)
01 July 2009

Contents

Figures

0 About this Document

This documentation summarises the state of the DELFI information system to show that potential new partners and their system providers are able to identify the advantages of the system and the efforts necessary to develop a system in order to be able to communicate with existing DELFI servers. This document follows the very formal guidelines according to the UML definition standard which are supplemented by figural explanations if necessary. In chapter 1, a short summary of the background will be given. In chapter 2, the general Use Cases and the mechanism of the communication flow is explained. The process of integrating metadata is described as well. Chapter 3 describes the methods, data structures and their usage in detail. In chapter 4, the instruction manual is given which describes the implementation of such a complex system and its possibilities to test the system. Additionally, this document includes a short glossary and the annexes of the IDL print as well as theoretical background.

DELFI's long history of developments is documented in a series of papers (see also Section 2.1). A particular basic document according to DELFI 2 was prepared in view of a research project. This document represents the state of development at this time and can be used as additional literature but not as valid paper for the current status. In order to understand the data description model of each system provider in Germany, the data model of time table exchanges defined in DELFI 2 had been developed. There is no direct implementation of this model. All data models necessary for the DELFI system are presented in this document.

This version of the document reflects the newest version 5 of the DELFI API. With this version, the API has been switched from a CORBA API interface based on an IDL to a SOAP Web Service based on a XSD/WSDL specification.

1 What is DELFI?

1.1 Overview

Maintenance of our mobility is a central must and challenge for our society. With motorized mobility, which is a precondition for the collaboration of areas, economic entities and persons, problematic consequences are associated which are not easy to cover especially with regard to the growing globalisation. The fundamental starting point is to find solutions to those resulting problems which are well known. Real solutions, however, partially exist but are difficult to realize under the circumstances within today's framework of global economy. A central and realizable starting point is to use information more efficiently than today in order to optimise, prioritise and control physical movement. It is especially important to provide top quality and updated information in order to reach the aim of efficient utilisation of all kinds of public means of transport.

With regard to the aforementioned problem, the improvement of the competitive situation of the public transport deserves special importance, even though improved information systems are implemented. Surveys of mobile people prove that only approximately 10% of the potential users of public transport are good enough informed about the supply of public transport and the possible itineraries. About 60% of the questioned people felt not to be or only badly informed. Just a third of the regular customers of public transport stated that they received enough information.

Figure 1: Itinerary information without DELFI: the different information of long-distance and local transport have to be selected and combined manually

According to the described problem, the Federal Ministry of Transport initiated the project DELFI in order to achieve the political aim of strengthening the public transport and, therefore, to manage the customers to overcome their inhibitions to use public transport. This is to be achieved by providing electronic travel information about all public means of transport at national level by establishing an independent travel planner.

The Federal Ministry of Transport pursues the following policies:

- the realisation of a national information system for the complete public transport,
- maintenance and preservation of market variety of different transport information systems,
- overcoming one's inhibition and
- providing contributions in the field of future transport information systems in Germany

Due to the brisk information market, a number of public and private enterprises compete against each other. The decentralised structure of DELFI creates high flexibility and gains momentum; see the example available in the Internet.

The supply of adequate partial information from the different single information systems enables the DELFI system to realize its service by combining and connecting all partial information. This leads to sustainability and eases the co-operation of the partners who partially compete on the market.

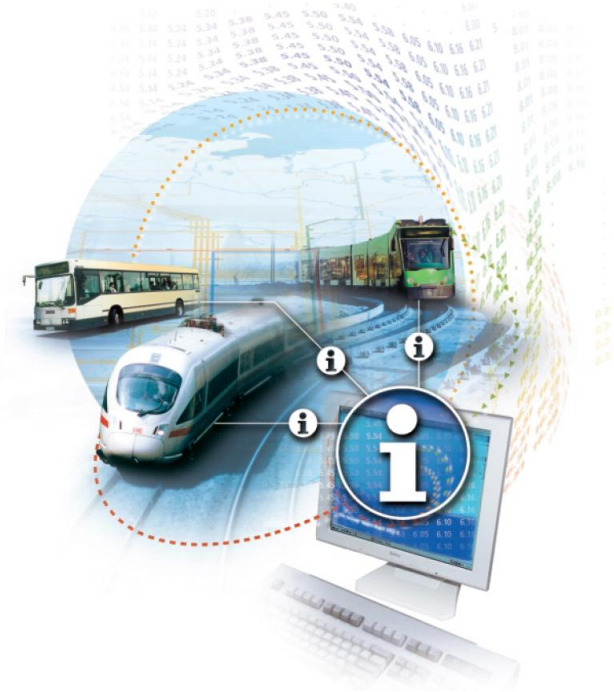

Figure 2: Itineray information via DELFI: necessary information will be combined with each other at the DELFI system and then provides a continuous itinerary for the customer

The following items emerged:

- Its objective is to realise a sustainable nationwide itinerary information system based on means of communication by using an Application Programming Interface (API).
- Harmonisation of database (no integration) is required
- It is an essential requirement for future telematics applications, especially at international level, to provide open interfaces. In that case, extended versions of electronic timetable information including international information data, integration of real time information, providing further services, and the extension of the project regarding new partners would be unproblematic.

1.2 Stakeholders of DELFI

All federal states of Germany must fulfil the necessary requirements in order to be able to find solution strategies.

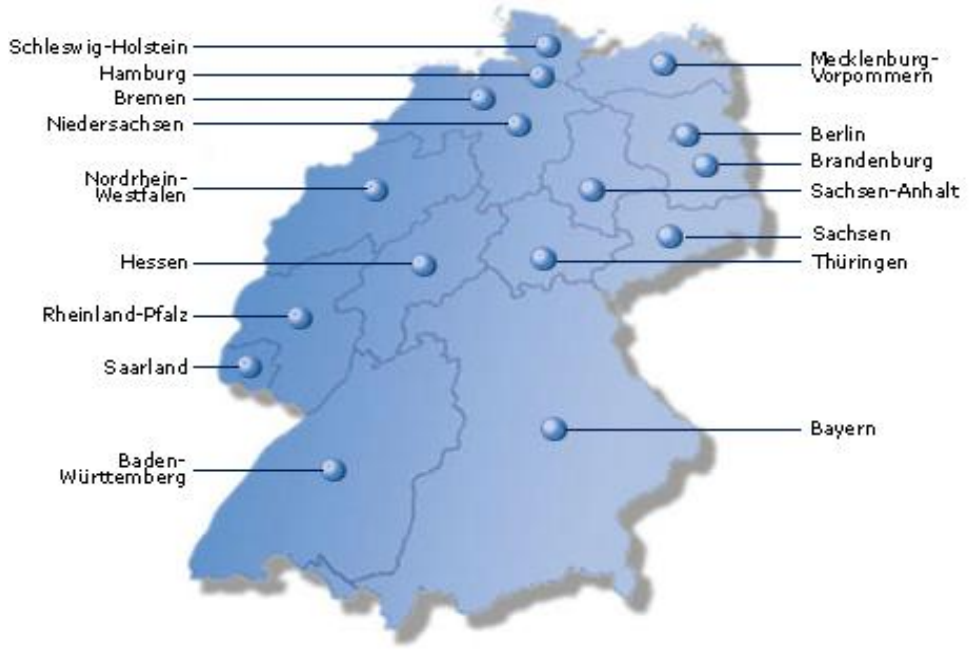

Figure 3: The partners of the DELFI information system: all federal states of Germany together act as the provider together with DB AG that provides the long-distance information

According to a call from the Federal Ministry of Transport, the Delfi partners formed the current work group. A steering committee was set up consisting of specialists from each German federal state. The Delfi group discussed possible solutions to provide continuous travel information nationwide very early in order to be prepared and organised by providing these solutions for the operation phase. Each partner entered into an unsolicited commitment to be responsible for the integration and maintenance of the timetable data within their region.

Furthermore, organisational and instrumental possibilities were discussed and evaluated in the latest project phases in order to ensure quality and stability of the

system.

All federal states of Germany act as provider together with DB AG that provides the long-distance data. Every partner provides organised timetable data integrated in its own system. In some cases, timetable data were integrated in a supra-regional system, for instance in Lower Saxony (Niedersachsen), Bremen and Hamburg.

1.3 Summary

The main contribution to the safeguarding of mobility aimed by the DELFI project consists of the basic requirements for an economic adjustment of the market and potential users who focus on intermodal countrywide linked transport information system. The adequate approach of a distributed open concept is based on systems which are coupled by means of communication; that is very suitable in order to reach the (difficult to reach) high grades of dependencies of information among each other.

The core of DELFI can be used for further developments by using a convenient algorithm for the distributed itinerary search:

- integration of supra-regional tariff information,
- possibilities to booking,
- taxi information and booking,
- integrated payment and ticketing,

The communication among independent information providers and their information system together with the combination of partial information to a continuous itinerary provide a complete new (high) quality of information for the customer. The solution by means of communication in this situation is found when the complete time table data exchange and pooling among all participants is economically not arguable or not desired. The idea of a solution by means of communication on the base of an interface between different servers leads to an architecture which

- allows a more flexible adoption of information server to existing constraints (i.e. scaling of information systems)
- allows the exchangeability of partial systems and by this a more fair competition (of system providers)
- allows the integration of new information servers on public transport (i.e. on European level) even as the acquisition of new data in existing systems (extend-ability of information server)
- eases the adoption process to other modes of transport (i.e. motorized individual transport or air transport) and new functionalities (i.e. price information)
- eases the extension to real time information which is in most cases very close to the origin data source.

2 The Principle Behind

2.1 History

The central algorithmic problem with the distributed itinerary computation is the optimization of partial information from single systems, especially under the aspect of communication costs among the systems. On the other hand, the optimization criterion of customers has to be taken into account. General solutions for optimal itinerary searches as short path searches in directed graphs are well known for standard information systems with pooled time table data (see section 4.3).

The main questions of the distributed itinerary computations were:

- Which information of what kind has to be offered at the open interface of the single information systems which can be retrieved by the search controller (see section 2.2), in order to realise an optimised selection and connection of partial information?
- Which meta information is necessary and allows selection and combination of partial information by the search controller?

Basic theoretical investigations on these questions were done using graph theory (see section 4.3).

DELFI was realised in a series of Steps:

1. DELFI I: from 1994 to 1996, basic investigations into possible solutions for nationwide electronic time table information
2. DELFI II: from 1996 to 1997, a first design and general developments in a communication interface and a harmonised data base
3. DELFI III: from 1999 to 2000, a test of the possible solution by means of communication showed the performance and quality of the system
4. From 2001 to 2004, the development of the metadata base and the preparations into the introduction of the system

In 1994, the Federal Ministry of Transport (BMV) set up a project in the frame of a research program (FOPS) which will indicate the technical feasibility of nationwide electronic time table information (DELFI I). The evaluators draw a positive conclusion but also remark that detailed investigations from their position will be necessary. During this stage, the German federal states were not involved. Afterwards, the Federal Ministry set up the following project DELFI II including all federal states. Since the federal states join DELFI it became clear that it is not only quintessential to approve the technological feasibility but also to step forward to practical implementation because the information system of the DB AG and regional information systems of a series of other providers are online. Thus, the task was to connect the single systems for a continuous itinerary. The project started in 1996 analysing the actual situation and possibilities of transport information systems. The

target was to evaluate the most successful approaches for continuous itinerary information. It resulted in the fact that it will be developed on a basis of means of communication between single transport information systems. Beside this, a common model of time table data interpretation improves the data exchange between different information provider at which the communication model may have not enough performance and at which the effort for data exchange is legitimated.

In 1997/1998, a first design of the communication interface and the common data exchange model was developed and tested by a few providers. During this stage of the project, problems of the interface definition and algorithms were identified and solutions adopted. First requirements on connecting data (metadata) were defined. Based on the specifications of the first stage, a first implementation of the interface description was implemented and prototypically tested during the second phase. A detailed specification document was developed in order to approve the implemented interface and compare it with integrated data bases. The following project flow was established in parallel:

- Technical part: In the frame of the project „DELFI II", most German system providers for public transport information systems co-operated in order to find a feasible technical solution for chaining different systems. Requirements were: indiscriminating and open for others.
- Administrative part: The federal states together with the federal ministry and the DB AG aimed to define and fix the frame for responsibilities, organisations and financing. Therefore, a board of all partners (steering committee or DELFI board) was established.

DELFI II showed the desired result. It was pointed out that it is possible to realize the core information by the two technical approaches if the frame conditions are adequate:

- The already exercised integration of different time table data of different formats into a data pool for a single information system; this is called pooling. The resulting integration effort is proportional to the frequency and size of integrations.
- The open system of the distributed itinerary search (interface-solution); during a request covering more than one regional system, every system provides a partial itinerary which is combined by a super composer to a continuous itinerary information. A data pooling is herewith no longer necessary. Data can be more up to date and an expansion over boarders is possible without the necessity of data pooling in a single system.

In a series of sessions the members of the DELFI board agreed to further develop an interface solution. At the same time, an agreement about the organisational constitution and a start-up financing for the first three years was achieved. The target of the project and the principles of the co-operation, organisation and financing were constituted in a convention to which all German federal states and the DB AG started to join step by step.

After the technical basics had been developed via the project DELFI II, the Federal Ministry of Transport (BMVBS) was ready to start the following project – DELFI III – in 2001 which focused on the demonstration of the maturity of the system by a consortium of all partners of DELFI II.

A DELFI co-ordinator for all participants was established whose task is to ensure a collaboration. At the conference of transport ministers in 2000, the DB AG took up the job because the chairman of the board of the DB AG offered and thus promised to fulfil this function unlimited in time and at no charge for the federal states.

DELFI III had been successfully finished in June 2002. It was demonstrated at a test environment that the distributed itinerary computation performs well.

From 2001 to 2004, the metadata base was built up and training courses for local data administrators were carried out in order to guarantee a smooth system start. Additional system checks were made to integrate all German federal states.

Between July 2003 and May 2004, all partners had raised the quality and the performance of the system so that the distributed system is nearly equivalent to an integrated system. In the final test, which took place in the first half of 2004, 3398 randomly chosen start-destination requests had been performed resulting correctly. Another result of this phase was a forum where all developers and the responsible data administrators of the federals states can report detected problems or mistakes and follow the process of clearing.

In the frame of a public presentation of the Federal Ministry of Transport in Berlin, the official start for the productive DELFI information system was announced on 28 June 2004. Ever since, all federal states of Germany provide information via their Internet portals based on the DELFI technology. Partners are the transport associations and information systems of the federal states which integrated the DELFI-Technology into their regional system.

Hereby, the customer receives complete itinerary information which can be developed in two ways:

1. If start and destination is covered by the regional system the customer receives his information only via the regional system in a well known form.
2. If the active system discovers (using the town information of the metadata table MetaCity), a requested location that does not lie within the area covered, the DELFI technology will be used to request this location via other responsible local systems and the partial connection information will be computed. This is possible since 28 June 2004. The active system combines the partial trip information to the complete itinerary. This form of computation can realise real time information in an easier way compared to an integrated system (also called pooled system) that allows nationwide high standard information. The over all possibilities are dependent on the abilities of the single systems and can be different for start and destination (i.e. addresses are not in every system available). The customer receives an itinerary in any case.

To guarantee this high standard of quality and in order to correct identified errors, a quality management was set up which is managed by the DELFI co-ordinator. A long-lasting technology has yet been developed and all actors agreed about a consensus to ensure the continuous operation of DELFI. Therefore, the DELFI information system has been available overall in Germany since 28 June 2004 and is invisibly integrated into the Internet information portals of the German federal states.

2.2 The Architecture and Partners

The approach to a distributed information system is based on a co-operation of stand-alone systems of the federal states.

Figure 4 to Figure 7 show the schematic principle of the DELFI system architecture. In order to search for a connection of a query, the customer can look for a connection in any webportal, usually his regional transport company's webpage. The local system decides if the query can be answered by the local system or if to query the distributed system via the search controller. This controller communicates with the other partners' servers in order to retrieve the information. The metadata base knows which server provides the requested data.

In contrary to a solution where all timetable data has to be integrated into a single data base a DELFI data exchange is limited to the metadata base which contains only the responsibility and contact information which is much less changing than timetable data.

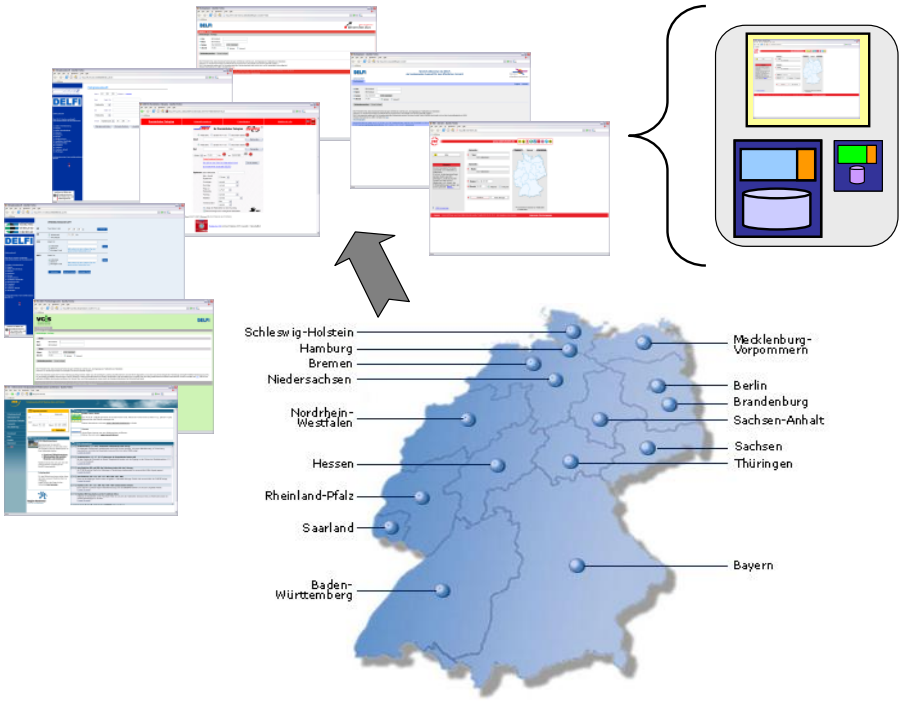

Figure 4: Overview: every German state has one or more information portals with its public transport information system

Every DELFI provider has to extend his normal information computation system with an interface which is able to provide partial itinerary information in a way which is well suited to be composed by the search controller (see Figure 5). The resulting component is called a passive DELFI system.

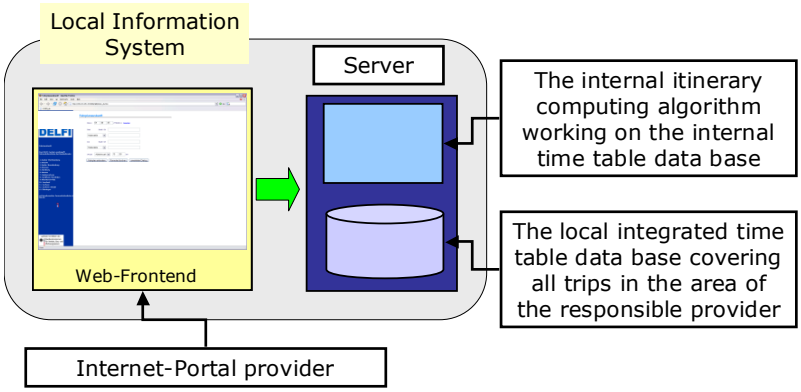

Figure 5: Overview: The principle of a stand-alone information server

In order to retrieve itinerary information from other passive DELFI servers, also called search controller, has to be implemented within the information server (see Figure 6).

The search controller is a kind of middleware between the front end web interface and other DELFI servers, communicating with them via DELFI-API. The search controller resolves complete continuous itineraries by means of communication from other DELFI servers. A DELFI server delivers partial itineraries and the search controller puts them together to complete itineraries and selects the optimal one. This result will be replied to the web interface to display it.

In order to perform the task, the search controller needs to provide metadata. The metadata is centrally hosted by a co-ordinator.

In Figure 6, the principle of a complete DELFI system is shown. The normal itinerary calculation component has to be extended with an interface which allows search controllers access to resolving locations and calculating connections. The search controller is a new component which has the task to decide about the appropriate server. The chosen server has to provide the data needed to answer the request so that complete and homogenous itineraries will be the result.

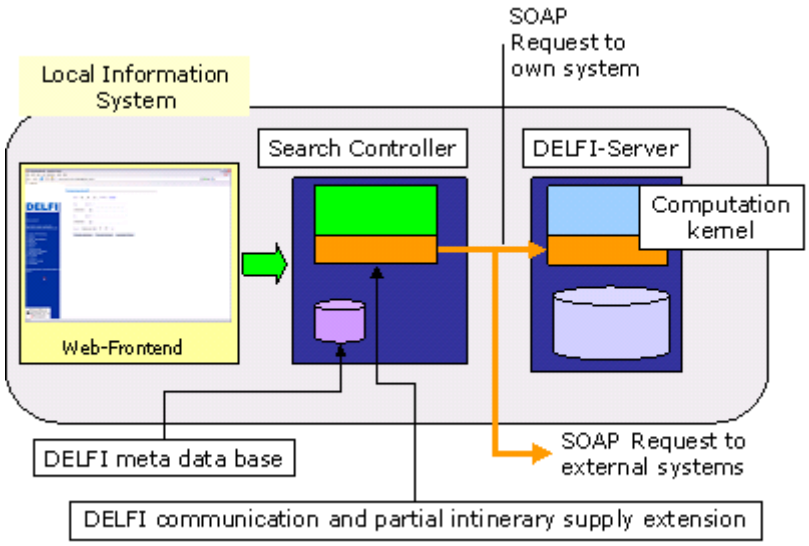

Figure 6: Overview: a complete DELFI system with a DELFI interface extended server and a search controller (active system)

NOTE: Instead of CORBA requests as shown in the figure, web service operations are called.

The communication flow between the local search controller and other servers necessary for the whole itinerary is shown in a diagram in Figure 7. The detailed description of this communication process can be found in section 2.3.

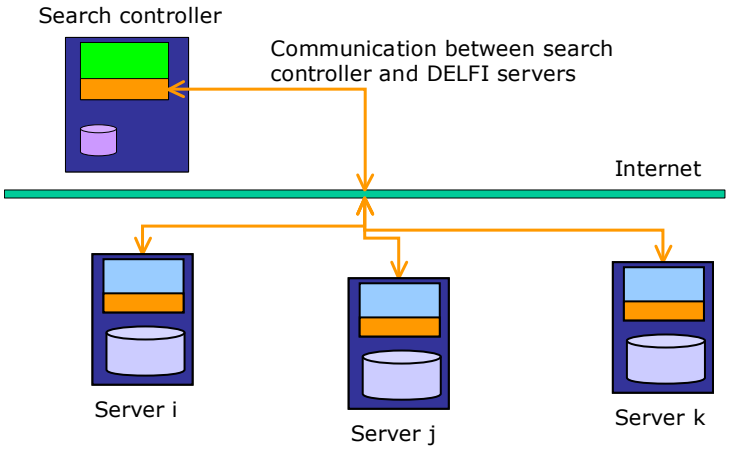

Figure 7: Communication via the DELFI-API extension between the search controller and passive servers using meta knowledge of the metadata base

According to the transport associations and DB AG, few requirements apply only to delivering partial information by the DELFI servers and to prepare and deliver information to the customer. In fact, the design of the existing servers within the DELFI co-operation does not need to be standardised. The partners only made arrangements for a few requirements regarding the use of the DELFI-communications manifested in the agreement.

The individual parts of the whole requested itinerary can be rated by the search controller but not interpreted. This means that a search controller can connect parts of the itinerary but is not allowed to filter specific elements. Especially the result can neither be shortened nor extended. The results of the individual parts of the whole itinerary should be connected by the search controller.

2.3 The Principle

The architecture is shown in the previous chapter. The following description is an abstract view on the general flow of communication between the three types of components existing in the DELFI architecture: We separate web front ends including any local component which provides information retrieved from the metadata base, search controllers (also called active servers) and servers (also called passive servers). The description in this chapter is a combination of abstracted flows and schematic figures. There are many possibilities of implementing the web interface and the search controller. Typically, the server is a stand-alone component based on a normal information server compiling itineraries from the time table data. The phases can be divided into two main elements:

1. the locations to be identified and matched with the appropriate server and
2. generating the connection of the itineray

NOTE:

The description of a function exists and is therefore available in two different models, at the implementation and thus at the description, for those flow parts of which the implementation of the *Locations()* operation at the search controller differs between a possible implementation and the one realised today in DELFI**.** That difference has no impact on the usage today because all search controllers are local components and are only accessible directly within a local system. These two flow possibilities are only documented in order to show the different possibilities of implementation.

In Figure 8, a formal diagram is shown which displays the first principle mechanism of interaction between the main types of components. This interaction diagram shows not the complete real detailed flow but illustrates the main mechanism of interaction between the base component types. All following figures illustrating the first flow type (principle flow) are named '.. possible flow ..', all figures illustrating the real implemented flow are named '.. real flow ..'.

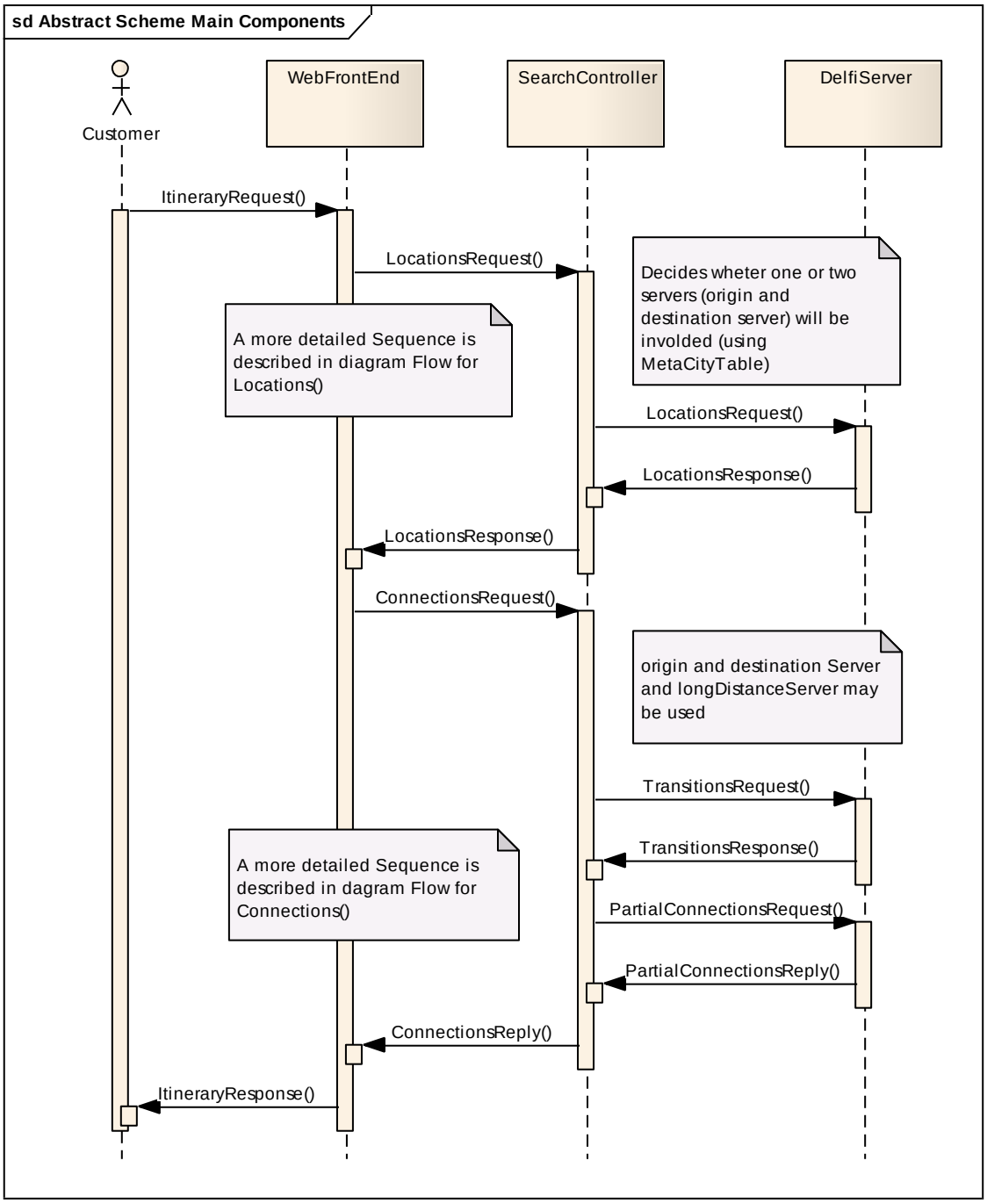

Figure 8: Scheme of possible flow between main types of components.

In Figure 8, the flow from the web front end to the search controller for the *Locations*() is shown which is dependent from the local implementation of the search controller. This form must be chosen whenever a search controller has to be usable for other web front ends.

In DELFI, all web front ends are directly connected with their own search controller. This fact and the fact that it is easier for the implementing partners to use their existing systems that extend their own algorithms for stop searches (depending on the system), led to the actual implemented flow shown in Figure 9.

The difference is that the first step of location resolving is done by another component and not by the search controller.

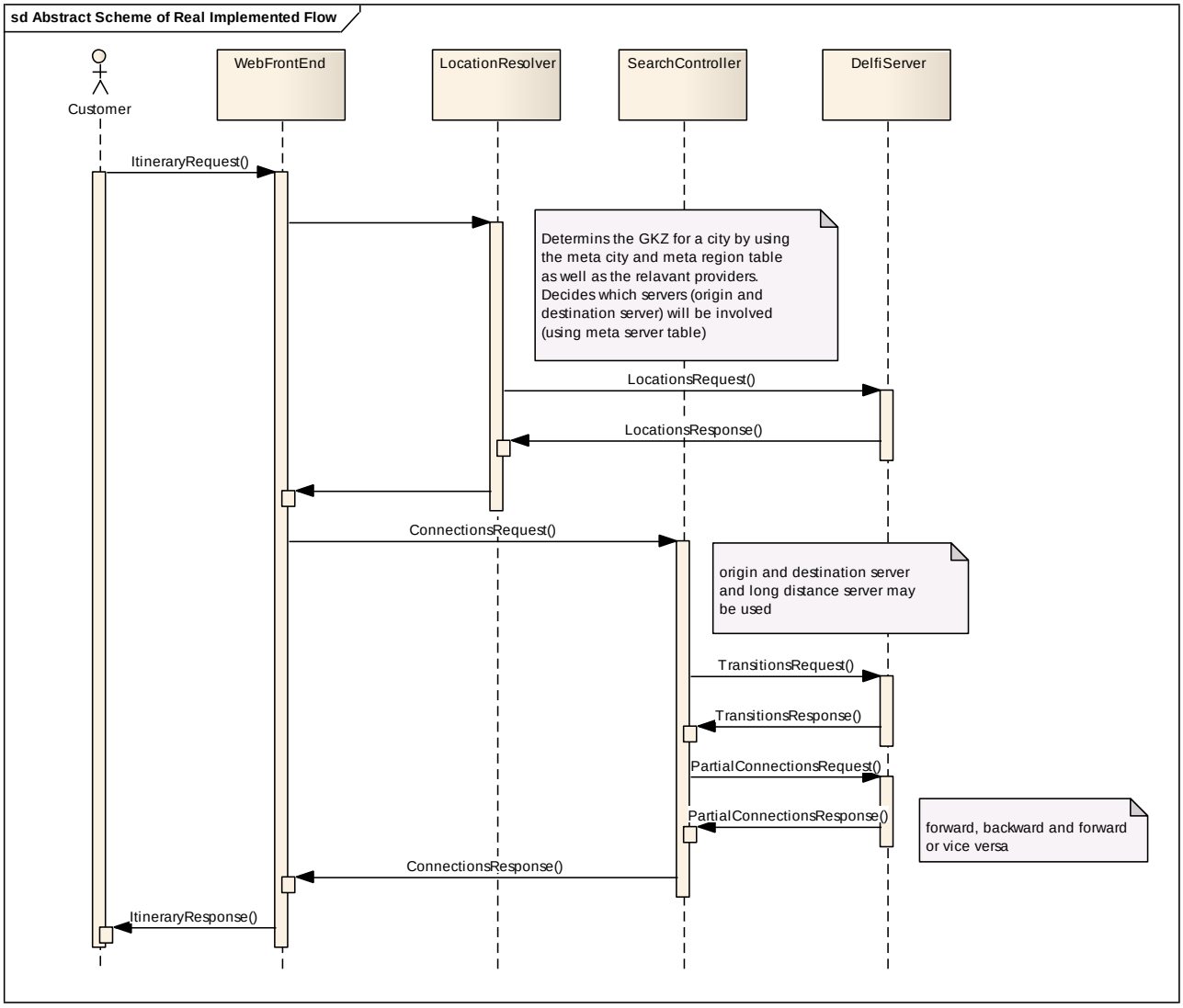

Figure 9: Scheme of real implemented flow between main types of components.

All parts of the flow will be described in the following sections.

2.3.1 Resolving and Identifying Locations

NOTE:

The following description illustrates a flow which indicates the pre-processing (metadata processing) in the search controller component. The real implementation differs in this point because the search controller looks at the real implemented pre-processing externally and expects identified cities. Therefore, they only have the task to send the request with the resolved city (with the unique identified city by the city ID, see metadata section 2.4) and the string for the stop or address to the responsible local server. In fact, this real implementation has no impact on other partners as long as every partner implemented this construction in its local system. Both

implementation forms will be illustrated for potential implementations.

Description of the task: A customer types a series of strings into the web interface in order to define the origin and destination of his trip, his start or arrival date and time (perhaps additional selections of products or attributes). This can be done in different ways but for DELFI it is necessary that the cities of origin and destination are named. The dialogue and the form of the presentation in general is not part of DELFI itself. Both, origin and destination have to be identified in a way that the connection calculation process can be done with individually identified locations. Stops or addresses are possible for a calculation.

A typical flow for the identification of a location as follows:

Flow elements:

In Figure 10, the input of the customer to the ***web front end*** (itineraryRequest) leads to the first call of a DELFI method: Locations (all methods are described in detail in section 3). There might be a lot of possible implementations (the web front end can handle more functionalities and the connection to an existing local server can be implemented in a way that the search is done first locally and only in case the location cannot be found, the DELFI mechanism will be contacted) but in this description we prefer a form where the components are separated from each other. Therefore, the controller is a separate component which is addressed from the part which sends the direct input/output to the customer, here, called the web front end.

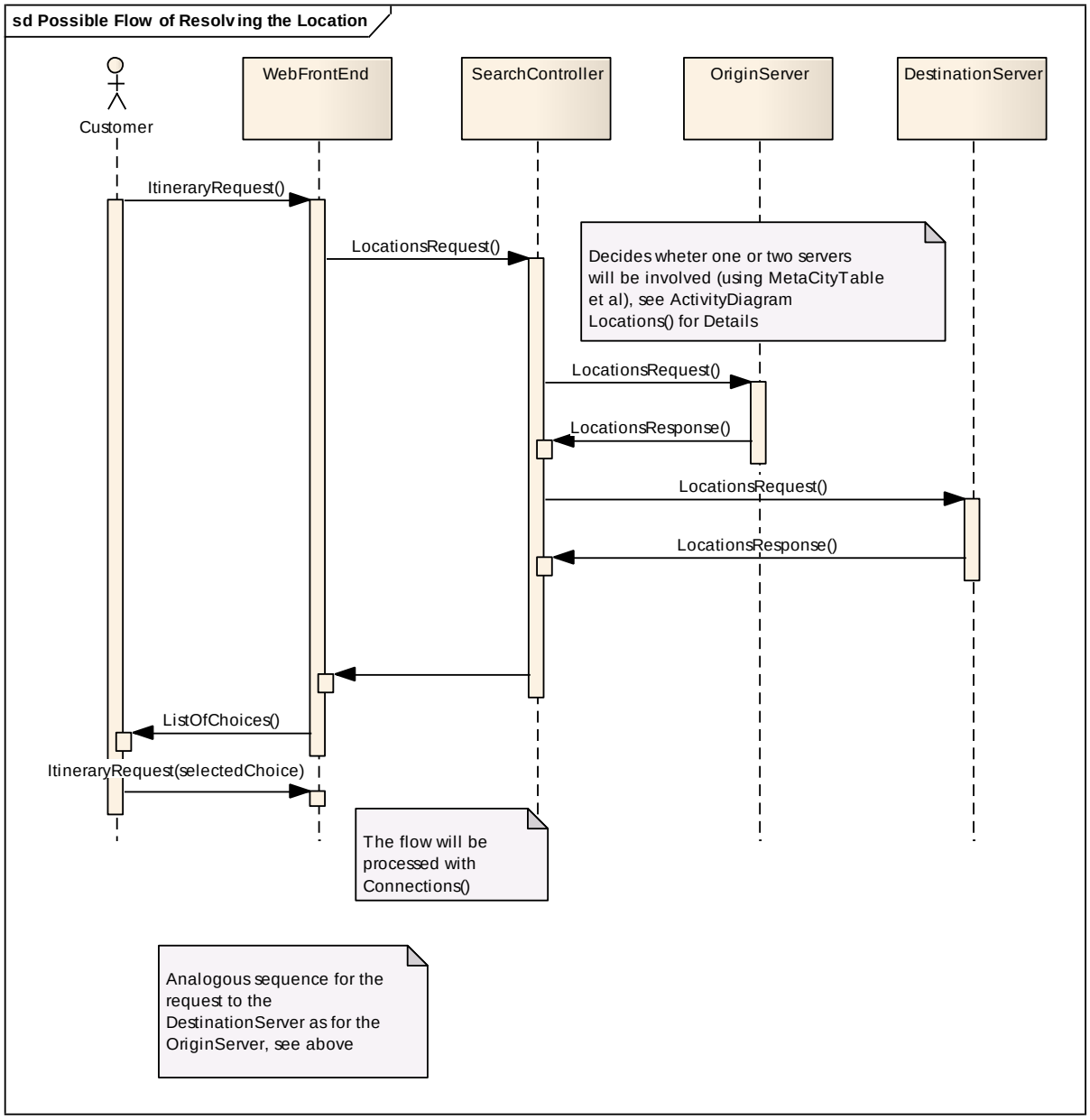

Figure 10: Possible flow of resolving the location so that an itinerary calculation is possible

The call of *Locations*() in the controller causes a flow shown in Figure 11. Both city names (origin and destination) will be matched with all city names in the table MetaCity of the metadata. Metadata are described in section 2.4. Here, it might be interesting that, besides the table MetaCity, two other tables named MetaRegion and MetaServer will be used. The resulting match of every city name input has one (in case of an explicit identification) or more possible entries or an empty resulting list. If there are more than one possible matches the customer will receive a list with the different possibilities in order to choose an appropriate one (for more details see chapter 3) or renew his request. The customer has, of course, also to renew his request if there are no results available. If there is only one match, the city-ID (AGS) of the city will be used in order to figure out the responsible provider for the city. Therefore, the table MetaRegion is to be used and the centre co-ordinates every AGS

from the table InfoRegion. In addition, the table MetaServer provides the actual valid server access data for every provider.

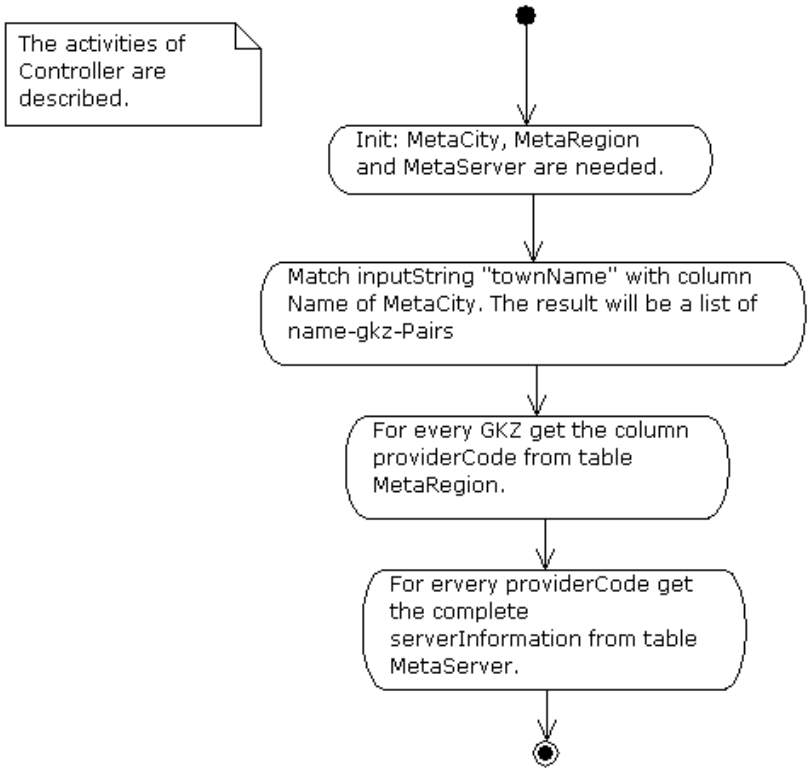

Figure 11: The flow illustrates the mechanism of server identification by the search controller or any other component using the string information of the origin and destination cities and the metadata tables.

If the responsible provider for origin differs from the provider responsible for the destination (see Figure 13), a distributed itinerary calculation is necessary.

The next step is to verify the input string by entering a stop or address name selected by the customer. The string will be sent to the responsible server for the location. This indicates a *Locations*() call to both of the identified servers but it is possible that the request of the customer, who defines a stop or address, cannot be resolved. In this case, possible choices of stops or addresses will be available for the customer in order to select one of the presented choices. The chosen one will then be sent again to the local server in order to receive the appropriate data to provide a connection for the customer's request.

The actual realised flow in DELFI differs in a few points from the presented flow above:

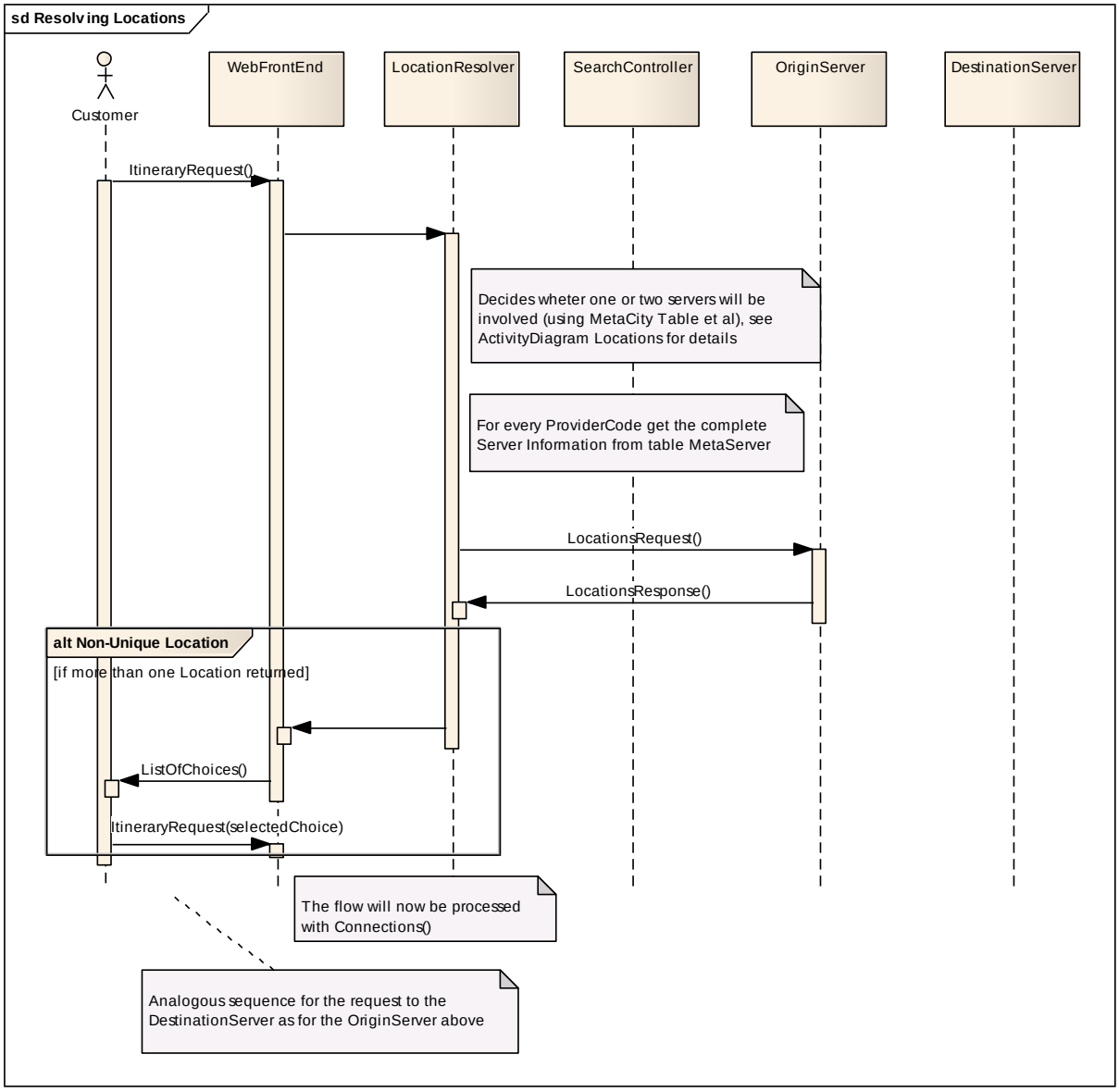

Figure 12: The flow currently implemented (real flow) for resolving locations - in order to be able to make an itinerary calculation

Flow elements in difference to the description above:

In Figure 12, the customer's request, e.g. a city name, is, at first, resolved and identified by a local component. Afterwards, the DELFI method Locations of the responsible passive server is called. The principle flow within the local component is the same as the one in Figure 11: Both city names (origin and destination) will be matched with all city names in the table MetaCity of the metadata. Here, the difference is that both resolving functionalities are splitted into the local city identifier and the itinerary calculation.

All passive DELFI server expect an identification of the city with its city-ID (AGS). Even the responding provider must be identified since the request will be transferred to his server.

The schematic illustration of the resolving process is shown in Figure 13. Whenever the customer makes a request for his or her itinerary, the request will first be led to the identifier and thus to the identifier of the responsible local server. The functionality is the same as described above but the resolving component is separated from the search controller. There, the resolved local city sends a request to the responsible passive server in order to even get the identification of the stop or address string.

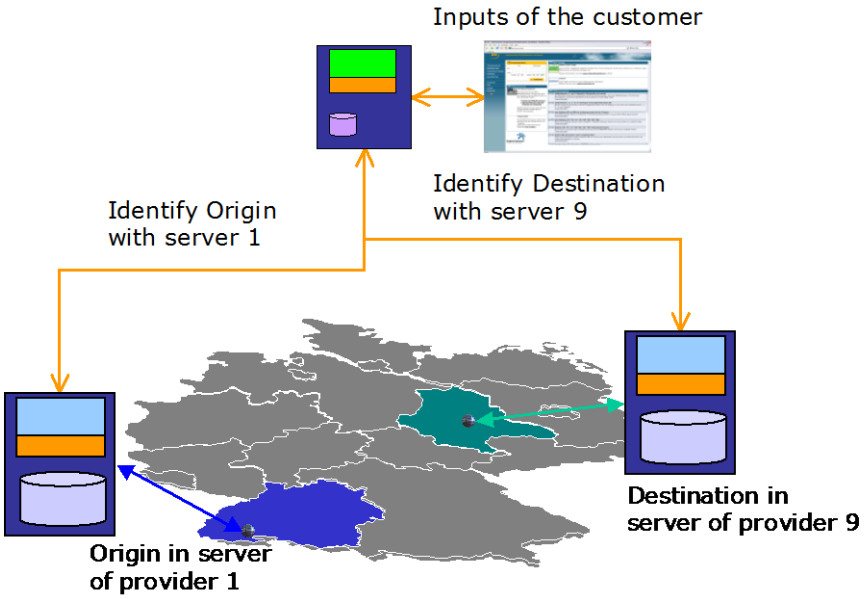

Figure 13: Identification of origin and destination with two different servers (distributed itinerary calculation)

If the identification of the location has been completed successfully, it will be proceeded without any further action of the customer.

2.3.2 Calculating the Connection

Besides the determined responsible servers for origin and destination, an additional server for the long-distance transport modes is necessary. The whole connection will be calculated by a maximum of three servers in the actual configuration. The special case that the complete connection can be computed by a single server for the situation that one server is responsible for origin and destination represents the usual situation of a stand-alone information server.

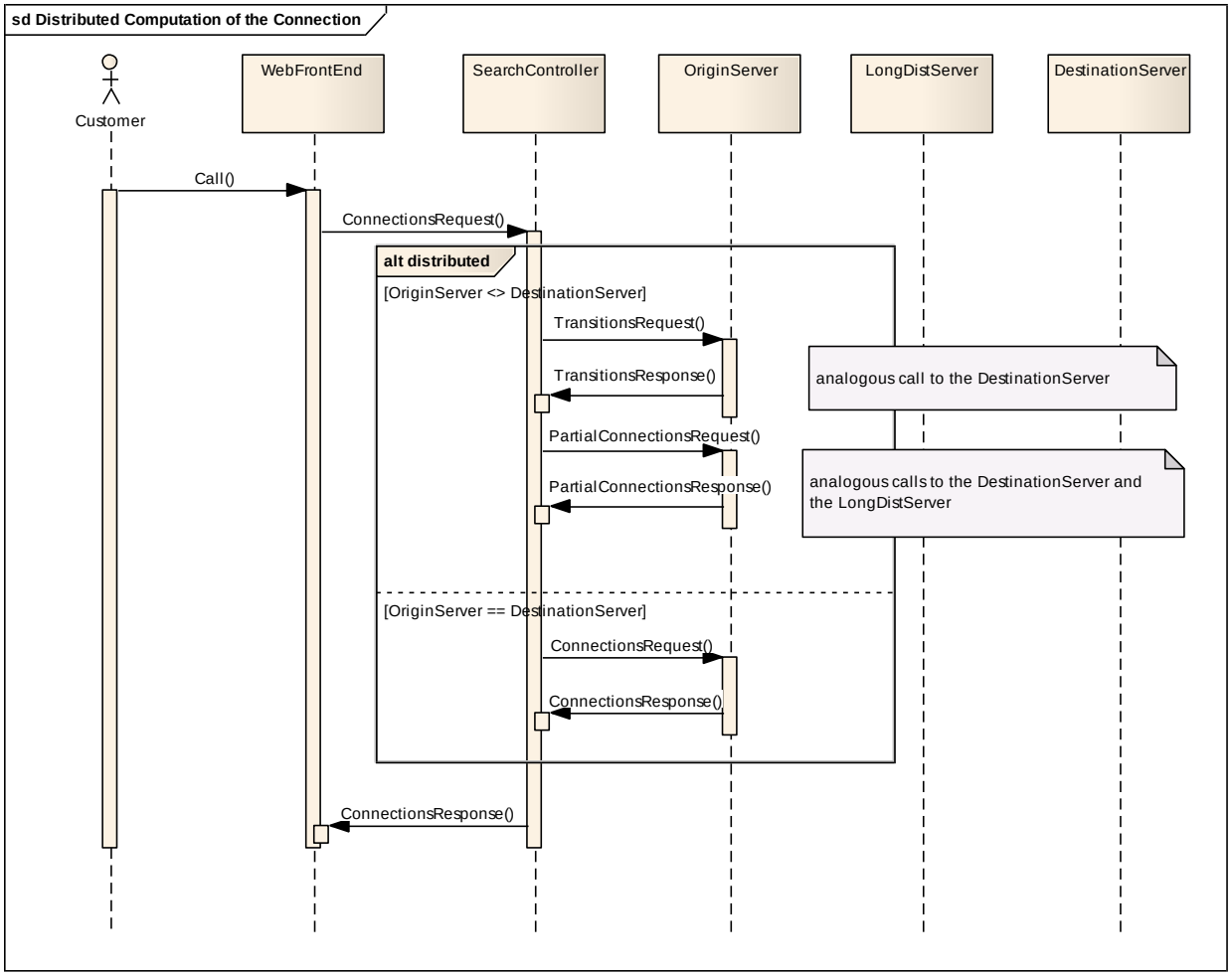

Figure 14: Next processing part: distributed computation of connection (forward)

The detailed technical description of the process is documented in chapter 3. Here, a description of the main principle is described schematically. The overview of the process is shown in Figure 14: The controller resolves and figures out the server responsible for origin and destination. If two different responsible servers result for the two locations, it leads to the configuration illustrated above: three different servers join the calculation. As one can be seen in Figure 14, the flow consists of four single activities (forward search = departure search)

1. Transitions
2. PartialConnections (calcConnectionsArrivalTimes)
3. PartialConnections (calcConnectionsDepartureTimes)
4. PartialConnections (calcRoutesArrivalTimes)

In Figure 15 (Transitions), the first step is illustrated: both origin and destination system deliver all or all relevant transition points to a Transitions-Request which is defined and agreed by the concerned data administrators (see Metadata Process).

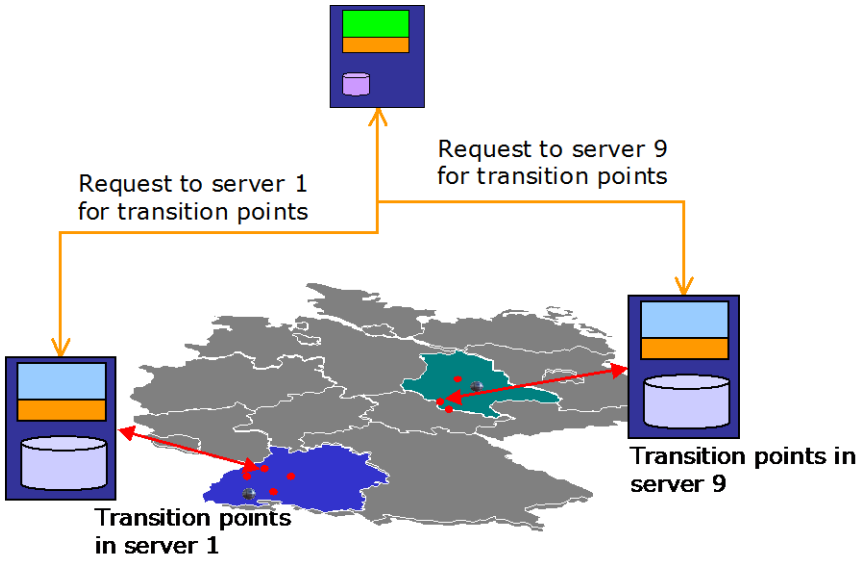

Figure 15: Request for transition points to the long-distance transport system

Those transition points can be main stations near the origin/destination location, or main stations of high relevance, or regional stations in border area of the local system. The result of the controller is a 1 (in case that one station only is relevant for the resolved location at origin), many combination (1:n connection computation case), and a m:1 combination for the same situation at the destination.

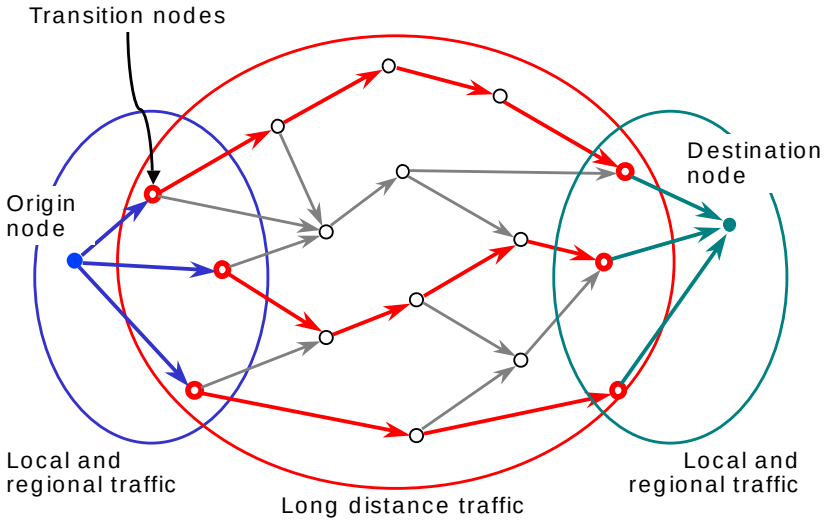

Figure 16: Spatial possibilities of routes from origin to destination; the calculation process is divided among the three servers and connected by the transition points

In Figure 16, the situation is shown more graphically. The overall itinerary results from partial routes of the origin system, the long-distance transport mode system, and the destination system. The first system (origin system) provides 1:n combinations for routes to be computed. The last system (destination system) provides m:1 combinations for the destination and at least the long transport mode system is able

to compute n:m different routes which is more theoretical. In real terms, the algorithms (i.e. Dijkstra) compute all arrivals from one start node as one phase so that only n calculations are necessary. The effort can be reduced to one calculation by using an algorithmic method if a pseudo start node is given with its arrival time points at the start nodes as an offset. For all calculations in a graph generated by the time table data, all necessary calculations can be done starting at a start node using the Dijkstra algorithm (see [20]). The proof that distributed itineraries can even be computed optimally is attached in section 4.3. In several cases, this calculation can even be done faster.

In Figure 16, the graph constructed from the time table data is illustrated in a spatial form. The given example starts at an origin including three transition points to the connecting long-distance transport system. On the right hand side, the destination system has also three transition points towards this long-distance transport system. Therefore, the origin system has to calculate three points (1:3) where the transitions' nodes can be obtained for three arrival times. The long-distance transport system has to calculate separately the (or one, if optimised via a pseudo node) arrival times of the end system from all three transition nodes in the origin system. The destination system must also calculate the arrival time beginning from the three (or also only one) now given computations of the long-distance system. There is a difference of this planar graph from the real graph which is discrete in time. Therefore, it is possible that a departure at a later time can be faster and gets, therefore, ahead of the found connection.

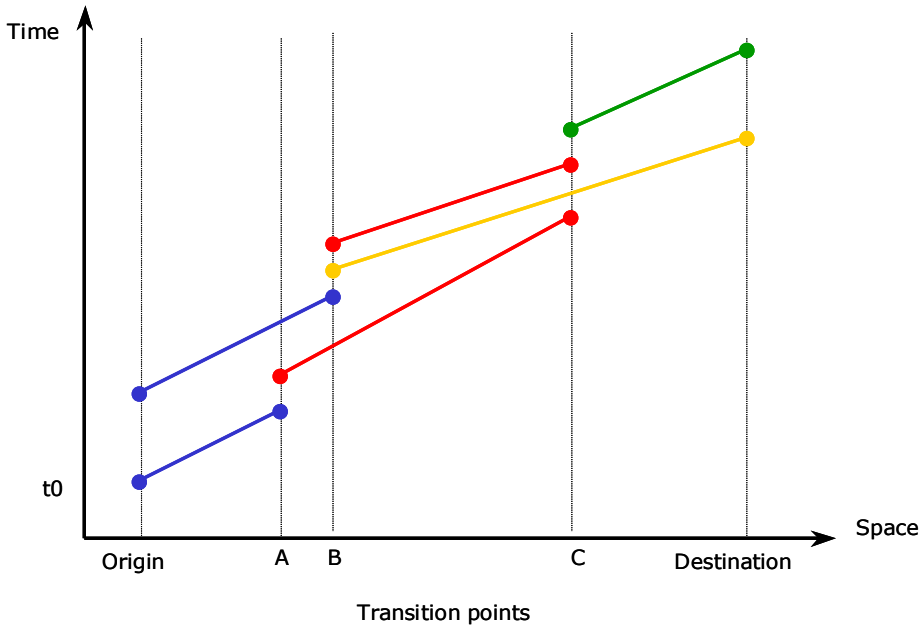

Figure 17: Overview: alternative routes with different travel times

The alternative(s) regarding date of arrival or departure can be found (illustrated in Figure 17, displayed in a time-space-diagram) if the computation is repeated in a revised order (Beginning at the destination at the calculated point in time backwards). All the partial connections for the final route are sampled in the last processing step. This led to the three steps usually named as search forward, search backward, and again search forward.

The first phase of requests *PartialConnections(calcConnectionsArrivalTimes)* determines in a sequential order, starting at the origin system (named system 1) via the long-distance transport mode system (named system 2) to the destination system (named system 3); starting from a given starting point in order of the arrival time for every target point (in system 1 to every transition point to system 2 and in system 2 from every transition point from system 1 to every transition point to system 3 and finally in system 3 from every transition point from system 2 to the destination point).

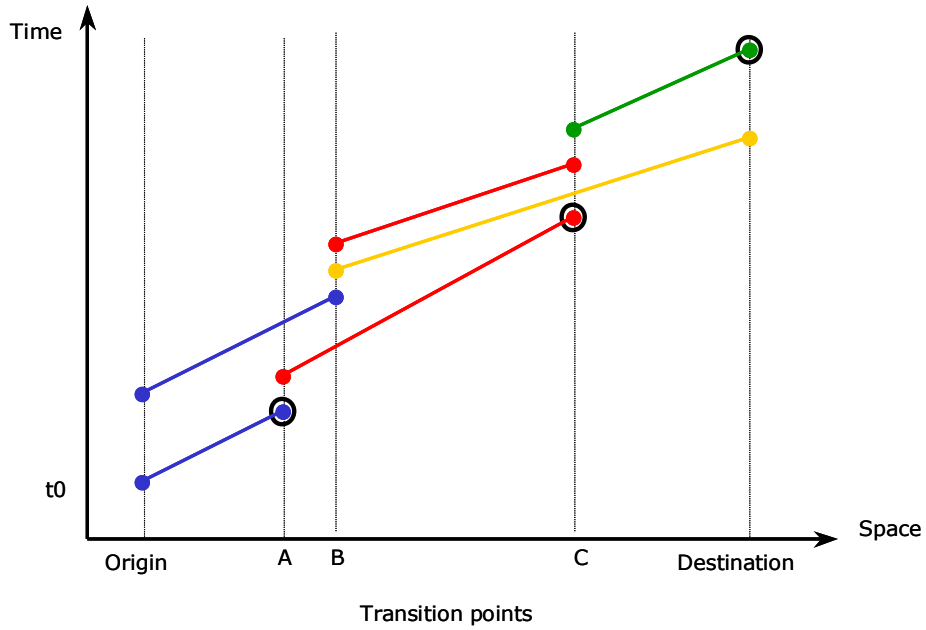

Figure 18: Phase 1: forward search, starting at origin (at given point in time) the fastest of all paths to all transition points will be searched (circles)

The main task of the operations

- *PartialConnections(calcConnectionsArrivalTimes) and*
- *PartialConnections (calcConnectionsDepartureTimes)*

is to determine the arrival time point at the destination node (respectively the departure time point at the origin node) and not the complete route information towards the node.

Due to the aformentioned description, a connection which starts later than the other one but arrives earlier at another transition point which allows a faster subsequent connection is possible. That path is calculated via the graphs when the *PartialConnections* operation is called but starting from the arrival time point backwards (searching the departure time). The sequence is performed in the vice versa order as the first *PartialConnections* computation.

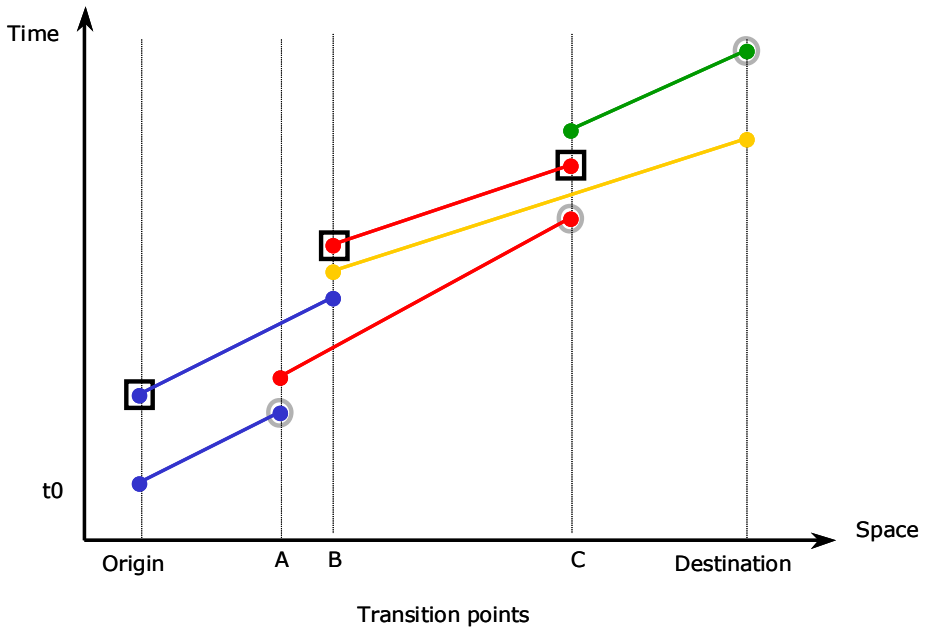

Figure 19: Phase 2: backward search, starting at destination at earliest arrival time backwards considering the fastest paths over all transition points to the origin (squares)

In the last step *PartialConnections(calcRoutesArrivalTimes)*, the process will be finished by means of establishing the route information by using the latest calculated departure time point at origin to the earliest possible arrival time point at destination. In this situation it can happen that a faster train passing the transition point C arrives earlier at the destination which then would be the fastest solution. In that case, the destination has to be included into the set of transition points, delivered by the third (destination) system.

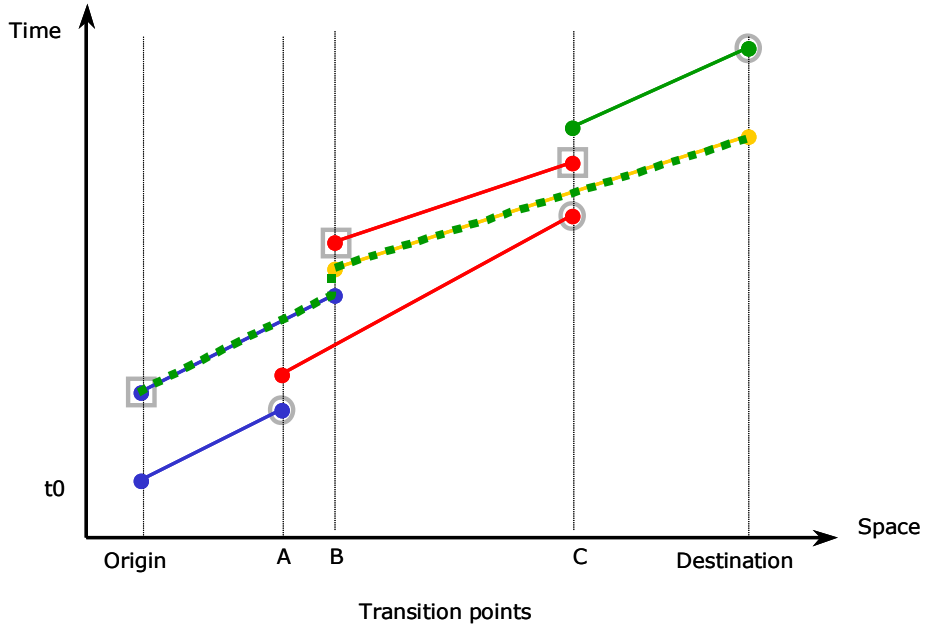

Figure 20: Phase 3: resulting fastest itinerary (dotted line)

In all cases, the necessary transition times in form of mode changes and/or walks are added by the responsible servers. The complete calculated itinerary might, therefore, not to be the only existing one. There can be several alternatives varying in optimization criteria. If so, it leads to a set of itineraries which has to be calculated (see Figure 21) and combined within the search controller.

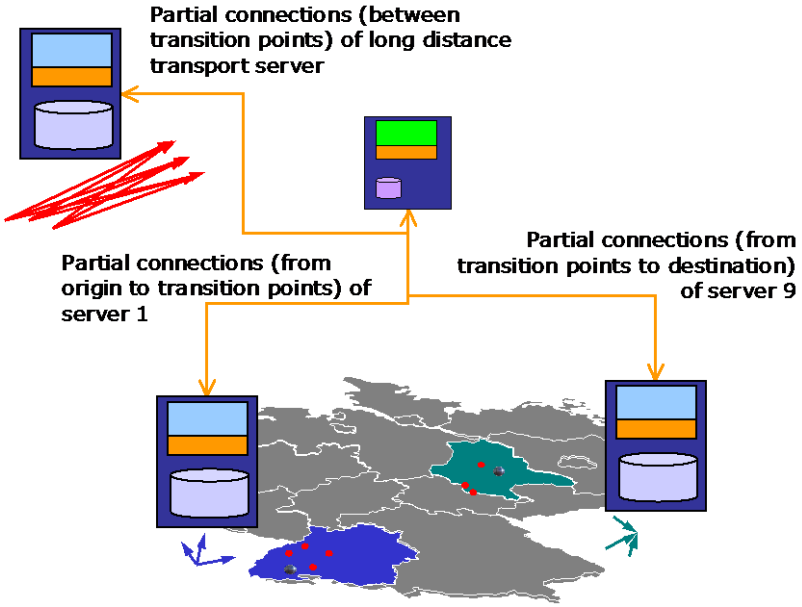

Figure 21: Overview: partial optimised itineraries as a result of the partial computation processes of the participating servers

Afterwards, all results are returned to the search controller. The search controller adds the partial itineraries to a continuous itinerary and presents the results on the web front end to the customer (see Figure 22).

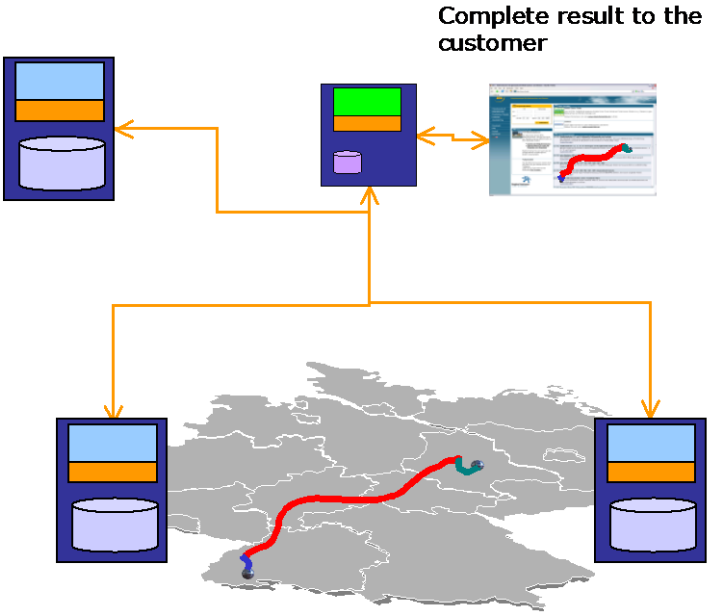

Figure 22: Complete return of itinerary information to the customer

The computation of alternatives is computed in the same way as the fastest connection.

2.4 Metadata

Metadata can be interpreted as the descriptive part of the organisation behind the distributed information system. The metadata of DELFI contain no time table data but all long time stable data which are necessary in order to identify the different responsibilities and to combine the different partial itineraries of the several systems.

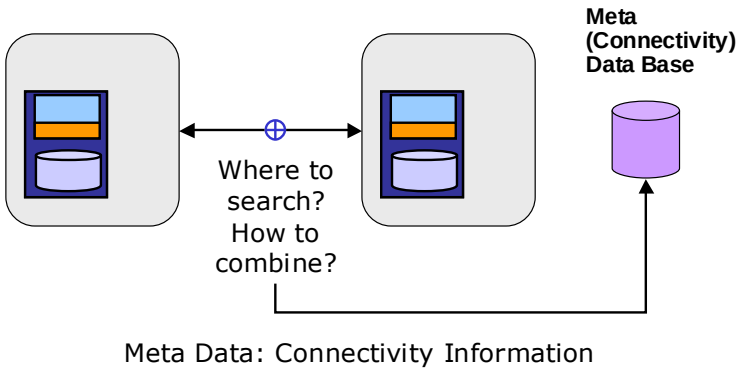

Figure 23: Overview: Principle of Metadata samples

Metadata (also called metainformation) allow search controllers to know where and how to access the DELFI servers. In Figure 23, the information which is based on metadata is illustrated. The information contains addresses of servers, spatial responsibility information of servers for cities and regions, and tables which allow this responsibility of origin and destination location to be resolved within the first step. Metadata can be hosted centrally or decentrally. In DELFI, the data of the partners is harmonised centrally which means that all DELFI partners provide their information for any other partners (see section 1.2).

In the following sections, the tables of this metadata base will at first be explained in order to explain the process of sampling this data afterwards.

2.4.1 Overview of the Metadata Base

The tables contain all necessary information mentioned in order to be able to identify the responsible servers and possible alternative servers. In section 2.3.1, a first view of how the metadata come into the process is illustrated. In Figure 24, an entity-relation-diagram of all tables in the metadata base is presented.

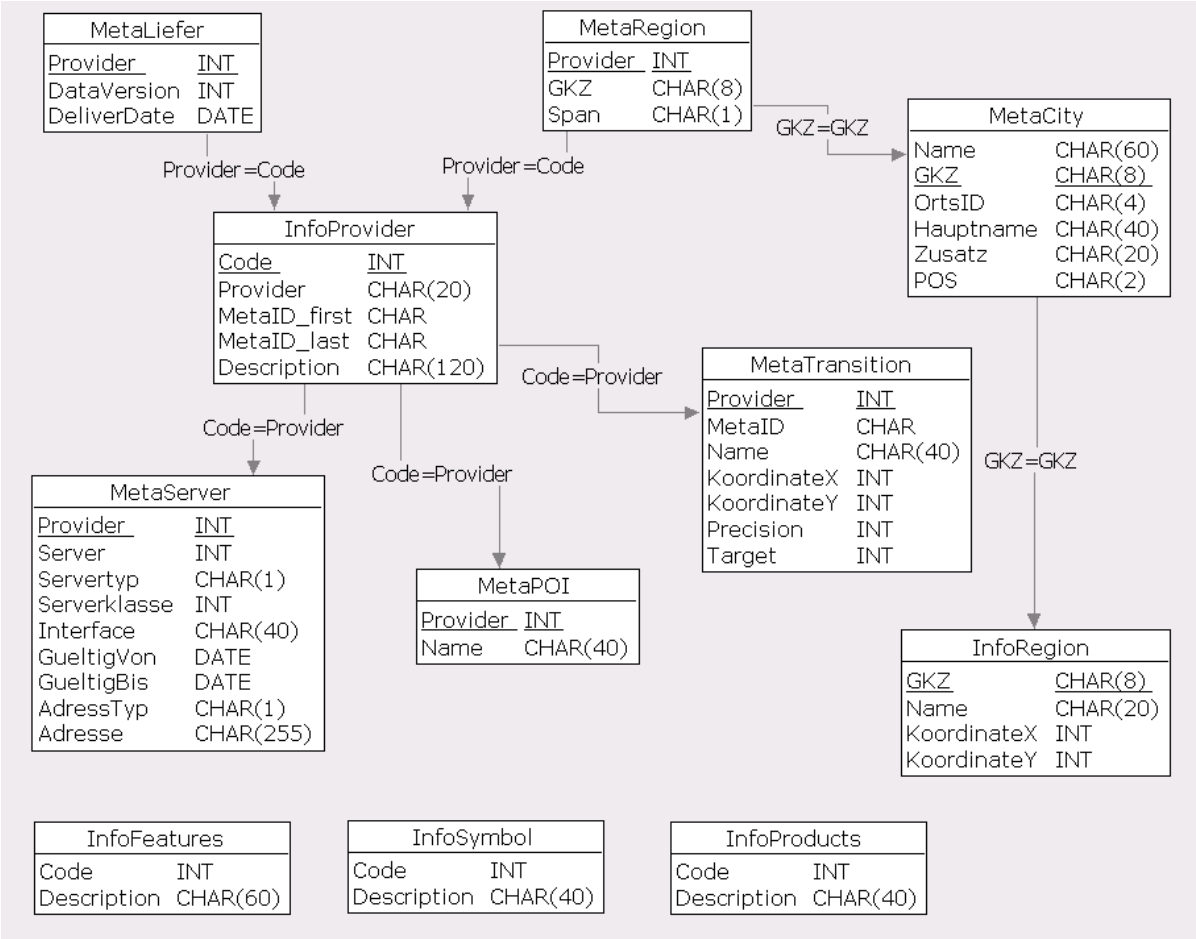

Figure 24: Overview of all tables of the metadata base

Main task of the metadata is the resolution of responsible servers for the input of the customer. For this purpose, the table 'MetaCity' is one of the main tables in the metadata base. This table contains all city and town names in Germany in an usual and completed form together with its town identification code (AGS). In Germany, all towns and cities are registered via a AGS which can be used as individual identifier. This identifier allows an explicit identification of the city. When the AGS is obtained and, thus, known the responsible providers for this city can be identified.

In real terms, time table data and corresponding stations are stored in several data bases on different providers; in areas where transport crosses different providers especially. Such means of transportation has a main responsible provider but is also included in the providers' data base into which that means of transportation moves. In order to differentiate between servers containing time table information of different detail levels, a qualifier value was developed. This qualifier is called span. Span is arranged in a hierarchy and is represented by the element 'Span' of the table 'MetaRegion'. Spans have the range of 0 to 9 whereas 9 represents the highest responsibility level. Other providers may act in this form as a fall back in case of problems. The result of the table 'MetaRegion' is a provider code which allows to select a server address in the table 'MetaServer'.

Most tables are standardised in order to provide a clean data holding. The tables are classified and therefore divided into the following types:

Type	Usage (Table contents)
Delivery	Information of data deliveries put in versions. Providers and the co-ordinator have their own history. Every provider is supposed to update his version for every delivery to the co-ordinator. The version history of the co-ordinator is related to the delivery of the complete metadata package to all providers (see description of metadata sampling process below).
Server	Information about provided servers and the kind of interfaces of a provider. This information is necessary if a server will be contacted.
Region	Contains area information related to the responsibility of the provider. The split up of areas is done by the German official identification system for cities (**A**mtlicher**G**emeinde**S**chlüssel = AGS). The whole area of Germany is split up into areas belonging to a city or municipality and covers the whole area. Every area is called region. Additionally, a table with coordinates for every AGS is available.
City and municipality names	Relationship of city/town/municipality names to regions. The city name strings of the customer inputs for origin and destination are the base for the selection of servers for the search of the final origin and destination (stop or address).
Point of Interest (POI)	Possibility of additional points of interests (famous buildings, airports…) which are often better known than the related town or stop name (this table is not used at the moment).
Transition	Transition points are stops which can be used for the interconnection of partial itineraries. Every transition point has an unique identifier, the MetaId, which can be defined in a predefined range for every data administrator according to the geographical responsibility.
Symbols, Features and Products	Symbols, features and products are used to identify the mode type codings of providers for a consistent interpretation and display. A provider defines the relation between his codings for mode types and attributes and the global defined symbols or codes. Every other code received as an answer of a server can only be shown in form of a text.

All tables will be explained in the following section, a usage example will be given at the end of this section.

2.4.2 MetaDeliver

The table MetaDeliver will be used in order to document the flow of the data update process (see description of this process in section 2.4.13). The table contains the data delivery information. The table will be filled depending on delivery direction (from provider to co-ordinator or from co-ordinator to provider). A provider delivers the table with one entry only: version and date. During the metadata integration process done by the co-ordinator (with deliveries of all providers) the date of provision of the complete integrated metadata base will be entered at first, followed by the version

data and dates of all providers (last states).

MetaLiefer		
Field	**Type**	**Description**
Provider	Int	Numeric code of the data Provider (see page 41); KEY
DataVersion	Int	Version of the delivered Data (starting with 1)
DeliverDate	Date	Date of Delivering; A string containing the date information in form of dd.mm.yyyy

Example

Provider	DataVersion	DeliverDate
219	8	06.03.2003
8	4	04.05.2004
15	12	02.06.2004
6	13	17.03.2005

The example shows the 8th version of data from provider 219 and the 4th one of provider 8 etc.

2.4.3 MetaServer

The table MetaServer contains the base data of the several single servers (passive systems) which provide information within the DELFI service.

MetaServer		
Field	**Type**	**Description**
Provider	Int	Numeric code of Provider of data (see page 41); KEY
Server	Int	Numeric identifier of the Server (unique number inside of the provider)
ServerTyp	Char(1)	Type of Server (A: Production System, T: Test System)
ServerKlasse	Int	Classification of the computing capability of the Server. For the classification numerical values in the range of 0-9 will be used. Currently, the following classes are defined: 1: < 1 request/sec 2: 1 - 5 requests/sec 3: 5 - 20 requests/sec 4: 20 - 100 requests/sec 5: > 100 requests/sec

Interface	Char(40)	Type of the interface (GetLocations, GetScheduleInfo, GetDistributedScheduleInfo), currently, this entry is not in a standardised format - therefore the entry must be interpreted by knowledge
GueltigVon	Date	Start date of accessibility of the server (format dd.mm.yyy)
GueltigBis	Date	End date of accessibility of the server (format dd.mm.yyy)
AdressTyp	Char(1)	I: IOR direct, U: URI connecting to IOR
Adresse	Char(255)	CORBA address of the server (IOR) or URI where the IOR is available

This table allows providers to host more than one server and especially to host production systems which should not be used for tests of any kind. Therefore, a test system may be set up and can be used in order to search for data problems or other failures. It can even be used for upgrades of functionality and improvements of software. Providers may also set up more than one production server in order to keep the overall system stable in case the main system crashes. Systems which are temporally available can be assigned with validity ranges (ValidFrom and ValidUntil).

Example

Provider	Server	ServerTyp	Serverklasse	Interface	GueltigVon	GueltigBis	AdressTyp	Adresse
8	1	A	3	DistributedScheduleInfo-3.7	01.01.2005	31.12.2500	I	IOR:010000002700000049444c3a44454c4649342
8	1	A	3	StartDestIdentification-3.7	01.01.2005	31.12.2500	I	IOR:010000002700000049444c3a44454c4649342
219	1	A	2	DistributedScheduleInfo	01.01.2005	31.12.2500	I	IOR:010000002700000049444c3a44454c4649342
219	1	A	2	StartDestIdentification	01.01.2005	31.12.2500	I	IOR:010000002700000049444c3a44454c4649342
15	1	A	2	DistributedScheduleInfo-3.6	01.01.2005	31.12.2500	I	IOR:010000002700000049444c3a44454c4649342
15	1	A	2	StartDestIdentification-3.6	01.01.2005	31.12.2500	I	IOR:010000002700000049444c3a44454c4649342
6	1	A	2	DistributedScheduleInfo-3.6	01.01.2005	31.12.2500	I	IOR:010000002700000049444c3a44454c4649342
6	1	A	2	StartDestIdentification-3.6	01.01.2005	31.12.2500	I	IOR:010000002700000049444c3a44454c4649342
13	1	A	3	StartDestIdentification-3.7	01.01.2005	31.12.2500	I	IOR:010000002700000049444c3a44454c4649342
13	1	A	3	DistributedScheduleInfo-3.7	01.01.2005	31.12.2500	I	IOR:010000002700000049444c3a44454c4649342

The table in this example shows not the complete IOR as it is normally in a size of 200 characters or more.

2.4.4 MetaRegion

The table MetaRegion contains information about all regions the provider is responsible for. The region is defined by the AGS. This is equivalent to the necessity to recognize all cities, stops and streets (if address data are available) in the defined region. The data for the long-distance traffic server (provided by DB AG) is not part of the file. Main stations are also covered by the regional providers.

MetaRegion		
Field	**Type**	**Comment**

Provider	Int	Numeric code of Provider of data; KEY
AGS	Char(8)	AGS number, official identifier code of cities and towns provided by the federal ministry of statistics
Span	Char(1)	Responsibility type (can be used in order to define redundant available providers (most common in overlapping regions), valid Span codes are: 9: complete 7: complete alternative 4: main transports / regional transports 2: minimal set of transports

Spans are predefined in the range from 0 to 9, whereas 9 and 7 denote complete knowledge of all public transport modes which are available in the area of the region. Span 4 denotes a partial knowledge of this provider. Most relevant public transport modes are known but not all of them. A span of 2 indicates only a rudimental knowledge of the public transport of a given city. This typically happens when transport providers have complete lines but a line also enters the area of another provider whereas the provider mentioned at first has only got its line in the city of the other provider.

All other span values are not defined at the moment.

Example

Provider	GKZ	Span
1	6431001	7
1	6431002	7
1	8116047	9
1	8116048	9
1	8116049	9
1	8116050	9
2	9174118	9
2	9174121	9
2	9174135	9
3	12068109	9
3	12068117	9

The example shows for provider 1: the provider is mainly responsible for the AGS 81xxxxx but he got also all time table data of cities of AGS 64xxxxx and can, therefore, act as a ‘second source’.

2.4.5 MetaCity

The table MetaCity contains information about the city names which has to be known by that component that resolves the city name explicitly (with or without additional interaction with the customer) in order to identify the cities. The resolved cities will be used in addition to the MetaRegion-table to identify the responsible passive server (or the alternative server).

MetaCity		
Field	**Type**	**Comment**
Name	Char(60)	The name of the city completed with OfficialName and Extension
AGS	Char(8)	AGS number, official identifier code of cities and towns provided by the federal ministry of statistics; KEY
OrtsID	Char(4)	Sub-identifier to the AGS for Suburban areas or Municipalities consisting of urban agglomeration (Cities)
Hauptname	Char(40)	Name of the city or municipality
Zusatz	Char(20)	Extension of name for correct identification in case of very common names
POS	Char(2)	Place Extension before "v" or after "n" the official name, "o" without brackets, "k" separated with comma, "n" or "v" are exclusive mandatory

Example

Name	GKZ	OrtsID	Hauptname	Zusatz	POS
Frankfurt (Main)	6412000	1	Frankfurt	(Main)	no
Frankfurt (Main)- Flughafen	6412000	11	Flughafen	Frankfurt (Main)	v
Frankfurt (Oder)	12053000	0	Frankfurt	Oder	n
Geesthacht, Stadt	1053032	0	Geesthacht	Stadt	nk
Gefell (Daun)	7233025	1	Gefell	Daun	n

The following scheme is valid for building complete names with the 'POS' element: The example is illustrated with the city name „Neustadt" and with the extension „an der Weinstraße":

POS	Comment
v	Extension is located in front of the OfficialName together with a dash and one space following. Example: an der Weinstraße- Neustadt
vo	Extension is located in front of the OfficialName without a dash and space, Example: an der Weinstraße Neustadt
n	Extension is located after the OfficialName and leaving one space and brackets, Example: Neustadt (an der Weinstraße)

no	Extension is located after the OfficialName and leaving a space but no brackets, Example: Neustadt an der Weinstraße
nk	Extension is located after the OfficialName separated with comma, Example: Neustadt, an der Weinstraße

The rules for naming are as follows:

- Exists only one municipality of an OfficialName (indicated with only one AGS), the name can be defined with or without extensions.
- Exists more than one municipality with the same OfficialName, none of them has to be defined without extension
- Only one primary responsibility (Span 9) is allowed to be defined for every AGS.

2.4.6 MetaPOI

MetaPOI		
Field	**Type**	**Comment**
Provider	Int	Numeric code of Provider of data; KEY
Name	Char(40)	Name of the Point of Interest

The table MetaPOI contains all information about specific locations, called point of interest (POI), i.e. airports which are well known but cannot be related to a city name or stop. A special example in Germany is the airport 'Köln-Bonn' which is neither belonging to the city of Köln nor to the city of Bonn. Other POIs which are only of local interest like hospitals have to be defined together with a city name and the detailed resolving has to be done via the responsible server.

Example

Provider	Name
13	Hotel Adlon
13	Konrad-Adenauer-Airport
13	Loreley

Hotels, other similar locations, and famous buildings can so be located in a direct way without knowing the name of the corresponding city.

The table MetaPOI is depreciated and will soon be removed from the Metadata Management system.

2.4.7 MetaTransition

MetaTransition		
Field	**Type**	**Comment**
Provider	Int	Numeric code of Provider of data; KEY
MetaID	Char(7)	Global unique ID of the transition (if the transition point is an element of the pool of the 219xxx, if it is a main station)
Name	Char(40)	Optional name of the transition point
KoordinateX	Int	Optional x coordinate geo-decimal (WGS84)
KoordinateY	Int	Optional y coordinate geo-decimal (WGS84)
Precision	Int	Precision of the coordinates
Target	Int	Addressed to… (Provider number of the provider who has to check this transition point)

The table MetaTransition contains information about transitions between different providers, mostly connected by long-distance traffic. At those transition points, partial routes can be connected. The concerned providers have to agree on the transition points. They have the necessary local knowledge in order to define the right point. For this process, the usage of the table is needed. The provider on which the point is located provides the meta number out of his meta number range for the point.

Transition points are modelled (see Figure 25) whereas the local provider is responsible for the transition time (walk) from the local transport stop (typically outside a station) to the platform of the long-distance means of transport. Therefore, the local information system provides that part in the computation.

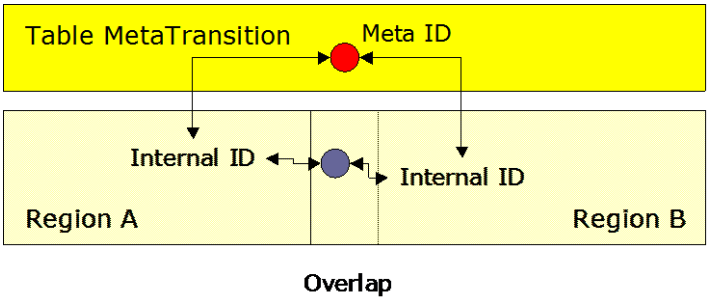

Figure 25: Scheme of meta coding transition points. The meta number has to be mapped internally on the internal stop number

A provider, who provides the transition point of his responsibility area, proposes a point to be a transition point connected with a meta number of his meta number range. All concerned providers (in border regions) have to agree and accept this transition point. Then, they must map this meta-ID on their internal numbering system.

Example

Provider	MetaID	Name	KoordinateX	KoordinateY	KoordinateZ	Precision	Target
2	100010	Karlstraße, Crailsheim	10073334	49137301	0	-1	
2	100012	Schulzentrum, Crailsheim	10073090	49141153	0	-1	
2	100013	ZOB, Crailsheim	10067279	49137493	0	-1	
2	100099	Ballmertshofen Bahnhof, Dischingen	10375031	48674594	0	-1	

2.4.8 InfoSymbols

The table InfoSymbols for information symbols harmonizes the visualisation and interpretation of transport modes. All different types of codes and identifiers for transport modes and types have to be presented on this symbol table and DELFI-servers have to use the defined symbol code.

InfoSymbols		
Field	**Type**	**Comment**
Code	Int	Code of the transport mode and types; KEY
Description	Char(20)	Description of the symbol, used for displaying it to the customer

Example

Code	Description
1	HGV-Verkehr
2	Fernverkehr
3	Nahverkehr
4	Spezialzug
10	U-Bahn
11	S-Bahn
12	Strassenbahn

The example shows the text which has to be displayed for given codes.

2.4.9 InfoRegion

InfoRegion		
Field	**Type**	**Comment**
AGS	Char(8)	Numeric Code of Provider of Data; KEY
Name	Char(20)	Optional name of the region
KoordinateX	Int	Optional x coordinate geo-decimal (WGS 84)
KoordianteY	Int	Optional y coordinate geo-decimal (WGS 84)

This table InfoRegion is an additional possibility to identify responsibilities. In the case that only co-ordinates are available (may be in situations where customers operate with maps or mobile digital assistants with GPS), the centre of the region has to be fixed in this table. It can be pictured in a AGS area and therefore onto a responsible provider.

Example

GKZ	Name	KoordinateX	KoordinateY
1051023	Dörpling	9302800	54260000
1051024	Eddelak	9138020	53946200
1051026	Eggstedt	9251520	54052000
1051027	Elpersbüttel	9024100	54064300
1051028	Epenwöhrden	9053110	54113000
1051030	Fedderingen	9132230	54282000
1051032	Frestedt	9175770	54023200

2.4.10 InfoProducts

InfoProducts		
Field	**Type**	**Comment**
Code	Int	Numeric Code of product type; KEY
Description	Char(40)	Text Description for the code

This table InfoProduct is similar to the table InfoSymbols. So called product types which are categories of transport modes in most cases with special characteristics (in terms of speed, comfort or price) can be separated and identified by these codes.

Example

Code	Description
0	Flug
1	ICE
2	IC/EC
3	IR
4	RE RB DNZ
5	Nachtzug
6	Schlafwagen-/Liegewagenzug
7	S-Bahn
8	U-Bahn
9	Straßenbahn
10	Stadtbahn

The example table shows typical values for such product types. Especially code 5 or 6 differ from normal trains.

2.4.11 InfoFeatures

InfoFeatures		
Field	**Type**	**Comment**
Code	Int	Numeric Code of feature type; KEY
Description	Char(60)	Text Description for the code

The table InfoFeatures is the third table for additional characteristics of the served mode. These features can be related to prices ("zuschlagsfrei" = without additional tax) or to handicapped ones ("barrierefrei" = without any barrier).

Example

Code	Description
0	Fahrradmitnahme
1	zuschlagfrei (Schnellbusse/Flughafentransfers)
2	barrierefrei (vollständig)
3	barrierefrei (bedingt)

2.4.12 InfoProviders

InfoProviders		
Field	**Type**	**Comment**
Code	Int	Numeric Code of the provider; KEY
Provider	Char(20)	Provider
MetaID_first	Char(7)	First number of the meta-ID range for this provider
MetaID_last	Char(7)	Last number of the meta-ID range for this provider
Description	Char(120)	Additional comments by the provider

This table InfoProvider is a main table of the metadata. Most other tables use the 'Code' of the table as key. This table contains all participating providers in DELFI. At the moment, providers of DELFI information are the German federal states.

Example

Code	Provider	MetaID_first	MetaID_last	Description
0	Metawissen	0	0	Delfi-Koordinator
1	Baden-Württemberg	100000	199999	
2	Bayern	200000	299999	
3	Berlin-Brandenburg	300000	399999	
4	Bremen	400000	499999	
5	Hamburg	500000	599999	
6	Hessen	600000	699999	
7	Mecklenburg-Vorpommern	700000	799999	
8	Nordrhein-Westfalen	800000	899999	
9	Niedersachsen	900000	999999	
10	Rheinland-Pfalz	1000000	1099999	
11	Saarland	1100000	1199999	
12	Sachsen-Anhalt	1200000	1299999	
13	Sachsen	1300000	1399999	
14	Schleswig-Holstein	1400000	1499999	
15	Thüringen	1500000	1599999	
219	DB	8000000	8099999	DB-Fernverkehr

This table is actual complete for Germany.

Usage example for the metadata tables:

A customer wants to travel from the city of 'Konstanz', stop 'Sternenplatz' in southern Germany to the city 'Euskirchen', stop 'Rathaus' in western Germany.

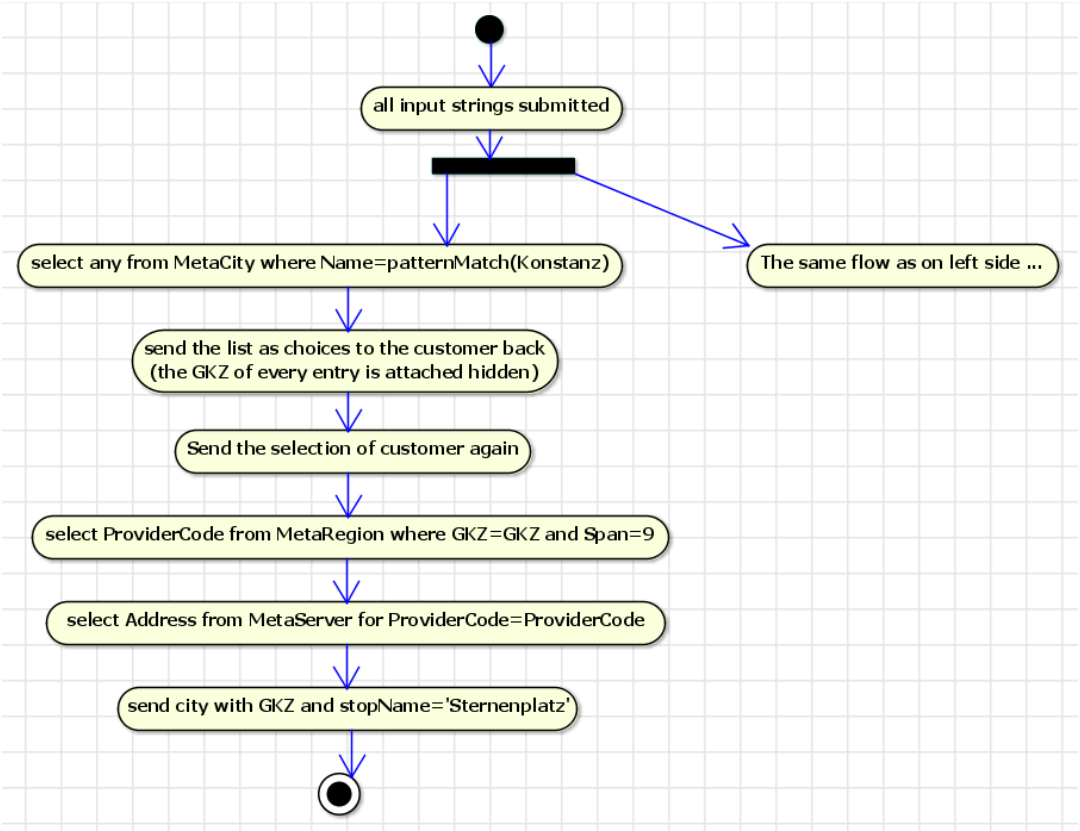

Figure 26: Example flow for using the metadata tables in order to resolve customers input strings for the identification of responsible local servers

The input strings of the customer are separated by the input fields of the screen into origin and destination. Therefore, the flow is also separated into a left and right part. The flow on the right hand side is the same as on the left. The string of the city field has to be matched with the values of the field 'Hauptname' and/or 'Name' of the table MetaCity. The result can be a set of matches between none and many. All results except an explicit match has to be sent back to the customer with a specific reply form (a choice set for many matches, an error message in case of no match). When the identification was successful, the AGS has to be used in order to identify the provider for this city. This must be the provider with span 9 in the first place. The StartDestIdentification-interface of the server can be retrieved out of table MetaServer in combination with this provider code. Finally, the stop name or address string of the customer will be sent to the local server for a complete resolution. This request must contain the AGS so that the local server must not try to identify the city again.

2.4.13 Metadata Update Process

The process of developing and updating the metadata was initialized by the co-ordinator (see Figure 27). He initially developed an empty set of tables which is only filled with base data like providers' identifiers and a range of meta-ID's for transition points. A copy of this set of tables is distributed to every provider. Every provider inserted all relevant data of his responsibility. This contains reachability data of each provider's servers, the cities and municipalities in his responsible area, and transition points agreed with his neighbours. After having received all data, the co-ordinator integrates all separate delivered tables. During this integration process errors and conflicts may occur. Especially during the start phase, conflicts between city names as described and the description of the table MetaCity occur. Those problems are sent back to the parties concerned.

When the integration of all tables was successful, the integrated set of tables as a complete metadata base will be returned to the providers and to their system providers in order to use that metadata for the task it was designed for.

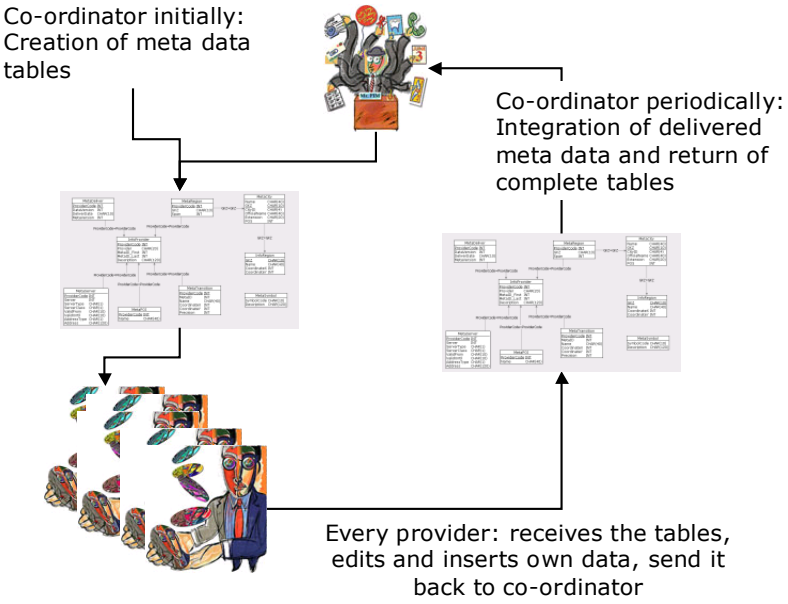

Figure 27: Overview: process of updating the metadata base

After the initial process updates only take place in case when a provider delivers a new update of the data base (i.e. when new transition points have to be added), server addresses change or servers are added or, seldom, provider responsibilities change.

Every delivery of a changed table set from a provider changes its version number, and every distribution via the co-ordinator changes its version number so that a list of version history exists available for every provider and the co-ordinator. The history of the co-ordinator represents the number of deliveries of complete metadata to all providers.

3 The DELFI Web Service

3.1 Technology Introduction

The XML Schema Definition (XSD) is used to define types of objects in an API.

The Web Service Description Language (WSDL) is used to define network services as a set of endpoints operating on messages containing information.

This chapter introduces the features of XSD and WSDL and illustrates the syntax used to describe interfaces. The introduction shall give a short overview and it is not intended to provide a complete understanding of the mechanism and the language. For a more detailed look of the specific documentations see http://www.w3.org/XML/Schema and http://www.w3.org/TR/wsdl. Overview informations can be found e.g. at Wikipedia under http://en.wikipedia.org/wiki/Xsd and http://en.wikipedia.org/wiki/Web_Services_Description_Language.

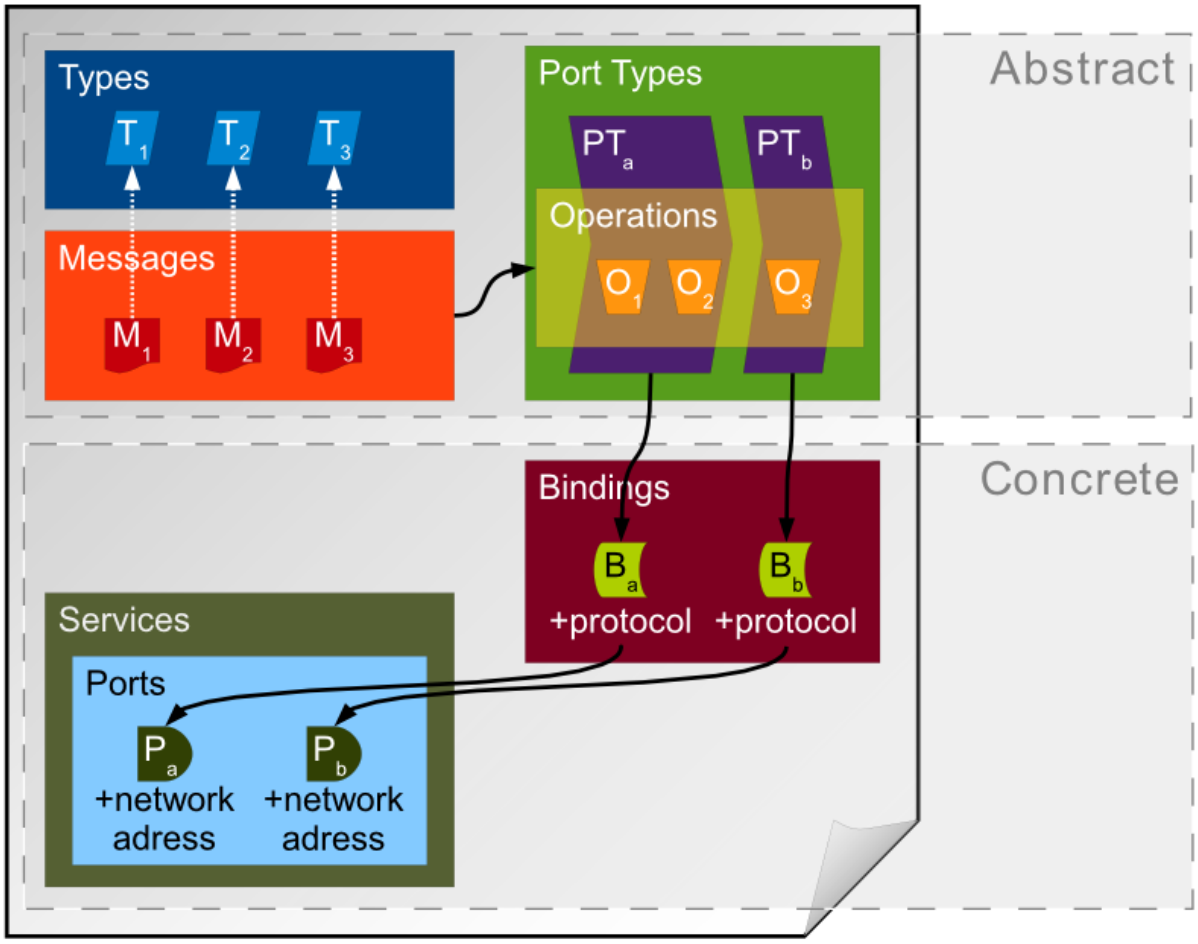

Figure 28: Overview: structure of a Web Service Defintion

In Figure 28, the principle structure of a Web Service Definition is shown. An an abstract level the operations of the web service are defined as messages of specific types transported over specific ports.

On the concrete level bindings and protocol are mapping these operations to dedicated ports of services on specific addresses.

3.2 DELFI Service Introduction

The DELFI Web Service consists of one namespace and 9 operations. The operations are permitting functions to

- resolving locations
    - Locations()
- computation itineraries
    - Transitions()
    - PartialConnections()
    - Connections()
- additional operations
    - RefineConnections()
    - Multi()
- retrieve informations of the server
    - Capabilities()
    - AllTransitions()
    - Usage()

The DELFI Web Service is specified in two files:

delfi5.0.eng.xsd	This file contains the XML Schema Definition. It defines attributes, simple types, complex types and request and response types.
delfi5.0.eng.wsdl	This file contains the Web Service Definition of the DELFI Service. It defines the port, the messages, the endpoints etc. It is generated from the DELFI XSD.

The file delfi5.0.eng.wsdl enables already the implementation of a DELFI Web Service. The other file contains reduandant information.

3.3 Structure of the Operation Declaration

The following sections explain the different operations in detail. In order to understand the structure of these explanations the following structure is used:

1. Flow of the usage of operations in interaction diagrams with the necessary objects including a description of the sub flows
2. Definition of every operation with its input (request) and output (response) data structure in formal tables, object diagrams with relations and object element description
3. Activity diagrams where necessary
4. Examples of inputs and outputs with some selected examples of alternative flows

The examples and flows cannot cover all possibilities of alternative flows and data combinations. They shall only support a developer to get to the first contact with the DELFI system.

In the following detailed description, all interfaces and contained methods will be explained in formal and informal form. That means basic types as objects can be left in an initialized state and objects or lists have to be compiled but with a default value.

NOTE:

Within the XML schema definition, all logical types are named '*Type'. Input message types are named '*RequestType' and output message types '*ResponseType'.

3.4 Descriptions of Flow

This interface implements the first method necessary for the first steps of calculating the itinerary.

NOTE:

In difference to the introducing description in chapter 2.3.1 (see above), the actual implementation expects a fully pre-processing of city name resolution outside the search controller. This flow is described as follows.

3.4.1 Description of the Flow for Location Identification

Figure 29 illustrates the interaction diagram of the resolution process of locations with customers' inputs. The flow description is used for a typical implementation in the

German DELFI environment. Other flows may vary dependent on the place where the customers' inputs are stored in order to recognise the inputs (state controlling), for example.

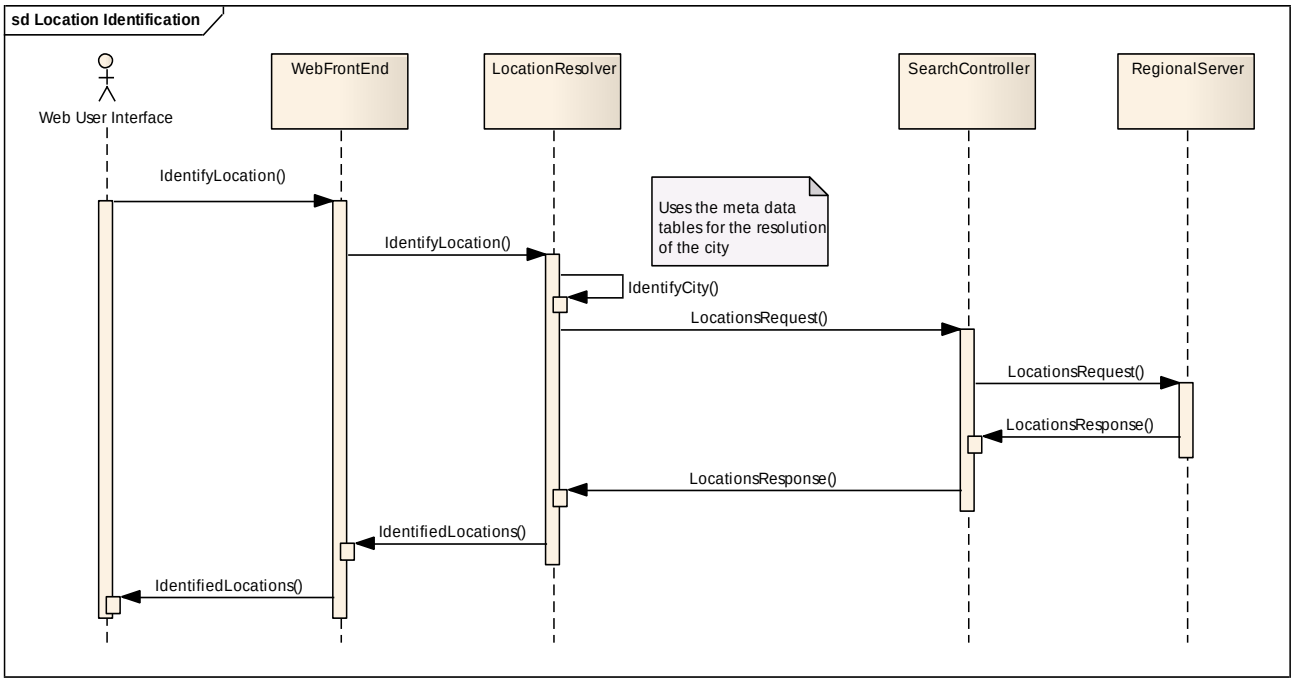

Figure 29: Flow diagram of resolving of origin/destination locations using Locations()

Flow:

1. In order to operate the process, a component is neccesary which resolves the city names given by the customer. This component is the object named 'LocationResolver' in the flow diagram. The component has the task to read the metadata tables and to use the information coded in this tables in order to resolve the city name, to retrieve the AGS and the responsible provider of this city. This identification process is already described in chapter 2.4.13. The result is usually a set of possible passive servers which should be able to resolve the input sequence completely. If one or both of the city name(s) cannot be resolved as individuals, a first reply to the customer is necessary with a list of possible matches of city names which are name-related to the input. In that case, the customer has to select one of the presented cities listed. That new city name chosen by the customer differs to the first input of the customer by already containing the AGS of the chosen city, which allows a direct forward of the completed sequence with the AGS.
2. At first, the origin and destination servers have to be identified and therefore match the AGS. A local implementation can even add that information to a list of choices useful for the customer. In either case, the table MetaRegion provides the relevant information of the responsible provider. This information leads to the possibility of resolving the corresponding server address in the metadata table MetaServer.

3. Then, the completed (with the according AGS of the town) set of *LocationsRequestType* is sent to the relevant local server for the AGS (span=9). The internal data structure *LocationEntryType* contains the data sequence.
4. The local server is now in charge to identify the second type of customer input: stop name, address, poi name or coordinate. The AGS in the submitted data structure ensures that the local server must not resolve the city name again.
5. If a local server cannot explicity identify the location string the passive server has to reply a list of possible alternatives to the controller which submits the list to the customer in order to render a possible selection. Meanwhile, the local server has completed the different choices with information which allow now a direct search by the search controller if the customer has chosen one of the alternatives. The data sent back to the customer contains the key for every submitted location which can be used by the search controller for the connection request.

NOTE:

In some implementations of web front ends it is common that a customer defines his input in an unstructured order so that the city name can be located before or after the location text and in one text field (no separated fields for the city name and the stop or address name). This leads to the problem that both alternatives have to be tested and all results are passed to the customer for selection.

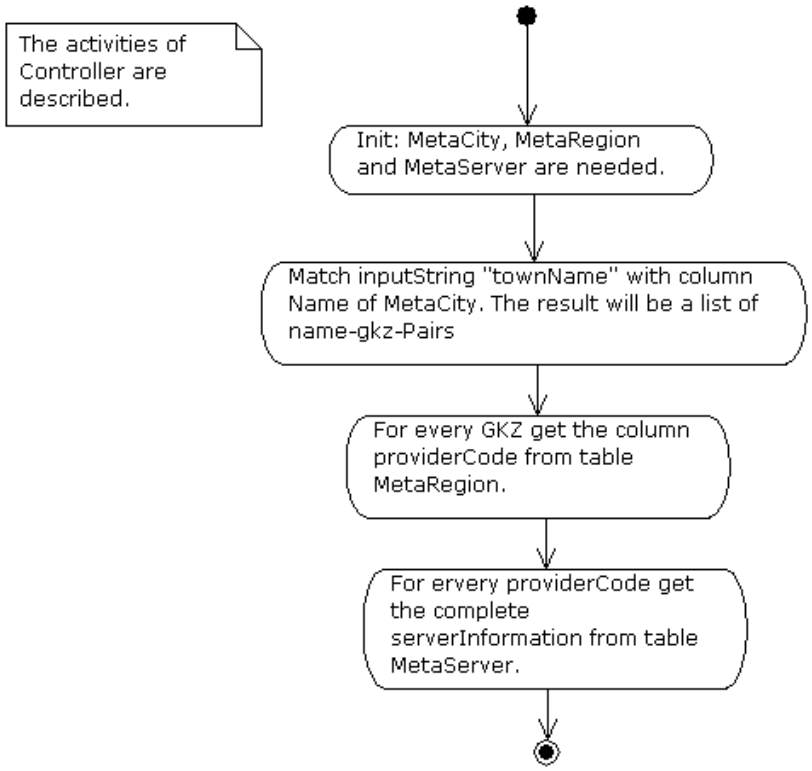

Figure 30: Flow of the activities of pre-processing the city names outside of the search controller

The flow in Figure 30 is an abstract one and can be implemented anywhere in a local component outside of the search controller. In existing information systems, it might be common to carry it out together with the existing location resolving part. The only task is to read the metadata tables and to process the data combinations.

3.4.2 Description of the Flow for Non-Distributed Trip Calculation

In Figure 31, the interaction diagram of search controller and its relation to servers using the Web Service interface is illustrated. The interface implements functions which represent the ordinary behaviour of a normal information server. The functions' task is to provide information about the valid time table period for requests and the calculation of complete itineraries. The operation *Capabilities()* can be used to get and store valid time periods internally in the search controller in order to avoid unnecessary calls but it can also be used for watchdog components to keep information about the availability of servers up-to-date.

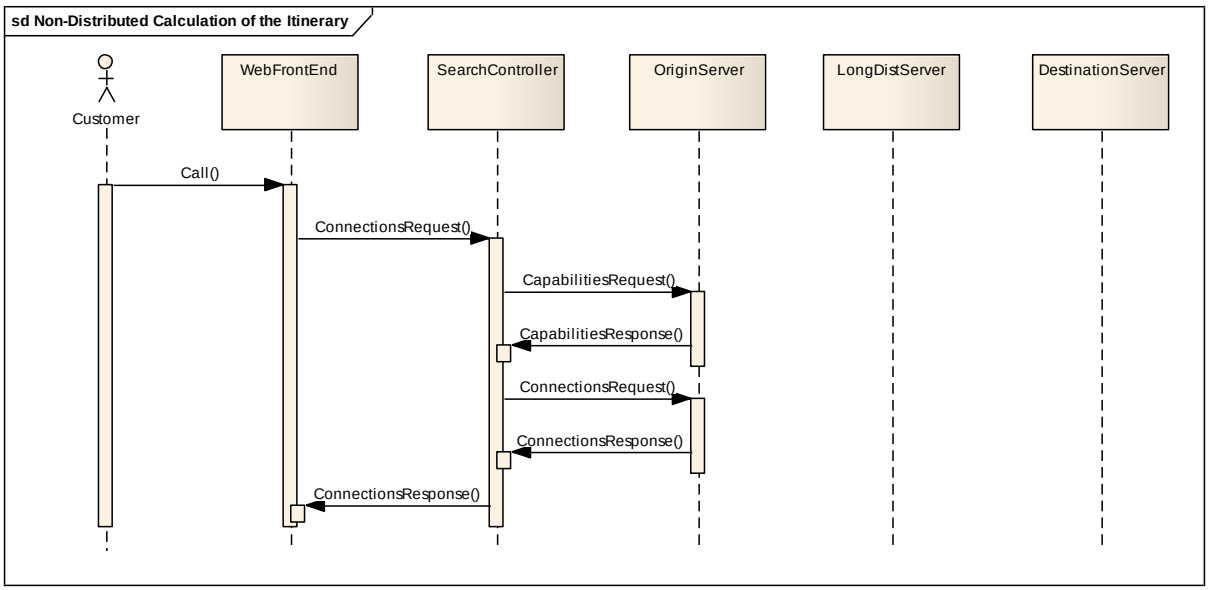

Figure 31: Sequence Diagram of Non-Distributed Trip Calculation

Flow:

1. After a successful identification of locations, the necessary keys of the relevant stops are available in the *LocactionKeyType* structure.
2. Secondly, the *Connections()* operation will be called after the identification of the locations to the search controller, which acts against the web front end as a kind of super information server, was successful.
3. Optionally, the preferred provider for the calculation of the long-distance part can be set. The real used provider will return there. It has to be notified that only two locations (origin and destination) can be set for the input.
4. The calculated connection will return in the output parameter field

ConnectionsResponseType.

5. Errors may occur if no transport is defined for the given stops or the time table is not within the requested time and date period or system errors avoid a result.

NOTE: the *Connections()* operation is designed in order to calculate an itinerary by a DELFI server which is completely responsible for origin and destination. This means that the server covers the complete itinerary by its internal time table data. If a client calls this method of a search controller the search controller acts as a kind of super information server which seems to cover the complete time table data. Such a client (like a web front end) does not care about which servers have to be contacted in order to calculate the complete itinerary.

3.4.3 Description of the Flow for Distributed Trip Calculation

In Figure 32, the interaction diagram of the process of distributed trip calculation is shown.

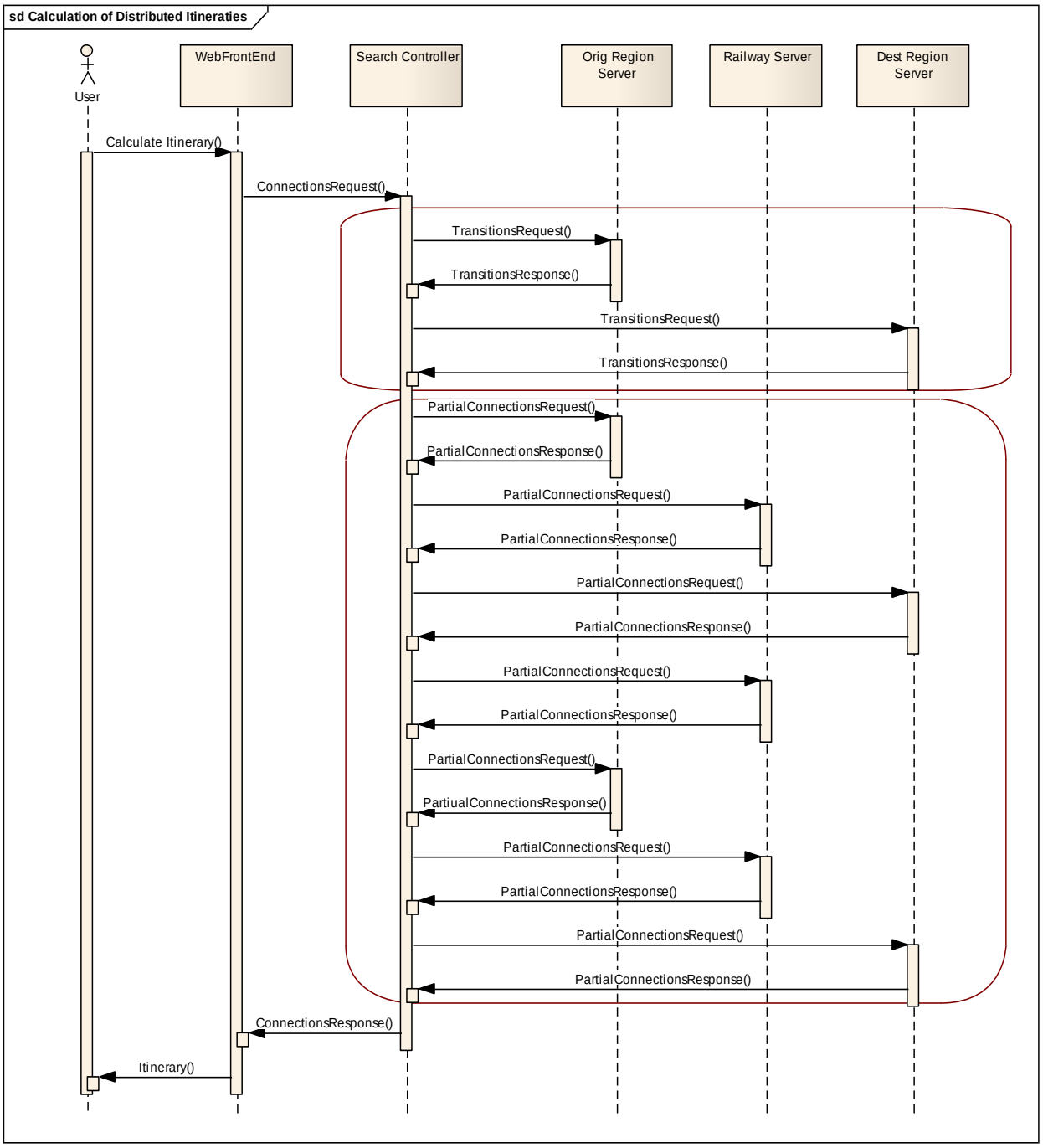

Figure 32: Sequence Diagram of Distributed Trip Calculation

Flow:

1. In the case that all city name entries have been identified at once, the complete input sequence is passed to the responsible passive servers. The structure LocactionKeyType is responsible for the correct identification of the location for further computation. It does not matter if the *Locations()* is performed by the web front end or by the local component responsible for the location resolve or by another local control component. The search controller has to be invoked to perform the distributed itinerary computation.
2. At first, the search controller checks if all servers are able to calculate the

partial connection within the desired travel date. All participating server use the *Capabilities()* operation in order to finding that out. If the date is scheduled during the valid period of all participating server further computation can be done. If not, the customer has to be informed that the desired travel date is out of the time table period.

3. If a computation can be started transition points are the next elements which are necessary. They are requested by the *Transitions()* or *AllTransitions()* operation from the origin and from the destination server.
4. With the origin location(s) and the transition points, the partial connections can be requested from the origin server in case of a departure search in the first phase (the flow is analogue to the abstract description of computation in section 2.3.2). In case of a departure search the result of the first (origin) server will be a set of arrival nodes (transition nodes) with time labels.
5. These nodes are used for the next request to the intermediate long-distance transport server. It has to compute the m:n combinations for all possible connections of possible transition nodes from the origin server (m elements, those with retrieved time labels) to all transition points of the destination server (n elements). The result will also be a set of nodes with time labels (n or less).
6. The resulting nodes will be used for the third step of the first phase; the calculation of connections to the destination from all arrival nodes of the long-distance server. The resulting nodes (again those with time labels) will be used to compute connections to the destination.
7. Afterwards, phases 2 and 3 will be performed where the backward search and the forward route computation is done the same way as in the first phase.

NOTE: The special feature of the implementation is the optimizing of the whole internal computation process in order to reduce complexity and time consumption for the computation.

3.5 DELFI Operations

The following table enumerates all operations of the DELFI Service:

Operation	**Description**
Capabilities()	Description of the capabilities of the passive server
AllTransitions()	Returns a list of all active transition points of the passive server
Locations()	Allows to identify a origin or destination location
Transitions()	Returns transition points for a given location
PartialConnections()	Calculates one or two partial calculations of a distributed

	calculation process
Connections()	Returns a complete trip between two locations. Called for passive servers if both locations are in the area of the passive server. Called for active servers start the distributed calculation process
RefineConnections()	Dedicated servers can enrich complete schedules with additional informations (e.g. tariff informations or map links)
Multi()	Allows optimization of remote calls of methods. A Multi request can contain multiple subrequests. The Multi reponse contains multiple subresponses.

The following diagram shows the operations of the port type delfi5Port

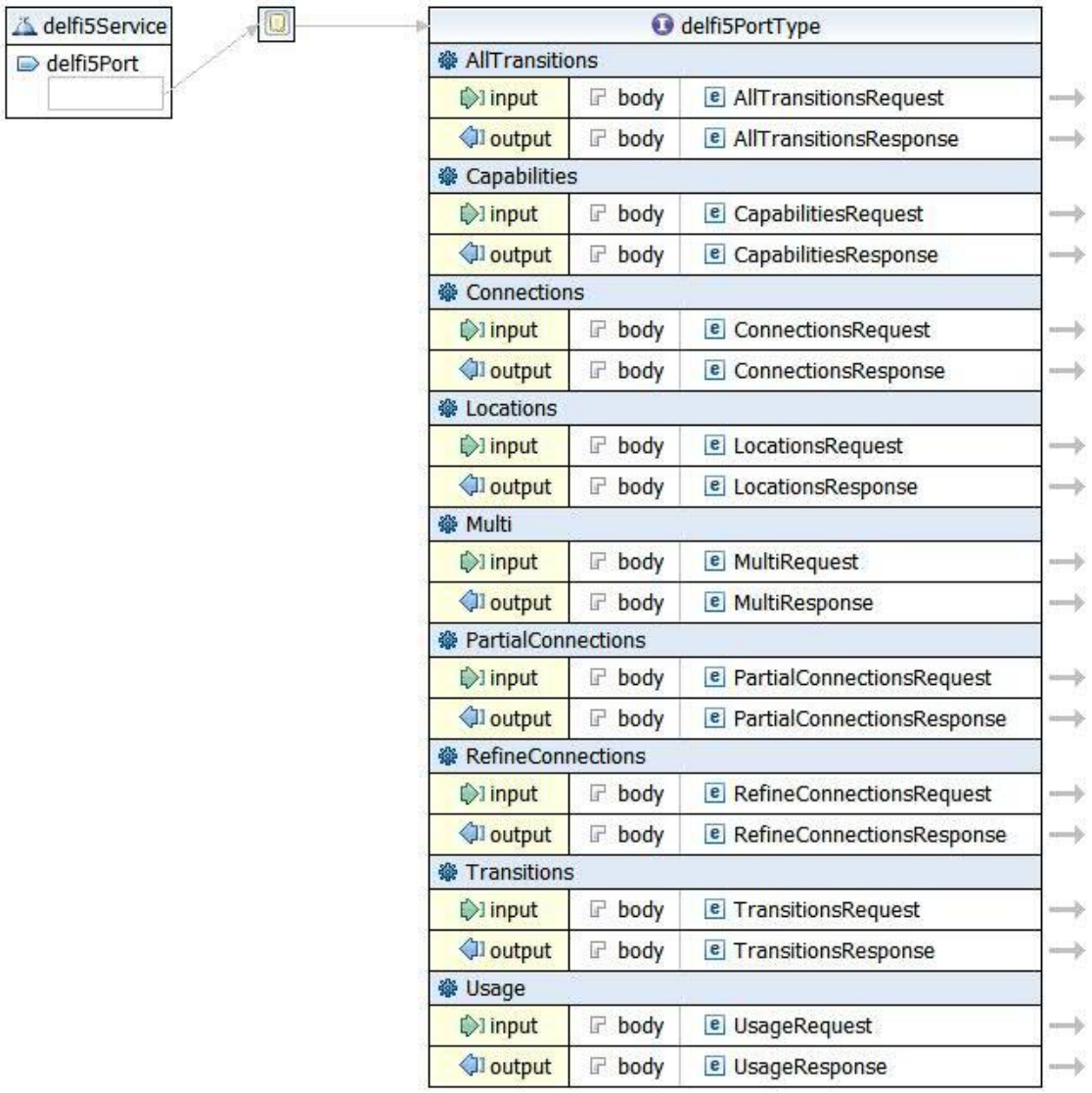

Figure 33: Operation of the Port Type delfi5Port

For each operation *<operation>()* two message types are defined

1) <*operation*>RequestType defines the type of the request message
2) <*operation*>ResponseType defines the type of the response message

All request types are based on the common *BaseRequestType*, containing common request attributes like the *TransactionType*.

All response types are based on the common *BaseResponseType*, containg common response attributes like the calculation time.

3.5.1 Operation Capabilities()

The operation *Capabilities()* provides meta information of servers.

3.5.1.1 Declaration

```
<wsdl:operation name="Capabilities">
    <wsdlsoap:operation soapAction="delfi5eng/Capabilities"/>
    <wsdl:input name="CapabilitiesInput">
          <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="CapabilitiesOutput">
          <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
```

3.5.1.2 Description

The *Capabilities()* operation has been defined in this interface for meta purposes. An active server can request meta information of the passive servers to optimize calculation and/or information messages.

But it can also be used for optimisation purposes. A search controller can store valid time periods for all servers and omit a call if it can be determined that a desired travel date is not during the valid time range of one of the necessary server.

Capabilities() returns information containaining

- the beginning and the end date of validity of the time table
- supported languages
- supported features
- supported products
- supported location types
- supported handicap options
- etc.

Time and date are local times referring to the location of the corresponding location and time table data.

3.5.1.3 Request Type CapabilitiesRequest

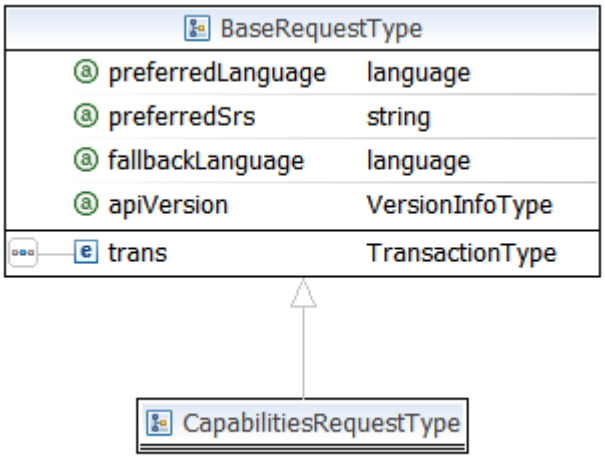

Figure 34: Class diagram CapabilitiesRequestType

The request type *CapabilitiesRequestType* is a pure *BaseRequestType* without any additional attributes.

3.5.1.4 Response Type CapabilitiesResponse

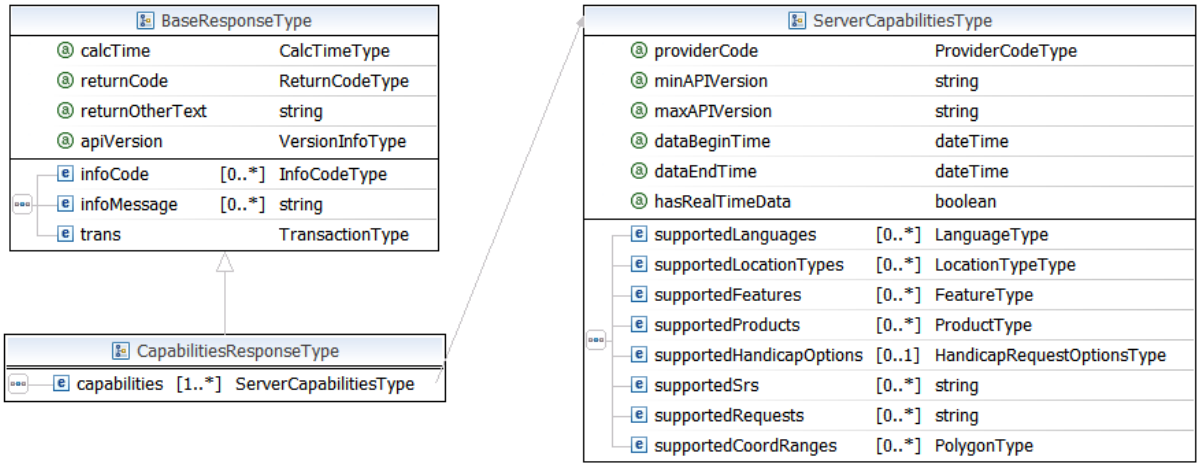

Figure 35: Class diagram CapabilitiesResponseType

The *Capabilities*() response type *CapabilitiesResponseType* contains the following information:

- providerCode of resonding server
- minimum/maximum API Version supported by the server (typically only one version)
- begin and end date of available time table data
- flag, if passive server knows real time data

- sequence of supported languages
- sequence of supported location types
- sequence of supported features
- sequence of supported products
- sequence of supported handicap options
- sequence of supported coordinate systems
- sequence of supported requests
- sequence of supported coordinate ranges (polygons)

3.5.1.5 Example

```
<?xml version="1.0" encoding="utf-8"?>
<CapabilitiesRequestType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" preferredLanguage="de" preferredSrs=""
fallbackLanguage="en" apiVersion="5.0.0.5">
        <trans providerCode="9" providerType="delfi" sequence="152235941" xmlns="delfi5eng"
/>
</CapabilitiesRequestType>
```

Figure 36: Example Capabilities() request

```
<?xml version="1.0" encoding="utf-8"?>
<CapabilitiesResponseType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" calcTime="0.015625" returnCode="OK" apiVersion="5.0.0.5">
    <infoCode xmlns="delfi5eng">ok</infoCode>
    <infoMessage xmlns="delfi5eng" />
    <trans providerCode="9" providerType="delfi" sequence="152235941" xmlns="delfi5eng" />
    <capabilities providerCode="0" minAPIVersion="Item5003" maxAPIVersion="Item5003"
dataBeginTime="2008-11-01T00:00:00" dataEndTime="2009-12-12T23:59:59" hasRealTimeData="false"
xmlns="delfi5eng">
        <supportedLanguages>de</supportedLanguages>
        <supportedLanguages>en</supportedLanguages>
        <supportedLocationTypes>stop</supportedLocationTypes>
        <supportedLocationTypes>address</supportedLocationTypes>
        <supportedLocationTypes>poi</supportedLocationTypes>
        <supportedLocationTypes>coord</supportedLocationTypes>
        <supportedFeatures>0</supportedFeatures>
        <supportedFeatures>4</supportedFeatures>
        <supportedProducts>0</supportedProducts>
        <supportedProducts>1</supportedProducts>
        <supportedProducts>2</supportedProducts>
        <supportedProducts>3</supportedProducts>
        <supportedProducts>4</supportedProducts>
        <supportedProducts>5</supportedProducts>
        <supportedProducts>6</supportedProducts>
        <supportedProducts>7</supportedProducts>
        <supportedProducts>8</supportedProducts>
        <supportedProducts>9</supportedProducts>
        <supportedProducts>10</supportedProducts>
        <supportedProducts>11</supportedProducts>
        <supportedProducts>12</supportedProducts>
        <supportedProducts>13</supportedProducts>
        <supportedProducts>14</supportedProducts>
        <supportedProducts>15</supportedProducts>
        <supportedProducts>16</supportedProducts>
        <supportedProducts>17</supportedProducts>
        <supportedProducts>18</supportedProducts>
```

```
        <supportedProducts>19</supportedProducts>
        <supportedProducts>20</supportedProducts>
        <supportedProducts>21</supportedProducts>
        <supportedProducts>22</supportedProducts>
        <supportedProducts>23</supportedProducts>
        <supportedProducts>40</supportedProducts>
        <supportedSrs>EPSG:4326</supportedSrs>
        <supportedRequests>AllTransitions</supportedRequests>
        <supportedRequests>Capabilitities</supportedRequests>
        <supportedRequests>Connections</supportedRequests>
        <supportedRequests>Locations</supportedRequests>
        <supportedRequests>Multi</supportedRequests>
        <supportedRequests>PartialConnections</supportedRequests>
        <supportedRequests>RefineConnections</supportedRequests>
        <supportedRequests>Transitions</supportedRequests>
        <supportedRequests>Usage</supportedRequests>
    </capabilities>
</CapabilitiesResponseType>
```

Figure 37: Example Capabilities() response

3.5.2 Operation AllTransitions()

The operation *AllTransitions()* returns a list of all active transition points of a passive server.

3.5.2.1 Declaration

```
<wsdl:operation name="AllTransitions">
    <wsdlsoap:operation soapAction="delfi5eng/AllTransitions"/>
    <wsdl:input name="AllTransitionsInput">
          <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="AllTransitionsOutput">
          <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
```

3.5.2.2 Description

This operation delivers all transition points of a passive server. This method can be used either by a monitoring system to achive the list of all active transition points or if no other possibility exists which reduces the list of transition points (see *Transitions()*). This method ensures that an optimal itinerary can be computed in a distributed way (see section 4.3) but in normal situations *Transitions()* is the appropriate method to get the transition points of a server, due to limitations in the passive server for long-distances. All transition points can be a large list but it can be used if *Transitions()* delivers an empty list.

The single input for the method is a BaseRequest-Element. The output provides the calculation time and a list of *TransitionType*-Elements. *TransitionType* contains the transition point in form of a *LocationType*-Element.

Nevertheless, in the productive DELFI system only the method *Transitions()* should be used to calculate trips.

3.5.2.3 Request Type AllTransitionsRequest

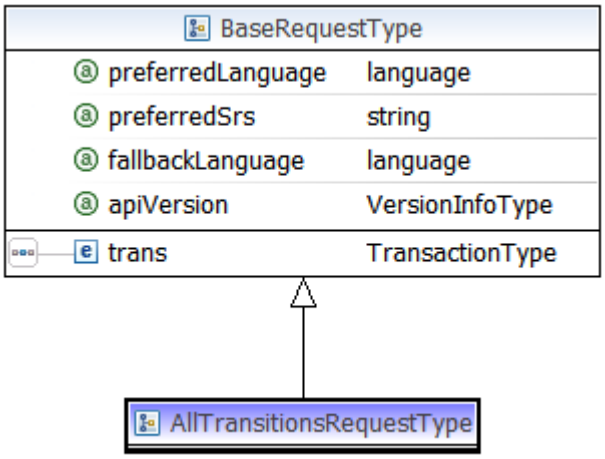

Figure 38: Class diagram AllTransitionsRequestType

The *AllTransitionsRequestType* is a pure *BaseRequestType* without any additional attributes.

3.5.2.4 Response Type AllTransitionsResponse

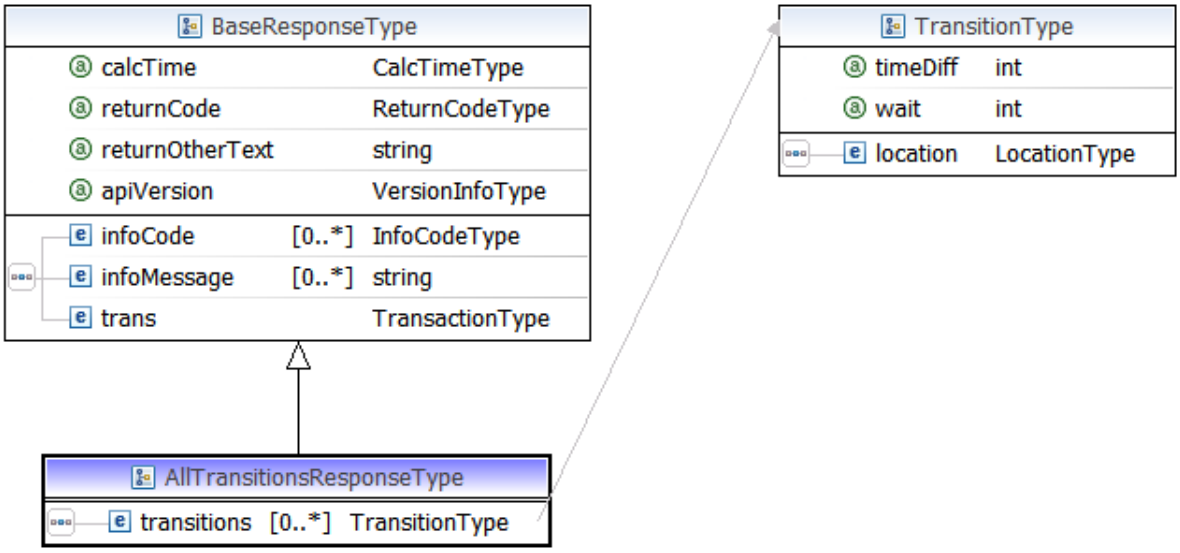

Figure 39: Class diagram AllTransitionsResponseType

The *AllTransitionsResponseType* contains beside the standard attributes of the BaseResponseType a sequence of *TransitionType* elements. Each *TransitionType* contains the transition point as a location element and two additional attributes timeDiff and wait, which are only relevant in the system EUSpirit, but not in DELFI.

3.5.2.5 Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<AllTransitionsRequestType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" preferredLanguage="de" preferredSrs=""
fallbackLanguage="en" apiVersion="5.0.0.5">
    <trans providerCode="9" providerType="delfi" sequence="152401832" xmlns="delfi5eng" />
</AllTransitionsRequestType>
```

Figure 40: Example AllTransitions() request

```xml
<?xml version="1.0" encoding="utf-8"?>
<AllTransitionsResponseType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" calcTime="1.921875" returnCode="OK" apiVersion="5.0.0.5">
    <infoCode xmlns="delfi5eng">ok</infoCode>
    <infoMessage xmlns="delfi5eng" />
    <trans providerCode="9" providerType="delfi" sequence="152401832" xmlns="delfi5eng" />
    <transitions xmlns="delfi5eng">
        <location name="Oberroth" state="complete">
            <key providerCode="0" key="100714" />
            <location name="Oberroth" regionName="Oberroth (b Dachau)" regionId="09174143:9">
                <coord x="0" y="0" />
            </location>
            <type>stop</type>
        </location>
    </transitions>
    <transitions xmlns="delfi5eng">
        <location name="Augsburg Hbf" state="complete">
            <key providerCode="0" key="8000013" />
            <location name="Hauptbahnhof" regionName="Augsburg" regionId="09761000:1">
                <coord x="0" y="0" />
            </location>
            <type>stop</type>
        </location>
    </transitions>
    <transitions xmlns="delfi5eng">
        <location name="Augsburg Hbf" state="complete">
            <key providerCode="0" key="8000013" />
            <location name="Hauptbahnhof" regionName="Augsburg" regionId="09761000:1">
                <coord x="0" y="0" />
            </location>
            <type>stop</type>
        </location>
    </transitions>
    <transitions xmlns="delfi5eng">
        <location name="Augsburg Hbf" state="complete">
            <key providerCode="0" key="8000013" />
            <location name="Hauptbahnhof" regionName="Augsburg" regionId="09761000:1">
                <coord x="0" y="0" />
            </location>
            <type>stop</type>
        </location>
    </transitions>
    <transitions xmlns="delfi5eng">
        <location name="Augsburg Hbf" state="complete">
```

```
            <key providerCode="0" key="8000013" />
            <location name="Hauptbahnhof" regionName="Augsburg" regionId="09761000:1">
                <coord x="0" y="0" />
            </location>
            <type>stop</type>
        </location>
    </transitions>
     …
</AllTransitionsResponseType>
```

Figure 41: Example AllTransitions() response

3.5.3 Operation Usage()

The operation *Usage()* can be called to retrieve statistical information of a server (active or passive).

3.5.3.1 Declaration

```
<wsdl:operation name="Usage">
    <wsdlsoap:operation soapAction="delfi5eng/Usage"/>
    <wsdl:input name="UsageInput">
           <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="UsageOutput">
           <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
```

3.5.3.2 Description

The operation *Usage*() was defined to allow a common reporting of the complete DELFI system in term of the usage of the servers.

A passive or active server can return statistical information of a complete period. The responding server is free to return a more or less long period (e.g. last month or only data since the last program startup).

3.5.3.3 Request Type UsageRequestType

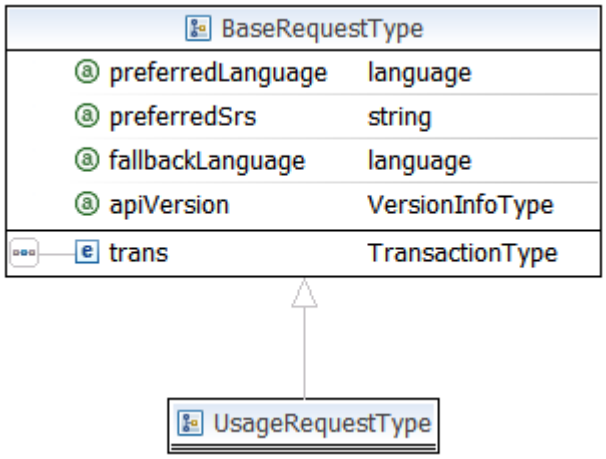

Figure 42: Class diagram UsageRequestType

The *UsageRequestType* is a pure *BaseRequestType* without any additional attributes.

3.5.3.4 Response Type UsageResponseType

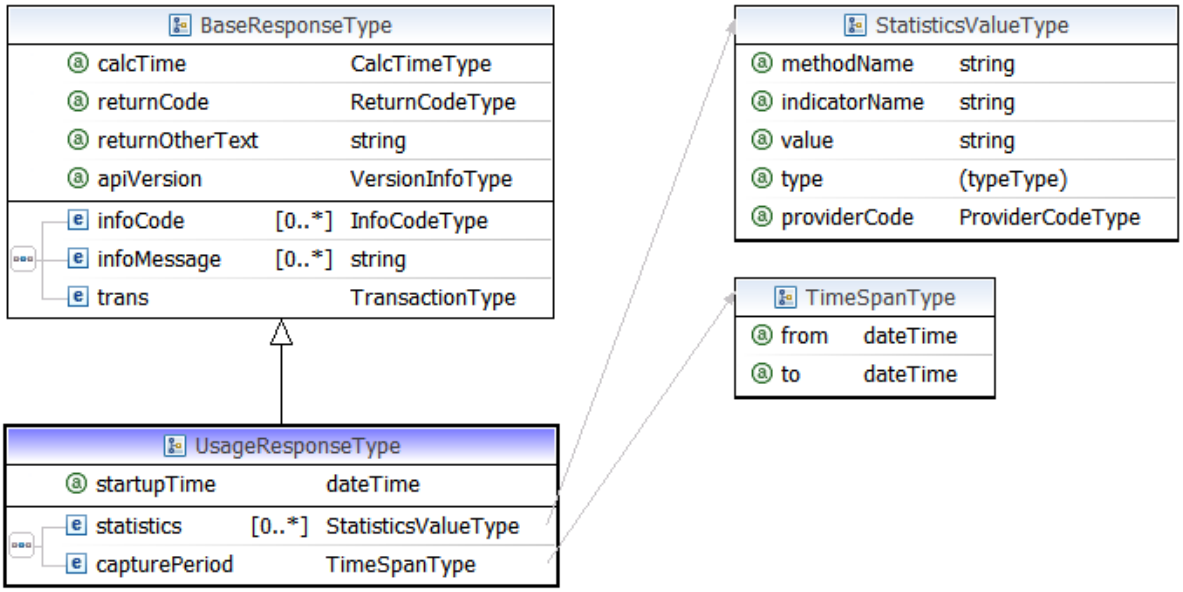

Figure 43: Class diagram UsageResponseType

The *UsageResponseType* contains the following information:

- capture period as two timestamps
- sequence of statistics values

Each statistics value contains the following attributes:

- name of the method
    - standard operation names (e.g. *Transitions* or *Locations*)
- name of the indicator
    - *Active* for values of outgoing requests (requests of active server)
    - *Passive* for values of incoming requests (of passive server)
- the provider code for this value
    - if indicator is Active then provider code of called passive server
    - if indicator is Passive then provider code of requesting active server
- type of the statistics value
    - avg for average values
    - sum for accumulated values
    - min for minimum values
    - max for maximum values
- (numeric) value of the statistics value

It is up to the responding server, which statistics values are returned in the operation response. Average and summarized values should nevertheless be returned to establish a common view of the overall system usage of the DELFI system.

The types *avg*, *min* and *max* are used to define response times in seconds. The type *sum* is used to define number of calls.

A common sense of minimum returned values is for a passive server that for each function and each provider code two numbers are returned:

1) sum containing the number of calls of the function from the provider

2) max containing the accumulated calculation time of these functions

With these values, the requesting system can determine the average calculation time of the functions because the capture period is also given in the response.

3.5.3.5 Example

```
<?xml version="1.0" encoding="utf-8"?>
<UsageRequestType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" preferredLanguage="de" preferredSrs=""
fallbackLanguage="en" apiVersion="5.0.0.5">
    <trans providerCode="9" providerType="delfi" sequence="152408098" xmlns="delfi5eng" />
</UsageRequestType>
```

Figure 44: Example Usage() request

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<UsageResponseType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" calcTime="0.015625" returnCode="OK" apiVersion="5.0.0.5"
startupTime="2009-01-14T15:24:08.3636802+01:00">
    <infoCode xmlns="delfi5eng">ok</infoCode>
    <infoMessage xmlns="delfi5eng" />
    <trans providerCode="9" providerType="delfi" sequence="152408098" xmlns="delfi5eng" />
    <statistics methodName="PartialConnections" indicatorName="Active" value="8892" type="sum"
providerCode="1" xmlns="delfi5eng" />
    <statistics methodName="Transitions" indicatorName="Active" value="2664" type="sum"
providerCode="6" xmlns="delfi5eng" />
    <statistics methodName="Locations" indicatorName="Active" value="114" type="sum" providerCode="5"
xmlns="delfi5eng" />
    <statistics methodName="Connections" indicatorName="Active" value="241" type="sum"
providerCode="13" xmlns="delfi5eng" />
    <statistics methodName="PartialConnections" indicatorName="Active" value="5793" type="sum"
providerCode="13" xmlns="delfi5eng" />
    <statistics methodName="PartialConnections" indicatorName="Active" value="13741" type="sum"
providerCode="219" xmlns="delfi5eng" />
    <statistics methodName="Transitions" indicatorName="Active" value="2033" type="sum"
providerCode="5" xmlns="delfi5eng" />
    <statistics methodName="Locations" indicatorName="Active" value="115" type="sum" providerCode="6"
xmlns="delfi5eng" />
    <statistics methodName="PartialConnections" indicatorName="Active" value="7073" type="sum"
providerCode="6" xmlns="delfi5eng" />
    <statistics methodName="Transitions" indicatorName="Active" value="3408" type="sum"
providerCode="1" xmlns="delfi5eng" />
    <statistics methodName="Locations" indicatorName="Active" value="127" type="sum" providerCode="13"
xmlns="delfi5eng" />
    <statistics methodName="Connections" indicatorName="Active" value="314" type="sum" providerCode="6"
xmlns="delfi5eng" />
    <statistics methodName="PartialConnections" indicatorName="Active" value="5189" type="sum"
providerCode="5" xmlns="delfi5eng" />
    <statistics methodName="Transitions" indicatorName="Active" value="2170" type="sum"
providerCode="13" xmlns="delfi5eng" />
    <statistics methodName="Locations" indicatorName="Active" value="70" type="sum" providerCode="1"
xmlns="delfi5eng" />
    <statistics methodName="Connections" indicatorName="Active" value="269" type="sum" providerCode="5"
xmlns="delfi5eng" />
    <statistics methodName="PartialConnections" indicatorName="Active" value="1.88" type="avg"
providerCode="1" xmlns="delfi5eng" />
    <statistics methodName="Transitions" indicatorName="Active" value="0.389" type="avg"
providerCode="6" xmlns="delfi5eng" />
    <statistics methodName="Connections" indicatorName="Active" value="2.376" type="avg"
providerCode="5" xmlns="delfi5eng" />
…
    <statistics methodName="Connections" indicatorName="Passive" value="358" type="sum"
providerCode="1" xmlns="delfi5eng" />
    <statistics methodName="PartialConnections" indicatorName="Passive" value="8899" type="sum"
providerCode="1" xmlns="delfi5eng" />
    <statistics methodName="AllTransitions" indicatorName="Passive" value="57" type="sum"
providerCode="9" xmlns="delfi5eng" />
    <statistics methodName="Connections" indicatorName="Passive" value="6319" type="sum"
providerCode="9" xmlns="delfi5eng" />  <statistics methodName="Capabilities" indicatorName="Passive"
value="13" type="sum" providerCode="9" xmlns="delfi5eng" />
…
</UsageResponseType>
```

Figure 45: Example Usage() response

3.5.4 Operation Locations()

The operation *Locations()* is used to identify origin and destination locations of a trip request.

3.5.4.1 Declaration

```
<wsdl:operation name="Locations">
    <wsdlsoap:operation soapAction="delfi5eng/Locations"/>
    <wsdl:input name="LocationsInput">
          <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="LocationsOutput">
          <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
```

3.5.4.2 Description

This function identifies or improves locations given by the input sequence. For each entry of the input sequence a list of locations is returned in the output sequence.

The input data structure *LocationsRequestType* consists of two main types: an array of *LocationType* elements which represents the input data of the customer or the partial resolved locations (customer's choice of a resolved city) and the *TransactionType* which contains the individual identification of the requesting search controller. It has to be clarified that every search controller has a range of individual identifiers for requests. The *LocationsRequestType* contains an element of *LocationType* and a limiter for the number of returned matches (*maxlist*). *LocationType* is a common element of the input and output structure and will be described later.

providerCode and sequence have to meet the demands of the active server (search controller). The sequence must be an individual one for every new request which is sent by a search controller that is able to identify every part plus the parts of a flow by the sequence and the followers of the subsequence.

The common structure used for inputs and outputs is the *LocationType*. The structure is shown in Figure 46. This data structure contains at first the text input of the customer. If the city is resolved the AGS of the stop or address remain unresolved. After the location is completely resolved, the structure provides a completed text string for the city and the location together with an entry which provides a key necessary for the connection calculation.

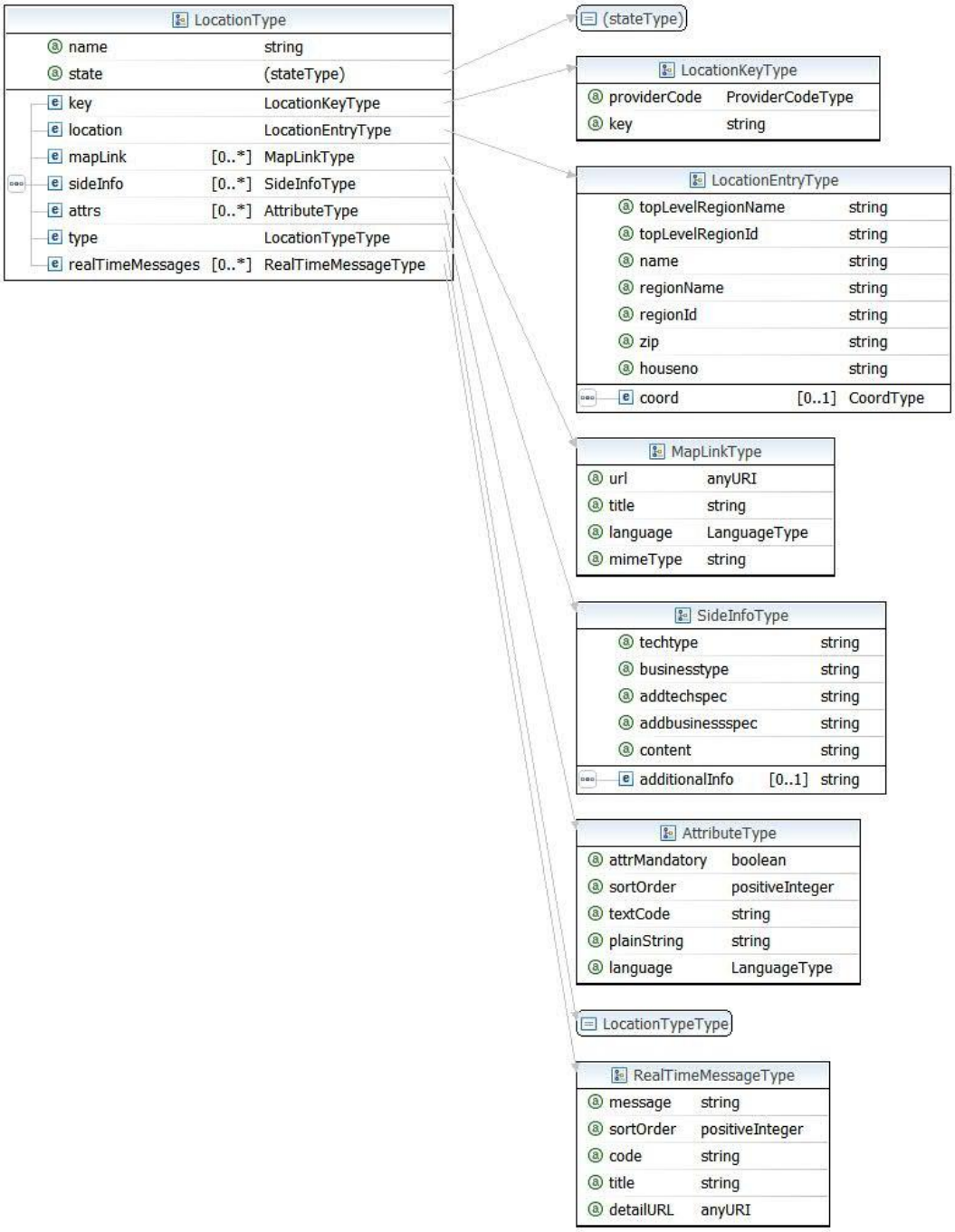

Figure 46: Detailed Class diagram LocationType

In order to complete the city name, name of the stop, or the address values like region name, complete stop name or coordinates are necessary. Every location object may have additional information kept in a list of the type *AttributeType*. *AttributeType* contains the text string with additional information for the customer, information if the

text has to be displayed mandatory, an order for the different text sequences is necessary, the language of this text, and the option to code this text.

The list of objects of the type *MapLinkType* is used for maps related to the ones like walk maps or situation views. The given information type and the physical type in MIME notation is mandatory together with the URL in the content of the map being delivered.

The list of objects of the type *SideInfoEntryType* is used for additional links, e.g. for booking systems.

LocationType contains the complete information of one location. A customer usually enters only a short sequence into the text fields for start and destination, these text elements will be completed during the identification phase. In the *name* field the complete name of a location must be set after completion.

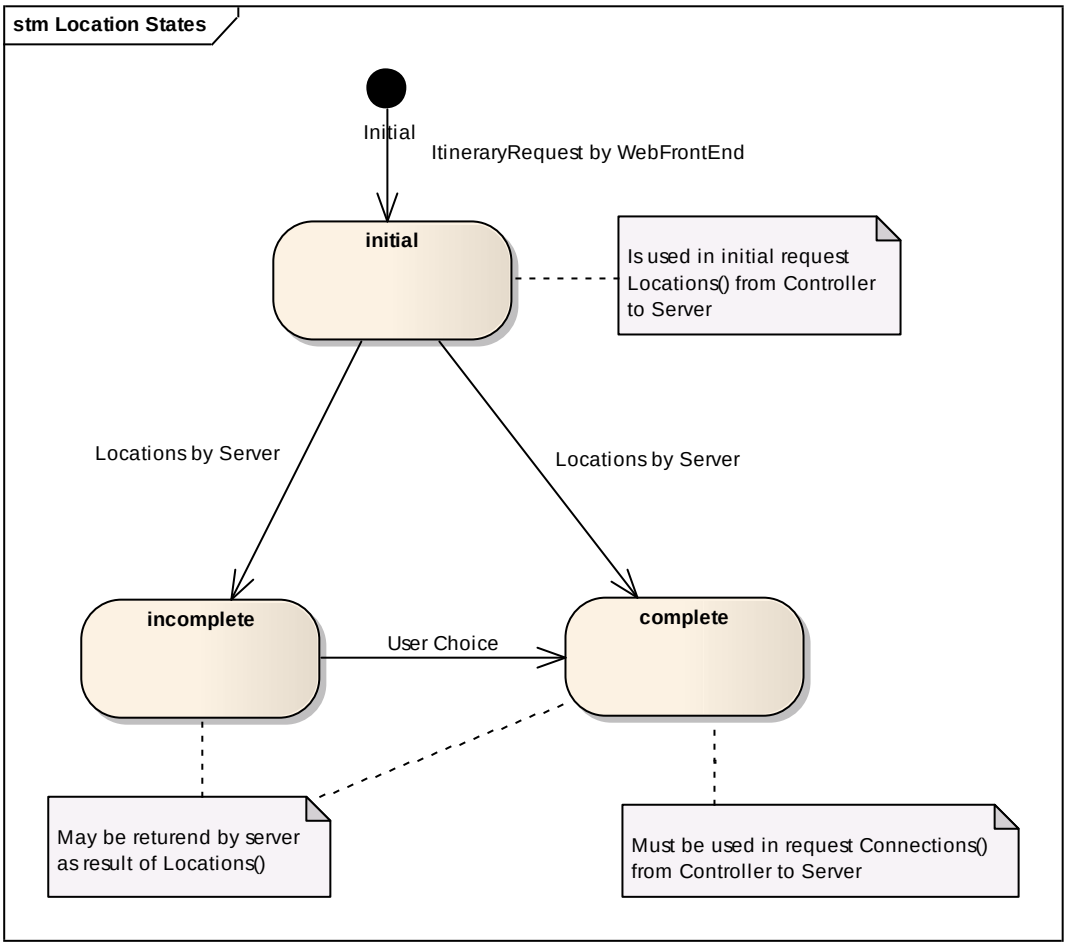

Figure 47: States of LocationType object during its usage as input/output object

The flow of states combined with data flow shall additionally illustrate the usage of the data flow.

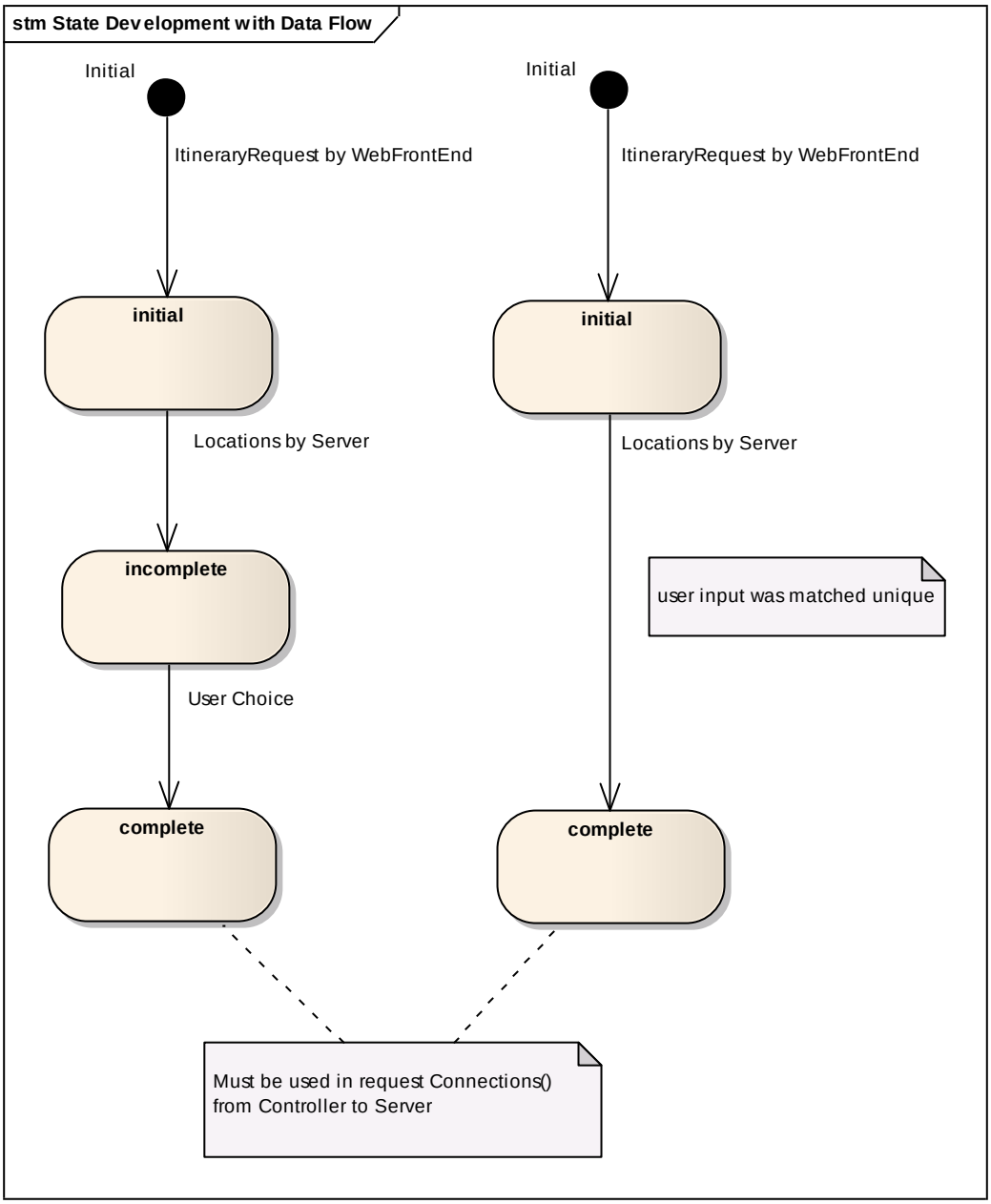

Figure 48: Example of state development with data flow

The following examples have been provided by mdv, a system provider for DELFI server. The XML formatted output is a form of formatting chosen by mdv and not necessary for DELFI itself. Logging can be done in any way.

The following example of a Locations-Request and the related answer contains a request to a town hall station. The input sequence contains only one location element. The sequences *LocationType* and *TransactionType* are filled in order to identify the request later. The *LocationType* contains the name of the location, the co-ordinates, and the city identifier (AGS). As the AGS is unique, a name for the region or toplevelregion has not to be given.

The response structure contains an updated version of *LocationType* filled by the responsible server and the *LocactionKeyType* as the status of the answer is *complete*. This key will be used for the itinerary request. It is obvious that the name in LocationType is defined in a different form than in the LocationEntryType. The name in LocationType (Kelsterbach Rathaus) is a combined string while the elements in LocationEntryType can contain a formal description which divides city name from stop name of the location in the answer structure. The kind of form used for the display or the printout depends on the web front end provider.

3.5.4.3 Request Type LocationsRequestType

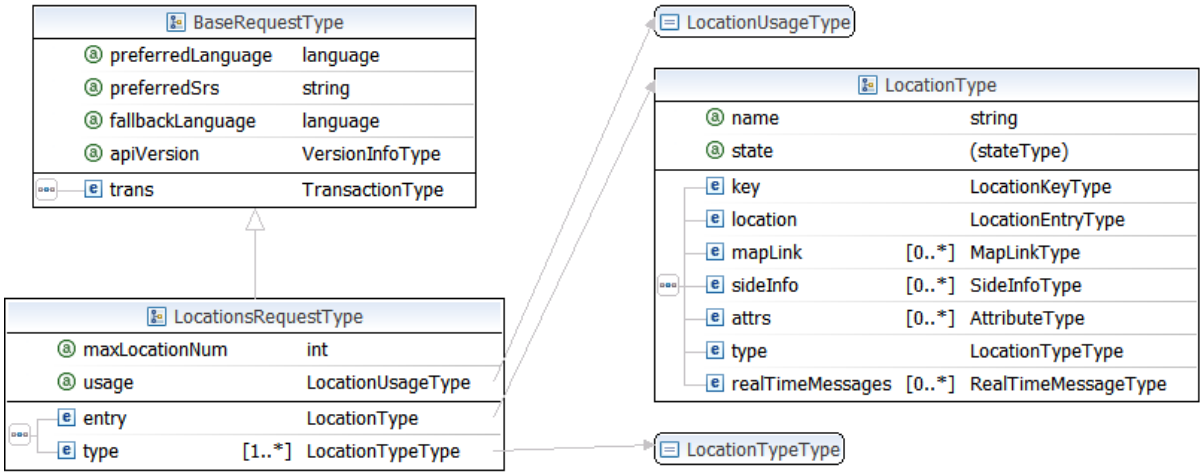

Figure 49: Class diagram LocationsRequestType

The *LocationsRequestType* contains beside the standard request attributes of the *BaseRequestType* a sequence of

- a non-empty LocationType
- a sequence of types of Locations

The entry element contains all already known information of the Location (i.e. user input of the location name or partially identified locations). The state of the location indicates the maturity of the location.

The sequence of types specifies in which location pools (stops, addresses, POIs or coordinates) the entry location should be matched.

3.5.4.4 Response Type LocationsResponseType

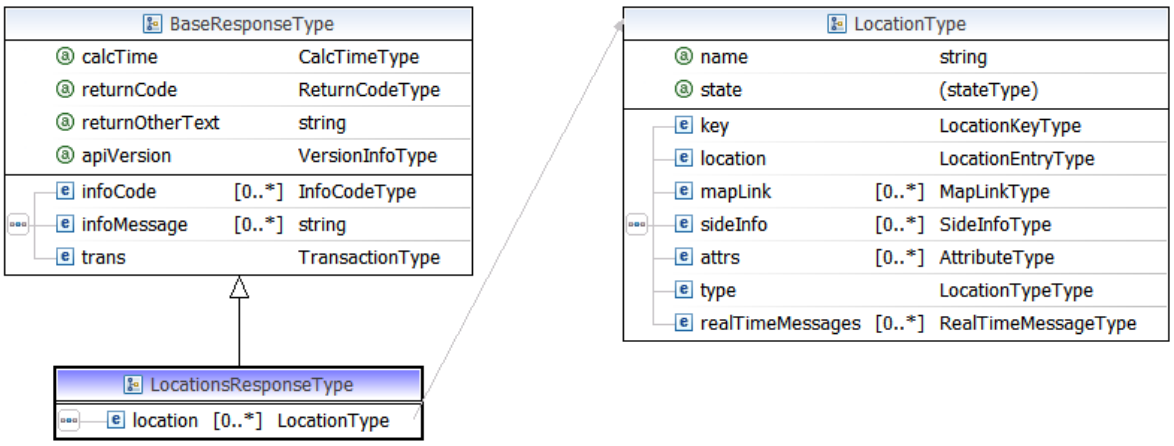

Figure 50: Class diagram LocationsResponseType

The response of the operation *Locations()* contains a sequence of *LocationType* elements. Each location is a candidate for a matching location. The state of the location indicates, if further requests are necessary to identify a completed location or not.

3.5.4.5 Example

```
<?xml version="1.0" encoding="utf-8"?>
<LocationsRequestType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" preferredLanguage="de" preferredSrs=""
fallbackLanguage="en" apiVersion="5.0.0.5" maxLocationNum="100">
    <trans providerCode="1" providerType="delfi" sequence="154506832" xmlns="delfi5eng" />
    <entry name="Kelsterbach rathaus" state="initial" xmlns="delfi5eng">
        <key providerCode="6" key="" />
        <location name="rathaus" regionName="Kelsterbach" regionId="06433007:1" />
        <type>null</type>
    </entry>
    <type xmlns="delfi5eng">stop</type>
</LocationsRequestType>
```

Figure 51: Example Locations() request

```
<?xml version="1.0" encoding="utf-8"?>
<LocationsResponseType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" calcTime="0.007" returnCode="OK">
    <trans providerCode="1" providerType="delfi" sequence="154506832" xmlns="delfi5eng" />
    <location name="Kelsterbach Rathaus" state="complete" xmlns="delfi5eng">
        <key providerCode="6" key="003002905-80" />
        <location regionId="06433007">
            <coord x="8532012" y="50061250" />
        </location>
        <type>stop</type>
    </location>
</LocationsResponseType>
```

Figure 52: Example Locations() response

3.5.5 Operation Transitions()

The operation *Transitions()* has to be performed to determine the transition points for a given location.

3.5.5.1 Declaration

```
<wsdl:operation name="Transitions">
    <wsdlsoap:operation soapAction="delfi5eng/Transitions"/>
    <wsdl:input name="TransitionsInput">
          <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="TransitionsOutput">
          <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
```

3.5.5.2 Description

This operation should deliver a limited list of transition points but the list shall contain those transition points which are necessary for the optimised itinerary. The *startLocation* is the location determined for the origin (starting point). The *destLocation* states the destination. Both locations can be used in order to determine the selection of appropriate transition nodes. This method must deliver transition points for the itinerary request. Possible methods for the selection of relevant transition points can be a search in the area around the start or destination or in the corridor between start and destination. The method of realisation for this selection is a matter of local implementation and knowledge.

All data types of *Transitions()* are described already. The locations used in the *Transitions()* operation need the keys and a minimum description of every location. This can be delivered by some servers but it might be more comfortable if some more information is available from a transition list.

Optimizations, for example, can be done by an internal storing of frequently requested start – destination combinations and their corresponding transition points which might vary less than the corresponding connections.

3.5.5.3 Request Type TransitionsRequestType

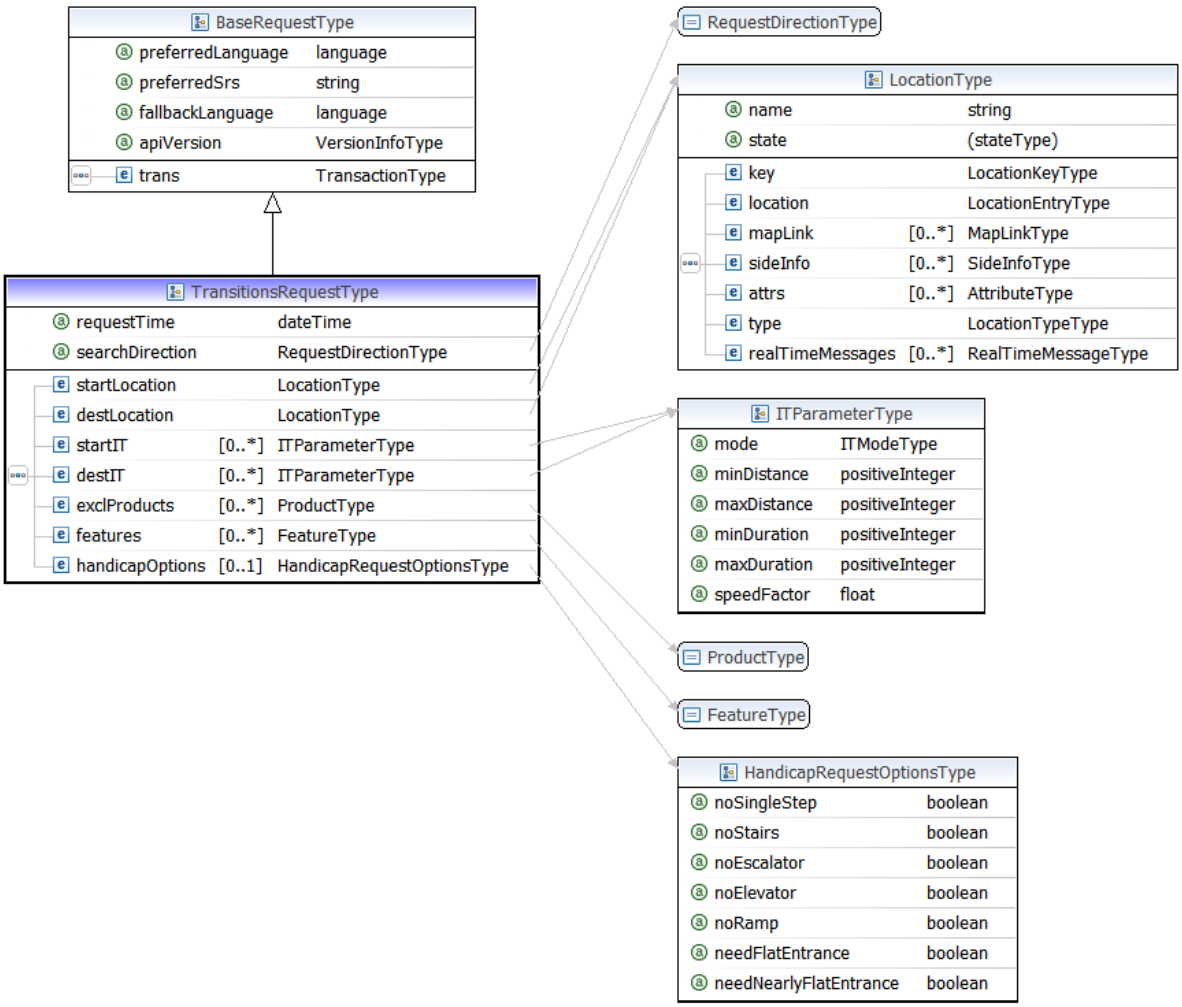

Figure 53: Class diagram TransitionsRequestType

The *TransitionsRequestType* contains the following additional attributes:

- startLocation : the location, for which the transitions should be calculated
- destLocation : the target location of the trip if TransitionsRequestType.startLocation contains the origin location of the trip and vice versa
- exclProducts : can influence the set of transition points
- features : can influence the set of transition points
- handicapOptions : can influence the set of transition points
- requestTime : can influence the set of transition points

The search controller will call the origin and destination servers which alterning locations. The call to the origin region server contains the origin location of the trip in startLocation and the destination location of the trip in destLocation. The call to the destination region server contains the destination location of the trip in startLocation and the origin location of the trip in destLocation.

3.5.5.4 Response Type TransitionsResponseType

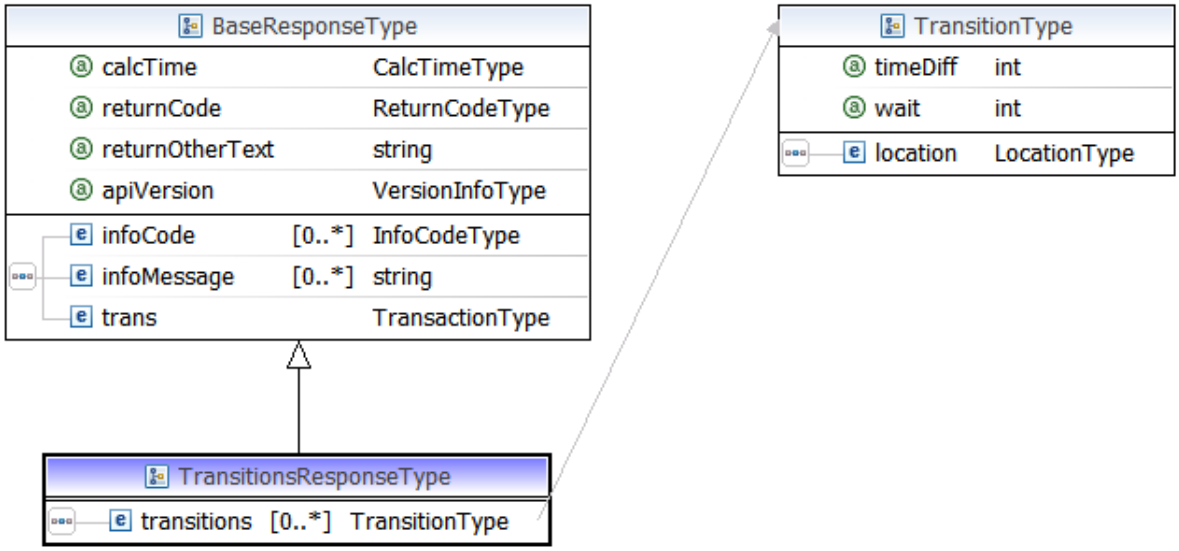

Figure 54: Class diagram TransitionsResponseType

The *TransitionsResponseType* contains beside the standard attributes of the BaseResponseType a sequence of *TransitionType* elements. Each *TransitionType* contains the transition point as a location element and two additional attributes timeDiff and wait, which are only relevant in the system EUSpirit, but not in DELFI.

3.5.5.5 Example

An example of the request for a *Transitions()* call is shown in Figure 55. The origin and destination location of the customers request are elements of the request. The co-ordinates and the AGS are the most relevant information for the local server. The local server is able to determine the best transition points for the desired whole itinerary with those two co-ordinates.

The response of *Transitions()* contains a set of locations which has to be used for the first computation of connections in the first phase (arrival time points in time at the destination transition points). Those nodes including a time label within a defined time range (upper time limit) will be used for further computations. This can cause a first reduction of further used transition points for the next computation steps. The structure *LocationType* for the transition points is often only filled with minimum information (location key) but not with the complete information like the location reply of the *Locations* response. In order to do the next computation, only the information of the *LocationKeyType* is necessary.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<TransitionsRequestType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" preferredLanguage="de" fallbackLanguage="en"
apiVersion="5.0.0.5" searchDirection="departure">
    <trans providerCode="1" providerType="delfi" sequence="81119746" subSequence="81120511"
xmlns="delfi5eng" />
    <startLocation name="Weinheim (Bergstraße) Museum" state="complete" xmlns="delfi5eng">
        <key providerCode="1" key="8226096:-1:1505:3476389:669945:NBWT" />
        <location name="Museum" regionName="Weinheim (Bergstraße)" regionId="8226096">
            <coord x="867270" y="4954684" />
        </location>
        <type>poi</type>
    </startLocation>
    <destLocation name="Dresden Universitätsklinikum" state="complete" xmlns="delfi5eng">
        <key providerCode="13" key="14612000:1:33000045:0:0:" />
        <location name="Dresden Universitätsklinikum" regionName="Dresden" regionId="14612000" />
        <type>stop</type>
    </destLocation>
    <handicapOptions xmlns="delfi5eng" />
</TransitionsRequestType>
```

Figure 55: Example Transitions() request

```
<?xml version="1.0" encoding="utf-8"?>
<TransitionsResponseType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" calcTime="0" returnCode="OK" apiVersion="5.0.0.5">
    <infoCode xmlns="delfi5eng">ok</infoCode>
    <infoMessage xmlns="delfi5eng" />
    <trans providerCode="1" providerType="delfi" sequence="80051968" subSequence="80053812"
xmlns="delfi5eng" />
    <transitions xmlns="delfi5eng">
        <location name="Weinheim, Bahnhof" state="complete">
            <key providerCode="0" key="8000377" />
            <location name="Bahnhof" regionName="Weinheim (Bergstraße)" regionId="08226096:999">
                <coord x="8665720" y="49553240" />
            </location>
            <type>stop</type>
        </location>
    </transitions>
    <transitions xmlns="delfi5eng">
        <location name="Heppenheim, Bahnhof" state="complete">
            <key providerCode="0" key="8002757" />
            <location name="Bahnhof" regionName="Heppenheim (Bergstraße)" regionId="06431011:30">
                <coord x="8633210" y="49641490" />
            </location>
            <type>stop</type>
        </location>
    </transitions>
    <transitions xmlns="delfi5eng">
        <location name="Bensheim, Bahnhof" state="complete">
            <key providerCode="0" key="8000031" />
            <location name="Bahnhof" regionName="Bensheim" regionId="06431002:10">
                <coord x="8616960" y="49681800" />
            </location>
            <type>stop</type>
        </location>
    </transitions>
    <transitions xmlns="delfi5eng">
        <location name="Heidelberg, Hbf" state="complete">
            <key providerCode="0" key="8000156" />
            <location name="Hauptbahnhof" regionName="Heidelberg" regionId="08221000:6">
                <coord x="8675840" y="49404470" />
            </location>
            <type>stop</type>
        </location>
    </transitions>
    <transitions xmlns="delfi5eng">
        <location name="Mannheim, Hbf" state="complete">
            <key providerCode="0" key="8000244" />
            <location name="Hauptbahnhof" regionName="Mannheim" regionId="08222000:18">
                <coord x="8469630" y="49479900" />
            </location>
```

```
            <type>stop</type>
        </location>
    </transitions>
    <transitions xmlns="delfi5eng">
        <location name="Darmstadt Hauptbahnhof" state="complete">
            <key providerCode="0" key="8000068" />
            <location name="Hauptbahnhof" regionName="Darmstadt" regionId="06411000:903">
                <coord x="0" y="0" />
            </location>
            <type>stop</type>
        </location>
    </transitions>
    …
</TransitionsResponseType>
```

Figure 56: Example Transitions() response

3.5.6 Operation PartialConnections()

The *PartialConnections()* operation is used to calculate the different steps of a distributed trip calculation.

3.5.6.1 Declaration

```
<wsdl:operation name="PartialConnections">
    <wsdlsoap:operation soapAction="delfi5eng/PartialConnections"/>
    <wsdl:input name="PartialConnectionsInput">
           <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="PartialConnectionsOutput">
           <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
```

3.5.6.2 Description

This operation calculates the partial connections and/or partial routes. Depending on the request attribute flags *calcXXXTimes*, connections and/or routes are calculated.

If connection times are requested, only the time when the arrival node can be reached (in the case of a departure search) will be returned. In case of an arrival search, the node of the latest departure time will be returned. This reduces the calculation time in the focus of the three phase computation, in especial. During the first and second phase, it is only necessary to know when a destination node can be reached (or when the start node must be left in case of a departure time search).

If route times are requested, the complete ride information including the necessary internal (transitions inside the partial itinerary computed by the requested system) transitions is delivered.

The data structure which has to be filled in order to make a request consists of the request direction, features selected by the customer (features are also called attributes and defines characteristics like barrier-free, bicycle transport allowed or similar), selected product types, start nodes (possible start stops at origin location for a departure search or start transition points for the intermediate system and the destination system), and a set of destination nodes (possible stop nodes at destination location for departure search or destination transition points). This request is complete when context and language information for the server to which the request is directed as well as optional types on information about the transport modes with which the customer reaches the start nodes or leaves the destination nodes are included.

If using this method call, beside origin- and destination nodes in form of station nodes, product selections can be defined for the search. Products can be excluded or included. Defaults of all product types are included. Features have also filter functionality. If set, only those rides shall be selected by the search which contain the selected attribute information. Therefore, the features are by default disabled. Additionally, submitting of context information is possible. The only supported search form is actually the three phase search. For the three phase search origin and destination nodes are closed.

The station node contains an additional parameter *wait* beside the location key of the stop and the connections. The term 'wait' will only be used with open nodes and defines the transition time which has to be used for a connection search. There are two states of connections: no connections have been computed so far. In that case, the list is empty and the value of 'wait' has to be 0 if the station will be reached by walking or a number which indicates the waiting time until the trip can be continued in the next system. If connections exist, 'wait' must have the value –1. This 'wait' is very relevant because it contains the changing time from one mode to another, mostly the changing time needed to walk from a local transport to trains or vice versa. The default value of *wait* is -1.

NOTE:

It has been agreed that local systems compute the passing time from a means of local transport to the station because local provider often have more detailed knowledge on the local situation and the time necessary for transitions which are mostly done by walking.

ConnectionInfoType contains the time which is either the arrival time (departure search) or the departure time (arrival search) together with the transition time to the meta stop (wait) for this ride.

If the returned destination nodes of *PartialConnections* are closed, there exists at least one connection to this destination node.

The destination nodes contain connection information which includes the information of the start nodes. At least a destination node can be found within a backward search for every destination node which is now a start node. The found destination node is reachable from the original start node. The return of origNode's (all or partially) can not be guaranteed.

The answer structure of *PartialConnections(calcRouteArrivalTimes)* and *PartialConnections(calcRouteDepartureTimes)* contains a set of *StationNodeTypes*. Every *StationNodeType* contains a destination location (element station) and a set of *ScheduleTypes* describing the itinerary parts of the rides to the destination location.

Finally, the element *ConnectionInfoType* contains information on waiting time, a number of legs, which are rides with transitions and the details of the route in the already described element *ScheduleType*.

It can be seen that only one stop is used for the start location but a set of transition points for the final forward search is used. Including those points in the forward search, it can be made sure that an even faster train starting at the start point is covered. This train must not be found in the first search phase when an earlier train starts at this stop node.

The answer delivers the partial routes for the final best path through the space - time relation spanned by the rides of the time table schedules.

In *mapLink,* a link to a map allows customers to view the situation at a specific location, i.e. for orientation for the transition. The complete information, i.e. the service type information, can be used for a comfortable output to the customer.

3.5.6.3 Request Type PartialConnectionsRequestType

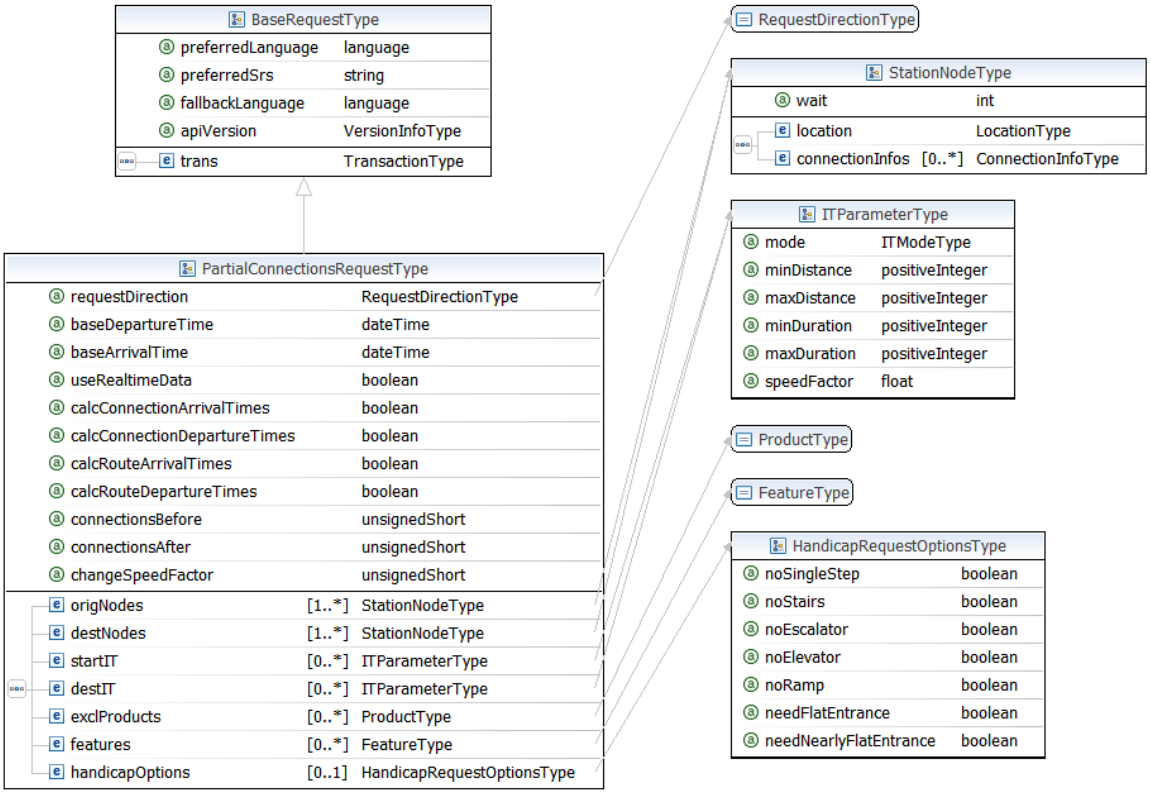

Figure 57: Class diagram PartialConnectionsRequestType

The *PartialConnections*() request type contains

- n *StationNodeType* elements for the origin locations
- m *StationNodeType* elements for the destination locations.
- flags of the requested information
  - *calcConnectionDepartureTimes*
  - *calcConnectionArrivalTimes*
  - *calcRouteDepartureTimes*
  - *calcRouteArrivalTimes*
- *ITParameterType* elements for origin and destination nodes
- excluded products
- required features
- required handicapOptions

The flags *calcXXXTimes* control which information the passive server should calculate.

Flag	**Description**
calcConnectionDepartureTimes	calculate backward the departure times for the origNodes (starting with the given connection times of

	the destNodes)
calcConnectionArrivalTimes	calculate forward the arrival times for the destNodes (starting with the given connection times of the origNodes)
calcRouteDepartureTimes	calculate backward the departure time and the detailed schedule elements for the origNodes (starting with the given connection times of the destNodes)
calcRouteArrivalTimes	calculate forward the arrival times and the detailed schedule elements for the destNodes (starting with the given connection times of the origNodes)

To avoid two consecutive *PartialConnections*() calls to the same passive server, these flags can be combined to request two informations in one call.

The following table is showing the possible combinations and their meaning:

Search Direction	**Flag**	**Description**
departure (forward)	*calcConnectionArrivalTimes* + *calcConnectionDepartureTimes*	called at passive server of destination region to calculate the earliest arrival time at the destination location and the latest departure times at the transition points of the destination region
departure (forward)	*calcConnectionDepartureTimes* + *calcRouteArrivalTimes*	called at passive server of origin region to calculate the latest departure time at the origin location and the detailed schedule elements to the transition points of the origin region
arrival (backward)	*calcConnectionDepartureTimes* + *calcConnectionArrivalTimes*	called at passive server of the origin region to calculate the latest departure time at the origin location and the earlist arrival time at the transition points of the origin region
arrival (backward)	*calcConnectionArrivalTimes* + *calcRouteDepartureTimes*	called at passive server of the destination region to calculate the earliest arrival time at the destination location and the latest departure time and the detailed schedule elements from the transition points of the destination region

The upper cases 2 and 4 are unique combinations of flags. To distinguish the cases 1 and 3, the passive server has to recognize the value of search direction. Search direction holds always the overall search direction, i.e. if the user gives a departure time the search direction is departure, otherwise arrival.

During the processing of combined calls the passive server can eliminate not possible

results in the second sub-step. But the search controller has to be aware, that the passive server does not eliminate those impossible results.

3.5.6.4 Response Type PartialConnectionsResponseType

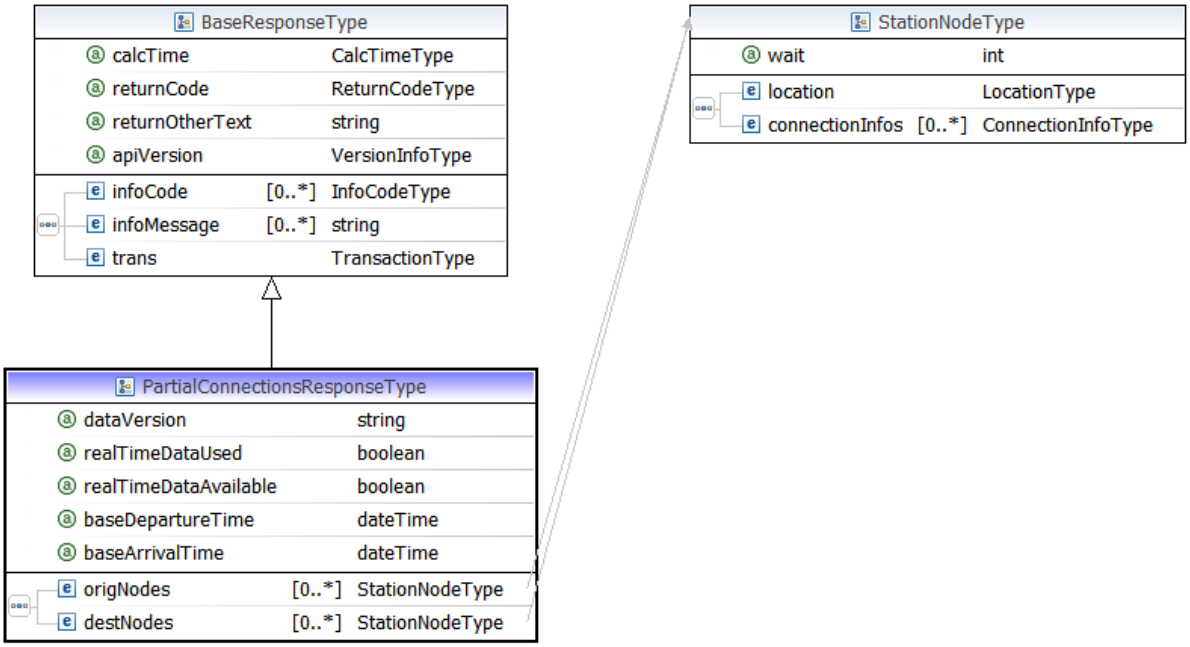

Figure 58: Class diagram PartialConnectionsResponseType

The *PartialConnectionsResponseType* contains the *StationNodeType* elements which are holding the requested information. If only one flag *calcXXXTimes* has been set in the request, only reached origNodes or destNodes are returned. If a combined call of two flags was requested both *StationNodeType* sequences (origNodes and destNodes) are returned for all reached nodes.

3.5.6.5 Example

```
<?xml version="1.0" encoding="utf-8"?>
<PartialConnectionsRequestType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" preferredLanguage="de" fallbackLanguage="en"
apiVersion="5.0.0.5" requestDirection="departure" calcConnectionArrivalTimes="true">
    <trans providerCode="1" providerType="delfi" sequence="81119746" subSequence="81120699"
xmlns="delfi5eng" />
    <origNodes wait="0" xmlns="delfi5eng">
        <location name="Dresden Universitätsklinikum" state="complete">
            <key providerCode="13" key="14612000:1:33000045:0:0:" />
            <location name="Dresden Universitätsklinikum" regionName="Dresden" regionId="14612000" />
            <type>stop</type>
        </location>
        <connectionInfos absoluteTime="2009-01-14T09:00:00" wait="0" legs="0" line="" />
    </origNodes>
    <destNodes wait="-1" xmlns="delfi5eng">
        <location name="Dresden Hauptbahnhof" state="complete">
            <key providerCode="0" key="8010085" />
            <location name="Hauptbahnhof" regionName="Dresden" regionId="14612000:1">
```

```
                <coord x="13732180" y="51040200" />
            </location>
            <type>stop</type>
        </location>
    </destNodes>
    <destNodes wait="-1" xmlns="delfi5eng">
        <location name="Dresden Bahnhof Neustadt" state="complete">
            <key providerCode="0" key="8010089" />
            <location name="Bahnhof Neustadt" regionName="Dresden" regionId="14612000:1">
                <coord x="13740880" y="51065430" />
            </location>
            <type>stop</type>
        </location>
    </destNodes>
.…
</PartialConnectionsResponseType>
```

Figure 59: Example PartialConnections() request

```
<?xml version="1.0" encoding="utf-8"?>
<PartialConnectionsResponseType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" calcTime="0.671875" returnCode="OK" apiVersion="5.0.0.5"
dataVersion="2009-01-13T18:07:47">
  <infoCode xmlns="delfi5eng">ok</infoCode>
  <infoMessage xmlns="delfi5eng" />
  <trans providerCode="1" providerType="delfi" sequence="81119746" subSequence="81120699"
xmlns="delfi5eng" />
  <destNodes xmlns="delfi5eng">
    <location name="Dresden Hauptbahnhof" state="complete">
      <key providerCode="0" key="8010085" />
      <location name="Hauptbahnhof" regionName="Dresden" regionId="14612000:1">
        <coord x="13732180" y="51040200" />
      </location>
      <type>stop</type>
    </location>
    <connectionInfos absoluteTime="2009-01-14T09:29:00" wait="5" legs="2" />
  </destNodes>
  <destNodes xmlns="delfi5eng">
    <location name="Dresden Hp Freiberger Straße" state="complete">
      <key providerCode="0" key="8011431" />
      <location name="Hp Freiberger Straße" regionName="Dresden" regionId="14612000:1">
        <coord x="13720130" y="51048620" />
      </location>
      <type>stop</type>
    </location>
    <connectionInfos absoluteTime="2009-01-14T09:33:00" wait="4" legs="2" />
  </destNodes>
  <destNodes xmlns="delfi5eng">
    <location name="Dresden Bahnhof Mitte" state="complete">
      <key providerCode="0" key="8013444" />
      <location name="Bahnhof Mitte" regionName="Dresden" regionId="14612000:1">
        <coord x="13723410" y="51055650" />
      </location>
      <type>stop</type>
    </location>
    <connectionInfos absoluteTime="2009-01-14T09:33:00" wait="5" legs="2" />
  </destNodes>
.…
</PartialConnectionsResponseType>
```

Figure 60: Example PartialConnections() response

3.5.7 Operation Connections()

The operation *Connections()* can be called at

- an active search controller to request a trip between two identified locations independent of the origin and destination region or
- a passive server to request a trip between two identified locations of the same region, the one of the passive server

3.5.7.1 Declaration

```
<wsdl:operation name="Connections">
    <wsdlsoap:operation soapAction="delfi5eng/Connections"/>
    <wsdl:input name="ConnectionsInput">
          <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="ConnectionsOutput">
          <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
```

3.5.7.2 Description

This method calculates complete itineraries between an origin and a destination stop. This method can be called directly from a DELFI server if the server is responsible for origin and destination or the method call to a search controller will be operated. A preferred provider for the long-distance part can be set but the real used provider is returned in this field. A search controller shall try to use the preferred long-distance provider but has to decide on its own whether this is possible or not.

The input data structure which has to be filled for requests consists of three types: the *ConnectionRequestType* which represents the input data of the customer for search and optimisation criteria, the *TransactionType* which contains the unique identification number for this transaction and, optionally, a provider key for the preferred long-distance information server.

Normally, product specifications are used to exclude some (often expensive) products from the search. Features act in the same way as filter but are usually not set which allows an unlimited search. To set feature filter or excluded products can cause an empty itinerary result if the requested feature is not available in the time table data or no alternatives to an excluded products are available.

A request can be directed to a given start time (*RequestDirectionType*.*departure*) or towards a given arrival time (*RequestDirectionType*.*arrival*). The search direction is a departure search by default.

Usage: *Connections()* encapsulates the complete itinerary. It is not allowed to display the calculated itineraries in parts only or to use the complete itinerary for other answers as a partial connection.

3.5.7.3 Request Type ConnectionsRequestType

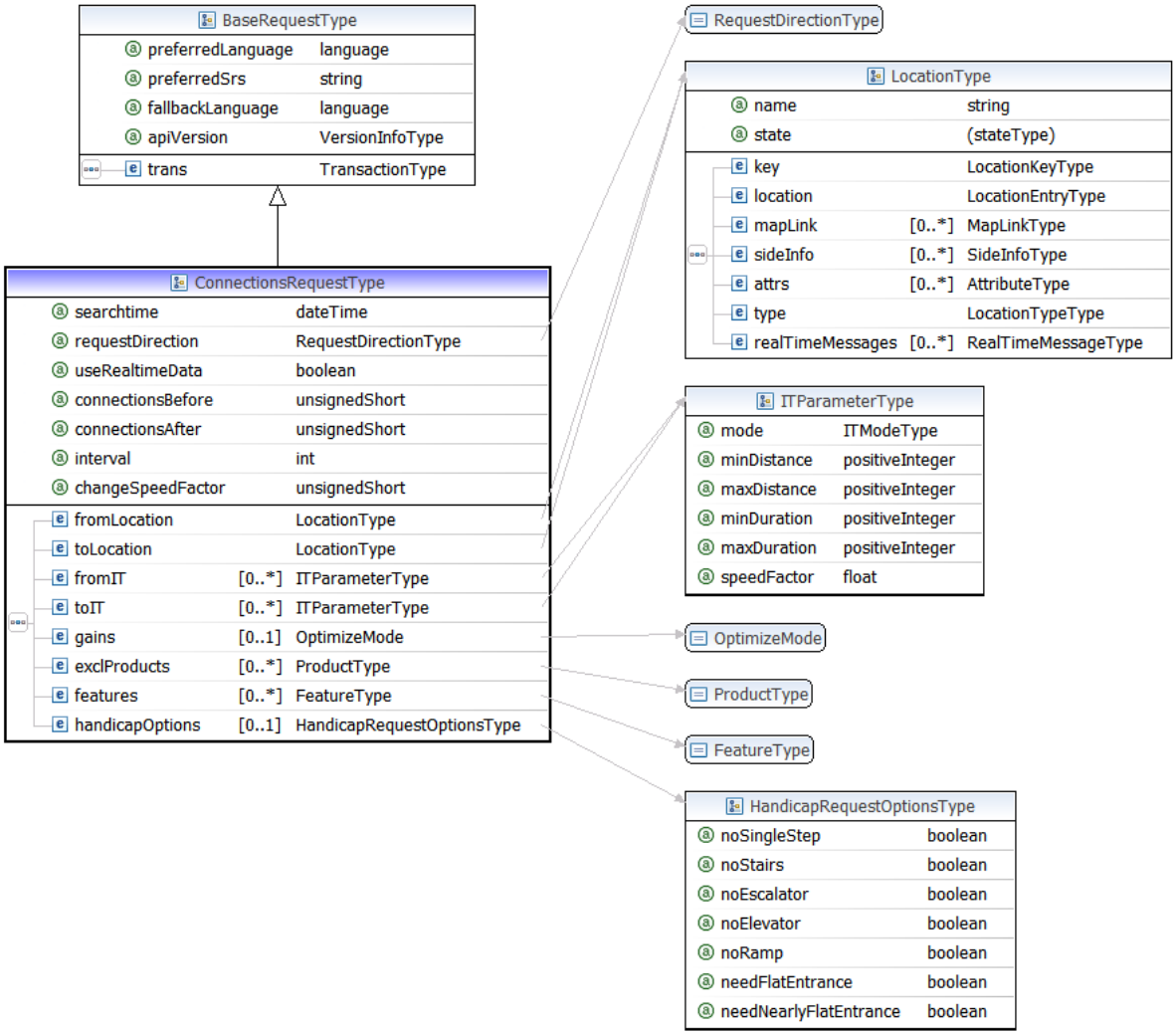

Figure 61: Class diagram ConnectionsRequestType

The *ConnectionsRequestType* contains additional attributes to define the search criterias for the trip like origin, destination, requested time (arrival or departure), restrictions for products or transfers by walk.

3.5.7.4 Response Type ConnectionsResponseType

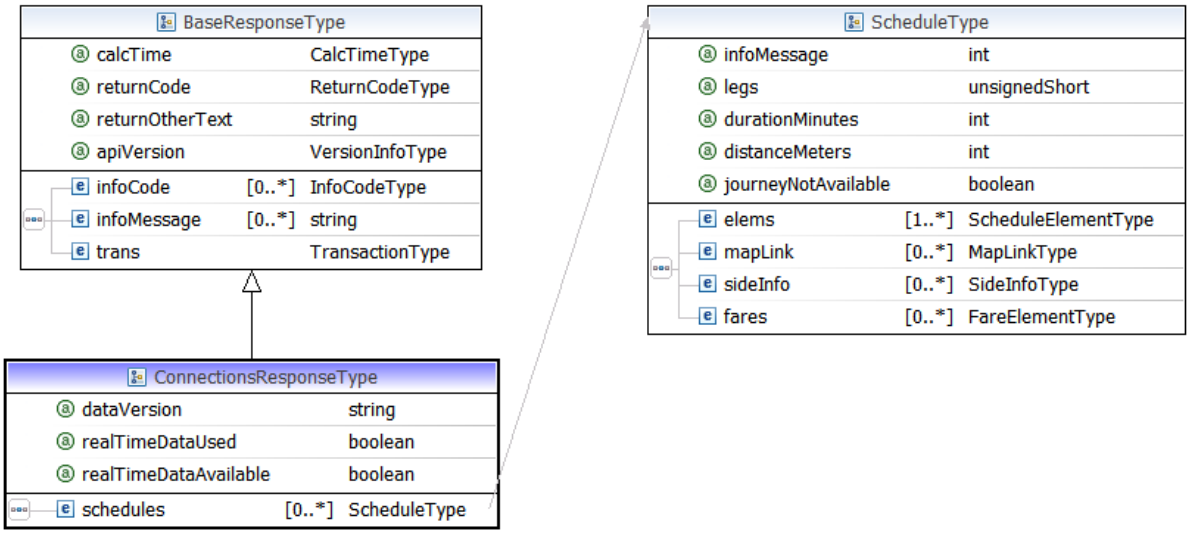

Figure 62: Class diagram ConnectionsResponseType

The *ConnectionsResponseType* contains the sequence of schedule elements (legs of the trip) together with enriched informations like coordinates, passed stops, links to maps, textual informations, etc.

3.5.7.5 Example

```
<?xml version="1.0" encoding="utf-8"?>
<ConnectionsRequestType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" preferredLanguage="de" preferredSrs=""
fallbackLanguage="en" apiVersion="5.0.0.5" searchtime="2009-01-14T09:00:00"
requestDirection="departure" connectionsBefore="0" connectionsAfter="1">
    <trans providerCode="9" providerType="delfi" sequence="152738723" xmlns="delfi5eng" />
    <fromLocation name="Ebsdorfergrund-Wittelsberg Markt" state="complete" xmlns="delfi5eng">
        <key providerCode="6" key="003013279-80" />
        <location name="" regionName="" regionId="06534008">
            <coord x="8855660" y="50760683" />
        </location>
        <type>stop</type>
    </fromLocation>
    <toLocation name="HG Rathaus" state="complete" xmlns="delfi5eng">
        <key providerCode="6" key="003031201-80" />
        <location name="" regionName="" regionId="06434001">
            <coord x="8622722" y="50224090" />
        </location>
        <type>stop</type>
    </toLocation>
    <handicapOptions xmlns="delfi5eng" />
</ConnectionsRequestType>
```

Figure 63: Example Connections() request

```
<?xml version="1.0" encoding="utf-8"?>
<ConnectionsResponseType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" calcTime="0.5" returnCode="OK">
    <trans providerCode="9" providerType="delfi" sequence="152738723" xmlns="delfi5eng" />
    <schedules legs="4" durationMinutes="138" xmlns="delfi5eng">
        <elems durationMinutes="26">
```

```
            <from name="Ebsdorfergrund-Wittelsberg Markt" state="complete" departureTime="2009-01-
14T12:19:00">
                <key providerCode="6" key="003013279-80" />
                <location name="Ebsdorfergrund-Wittelsberg Markt">
                    <coord x="8855660" y="50760683" />
                </location>
                <type>stop</type>
            </from>
            <passedLocations name="Ebsdorfergrund-Wittelsberg Markt" state="complete"
departureTime="2009-01-14T12:19:00" arrivalTime="2009-01-14T12:19:00">
                <key providerCode="6" key="003013279-80" />
                <location name="Ebsdorfergrund-Wittelsberg Markt">
                    <coord x="8855660" y="50760683" />
                </location>
                <type>stop</type>
            </passedLocations>
            <passedLocations name="Marburg-Moischt Kindergarten" state="complete" departureTime="2009-
01-14T12:23:00" arrivalTime="2009-01-14T12:23:00">
                <key providerCode="6" key="003013192-80" />
                <location name="Marburg-Moischt Kindergarten">
                    <coord x="8820898" y="50774068" />
                </location>
                <type>stop</type>
            </passedLocations>
.…
            <passedLocations name="Marburg Robert-Koch-Straße" state="complete" departureTime="2009-01-
14T12:40:00" arrivalTime="2009-01-14T12:40:00">
                <key providerCode="6" key="003019974-80" />
                <location name="Marburg Robert-Koch-Straße">
                    <coord x="8771754" y="50816416" />
                </location>
                <type>stop</type>
            </passedLocations>
            <passedLocations name="Marburg Hauptbahnhof" state="complete" arrivalTime="2009-01-
14T12:45:00">
                <key providerCode="6" key="003010011-80" />
                <location name="Marburg Hauptbahnhof">
                    <coord x="8773885" y="50818690" />
                </location>
                <type>stop</type>
            </passedLocations>
            <to name="Marburg Hauptbahnhof" state="complete" arrivalTime="2009-01-14T12:45:00">
                <key providerCode="6" key="003010011-80" />
                <location name="Marburg Hauptbahnhof">
                    <coord x="8773885" y="50818690" />
                </location>
                <type>stop</type>
            </to>
            <line operatorName="HN_W" product="Bus" trainName="MR-54" direction="Marburg Hauptbahnhof"
code="20" />
            <providerCodes>6</providerCodes>
        </elems>
        <elems durationMinutes="3">
            <from name="Marburg Hauptbahnhof" state="complete" departureTime="2009-01-14T12:45:00">
                <key providerCode="6" key="003010011-80" />
                <location name="Marburg Hauptbahnhof">
                    <coord x="8773885" y="50818690" />
                </location>
                <type>stop</type>
            </from>
            <to name="Marburg Bahnhof" state="complete" arrivalTime="2009-01-14T12:48:00">
                <key providerCode="6" key="003011081-80" />
                <location name="Marburg Bahnhof">
                    <coord x="8774721" y="50819679" />
                </location>
                <type>stop</type>
            </to>
            <line code="40" />
            <providerCodes>6</providerCodes>
        </elems>
        <elems durationMinutes="70">
            <from name="Marburg Bahnhof" state="complete" departureTime="2009-01-14T12:49:00"
track="4">
.…
```

```
</ConnectionsResponseType>
```

Figure 64: Example Connections() response

3.5.8 Operation RefineConnections()

The operation *RefineConnections()* can add information to an already calculated trip.

3.5.8.1 Declaration

```
<wsdl:operation name="RefineConnections">
    <wsdlsoap:operation soapAction="delfi5eng/RefineConnections"/>
    <wsdl:input name="RefineConnectionsInput">
           <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="RefineConnectionsOutput">
           <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
```

3.5.8.2 Description

The operation *RefineConnections()* can be called to request additional information like ticket booking links or overview map links to already calculated itineraries. It can be called for distributed calculated trips (with *PartialConnections()*) as for pooled caculated trips (with *Connections()*).

The responding server adds if possible information to the schedule elements and returns them in his response.

3.5.8.3 Request Type RefineConnectionsRequestType

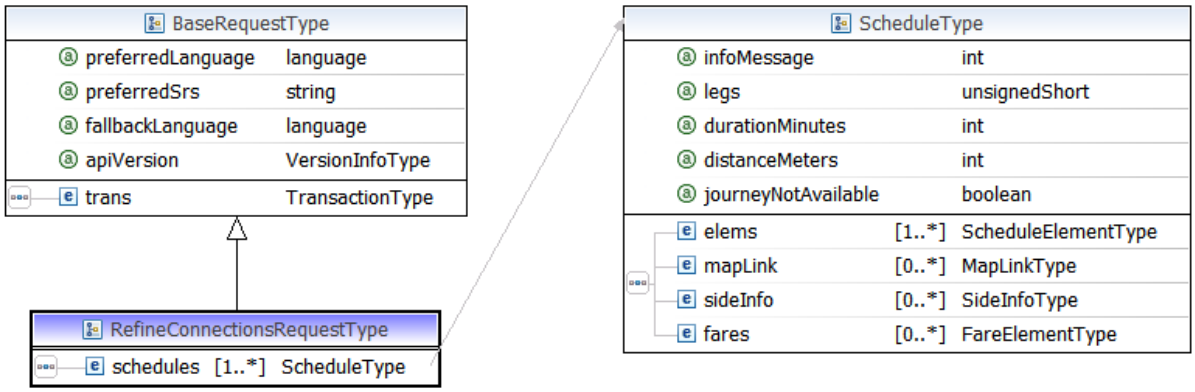

Figure 65: Class diagram RefineConnectionsRequestType

The *RefineConnectionsRequestType* contains the sequence of already calculated schedule elements.

3.5.8.4 Response Type RefineConnectionsResponseType

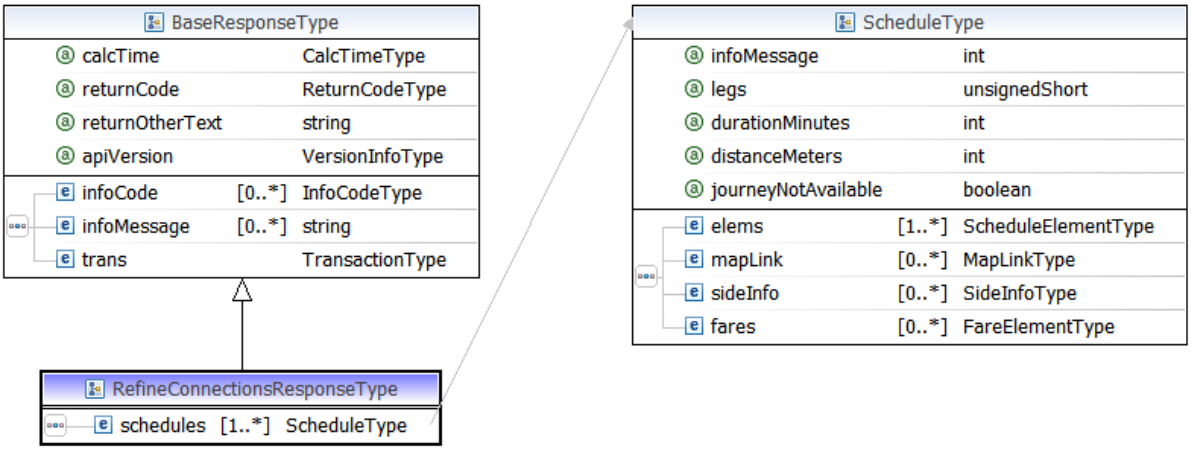

Figure 66: Class diagram RefineConnectionsResponseType

The *RefineConnectionsResponseType* contains the enriched schedule elements of the request. Even if no information was added, the schedule element of the request structure should be returned as it is.

3.5.8.5 Example

```
<?xml version="1.0" encoding="utf-8"?>
<RefineConnectionsRequestType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" preferredLanguage="de" fallbackLanguage="en"
apiVersion="5.0.0.5">
  <trans providerCode="99" providerType="delfi" sequence="104601361" xmlns="delfi5eng" />
  <schedules legs="3" durationMinutes="101" xmlns="delfi5eng">
    <elems durationMinutes="2">
      <from name="Chemnitz Theaterplatz" state="complete" departureTime="2009-01-15T09:45:00">
```

```
        <key providerCode="13" key="14161000:-1:36030004:4565315:526046:NAV4" />
        <location topLevelRegionName="" topLevelRegionId="" name="Theaterplatz" regionName="Dresden"
regionId="14161000">
          <coord x="12925730" y="50837310" srs="" />
        </location>
        <mapLink url="http://195.243.251.84:8888/vvo/FILELOAD?Filename=vvo_496F05D00.pdf"
title="Umgebungsplan" language="de" mimeType="application/pdf" />
        <type>stop</type>
      </from>
      <passedLocations name="Chemnitz, Theaterplatz" state="complete" departureTime="2009-01-
15T09:45:00">
        <key providerCode="13" key="14511000:1:36030004:4565316:526046:NAV4" />
        <location name="Theaterplatz" regionName="Chemnitz" regionId="14511000:1">
          <coord x="12925740" y="50837310" />
        </location>
        <type>stop</type>
      </passedLocations>
      <passedLocations name="Chemnitz, Schillerplatz" state="complete">
        <key providerCode="13" key="14511000:1:36030063:4565400:525822:NAV4" />
        <location regionName="Chemnitz" regionId="14511000:1">
          <coord x="12926980" y="50839320" />
        </location>
        <type>stop</type>
      </passedLocations>
      <passedLocations name="Chemnitz, Hauptbahnhof" state="complete" arrivalTime="2009-01-
15T09:47:00">
        <key providerCode="13" key="14511000:1:36030062:4565712:525826:NAV4" />
        <location name="Hauptbahnhof" regionName="Chemnitz" regionId="14511000:1">
          <coord x="12931400" y="50839250" />
        </location>
        <type>stop</type>
      </passedLocations>
      <to name="Chemnitz, Hauptbahnhof" state="complete" arrivalTime="2009-01-15T09:47:00">
        <key providerCode="13" key="14511000:1:36030062:4565558:525944:NAV4" />
        <location name="Hauptbahnhof" regionName="Chemnitz" regionId="14511000:1">
          <coord x="12929200" y="50838200" />
        </location>
        <mapLink url="http://195.243.251.84:8888/vvo/FILELOAD?Filename=vvo_496F05D01.pdf"
title="Umgebungsplan" language="de" mimeType="application/pdf" />
        <type>stop</type>
      </to>
      <line operatorName="City-Bahn Chemnitz" product="CB" trainName="56016" trainString="56016"
direction="Chemnitz, Hauptbahnhof" lastStop="Chemnitz, Hauptbahnhof" code="2">
        <sideInfo techtype="" businesstype="EFAEX" addtechspec="" addbusinessspec="" content="6;;City-
Bahn Chemnitz;56016;56016;" />
      </line>
      <attrs attrMandatory="true" sortOrder="0" textCode="" plainString="Fahrradmitnahme begrenzt
möglich" language="de" />
      <attrs attrMandatory="true" sortOrder="0" textCode="" plainString="nur 2. Klasse" language="de"
/>
      <attrs attrMandatory="true" sortOrder="0" textCode="" plainString="City Bahn Chemnitz"
language="de" />
      <providerCodes>13</providerCodes>
    </elems>
…
  </schedules>
</RefineConnectionsRequestType>
```

Figure 67: Example Refine Connections() request

```
<?xml version="1.0" encoding="utf-8"?>
<RefineConnectionsResponseType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" calcTime="0.005765145" returnCode="OK"
apiVersion="5.0.0.5">
  <infoCode xmlns="delfi5eng">ok</infoCode>
  <infoMessage xmlns="delfi5eng" />
  <trans providerCode="99" providerType="delfi" sequence="104601361" xmlns="delfi5eng" />
  <schedules legs="3" durationMinutes="101" xmlns="delfi5eng">
    <elems durationMinutes="2">
```

```
      <from name="Chemnitz Theaterplatz" state="complete" departureTime="2009-01-15T09:45:00">
        <key providerCode="13" key="14161000:-1:36030004:4565315:526046:NAV4" />
        <location topLevelRegionName="" topLevelRegionId="" name="Theaterplatz" regionName="Dresden"
regionId="14161000">
          <coord x="12925730" y="50837310" srs="" />
        </location>
        <mapLink url="http://195.243.251.84:8888/vvo/FILELOAD?Filename=vvo_496F05D00.pdf"
title="Umgebungsplan" language="de" mimeType="application/pdf" />
        <type>stop</type>
      </from>
      <passedLocations name="Chemnitz, Theaterplatz" state="complete" departureTime="2009-01-
15T09:45:00">
        <key providerCode="13" key="14511000:1:36030004:4565316:526046:NAV4" />
        <location name="Theaterplatz" regionName="Chemnitz" regionId="14511000:1">
          <coord x="12925740" y="50837310" />
        </location>
        <type>stop</type>
      </passedLocations>
      <passedLocations name="Chemnitz, Schillerplatz" state="complete">
        <key providerCode="13" key="14511000:1:36030063:4565400:525822:NAV4" />
        <location regionName="Chemnitz" regionId="14511000:1">
          <coord x="12926980" y="50839320" />
        </location>
        <type>stop</type>
      </passedLocations>
      <passedLocations name="Chemnitz, Hauptbahnhof" state="complete" arrivalTime="2009-01-
15T09:47:00">
        <key providerCode="13" key="14511000:1:36030062:4565712:525826:NAV4" />
        <location name="Hauptbahnhof" regionName="Chemnitz" regionId="14511000:1">
          <coord x="12931400" y="50839250" />
        </location>
        <type>stop</type>
      </passedLocations>
      <to name="Chemnitz, Hauptbahnhof" state="complete" arrivalTime="2009-01-15T09:47:00">
        <key providerCode="13" key="14511000:1:36030062:4565558:525944:NAV4" />
        <location name="Hauptbahnhof" regionName="Chemnitz" regionId="14511000:1">
          <coord x="12929200" y="50838200" />
        </location>
        <mapLink url="http://195.243.251.84:8888/vvo/FILELOAD?Filename=vvo_496F05D01.pdf"
title="Umgebungsplan" language="de" mimeType="application/pdf" />
        <type>stop</type>
      </to>
      <line operatorName="City-Bahn Chemnitz" product="CB" trainName="56016" trainString="56016"
direction="Chemnitz, Hauptbahnhof" lastStop="Chemnitz, Hauptbahnhof" code="2">
        <sideInfo techtype="" businesstype="EFAEX" addtechspec="" addbusinessspec="" content="6;;City-
Bahn Chemnitz;56016;56016;" />
      </line>
      <attrs attrMandatory="true" sortOrder="0" textCode="" plainString="Fahrradmitnahme begrenzt
möglich" language="de" />
      <attrs attrMandatory="true" sortOrder="0" textCode="" plainString="nur 2. Klasse" language="de"
/>
      <attrs attrMandatory="true" sortOrder="0" textCode="" plainString="City Bahn Chemnitz"
language="de" />
      <providerCodes>13</providerCodes>
    </elems>
…
  </schedules>
</RefineConnectionsResponseType>
```

Figure 68: Example Refine Connections() response

3.5.9 Operation Multi()

The operation *Multi()* is defined to optimize transmission traffic between a requesting server and the requested server.

3.5.9.1 Declaration

```
<wsdl:operation name="Multi">
    <wsdlsoap:operation soapAction="delfi5eng/Multi"/>
    <wsdl:input name="MultiInput">
          <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="MultiOutput">
          <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
```

3.5.9.2 Description

The operation *Multi()* is defined to optimize transmission traffic between a requesting server and the requested server. Instead of calling n single requests, one *Multi()* request can be used to transport the n requests and the n responses. This reduces significantly the overhead to establish the connection, sending the request data, receiving the response data and close the connection. Instead of n connections, only 1 connection is needed, so n-1 connections are saved.

The current definition allows only the frequently called request types to be included in a *Multi()* operation.

3.5.9.3 Request Type MultiRequestType

Figure 69 Class diagram MultiRequestType

The *MultiRequestType* contains sequences of

- *LocationRequestType* elements
- *TransitionRequestType* elements
- *PartialConnectionsRequestType* elements
- *ConnectionsRequestType* elements and
- *RefineConnectionsRequestType* elements.

3.5.9.4 Response Type MultiResponseType

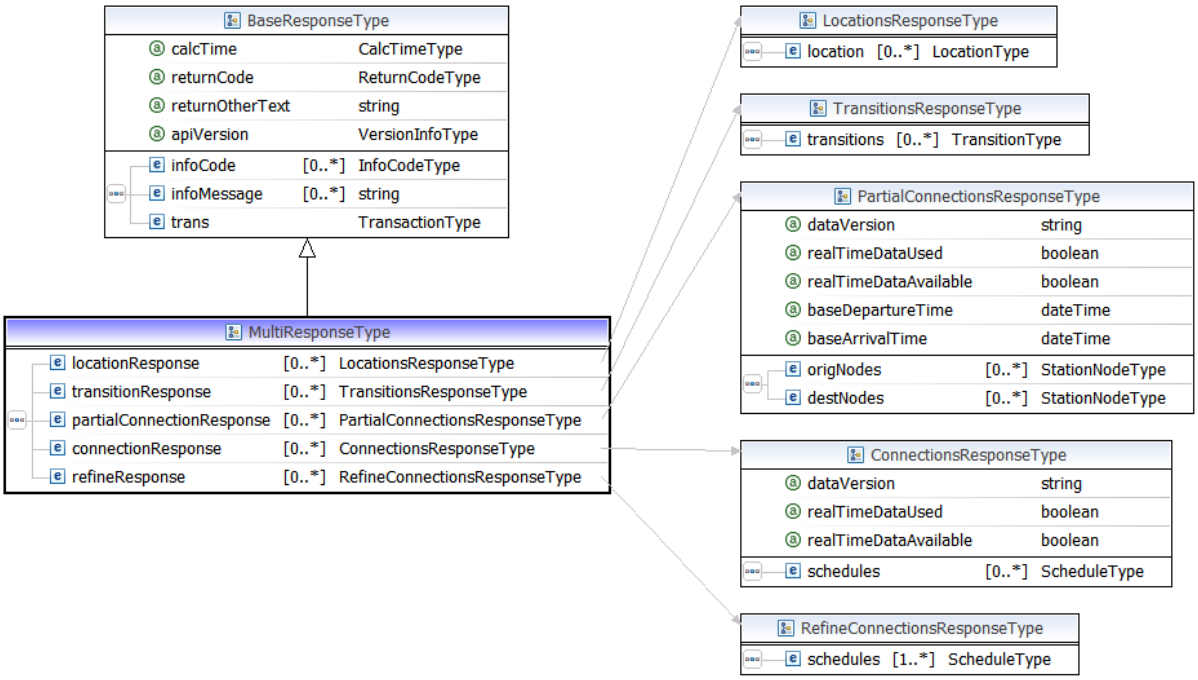

Figure 70 Class diagram MultiResponseType

The *MultiResponseType* contains all responses of all sub-requests. Therefore this type contains sequences of

- *LocationResponseType* elements
- *TransitionResponseType* elements
- *PartialConnectionsResponseType* elements
- *ConnectionsResponseType* elements and
- *RefineConnectionsResponseType* elements.

3.5.9.5 Example

Below an example of a *Multi*() request with

- 2 requests of *Locations*()
- 1 request of *Transitions*()
- 1 request of *Connections*()

```
<?xml version="1.0" encoding="utf-8"?>
<MultiRequestType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" preferredLanguage="de" preferredSrs=""
fallbackLanguage="en" apiVersion="5.0.0.5">
```

```
  <trans providerCode="9" providerType="delfi" sequence="184221119" xmlns="delfi5eng" />
  <locationRequest preferredLanguage="de" preferredSrs="" maxLocationNum="100" xmlns="delfi5eng">
    <trans providerCode="9" providerType="delfi" sequence="184221119" />
    <entry name="Theater" state="initial">
      <key providerCode="1" key="" />
      <location topLevelRegionName="" topLevelRegionId="" name="Theater" regionName="" regionId="" />
      <type>null</type>
    </entry>
    <type>stop</type>
    <type>address</type>
    <type>poi</type>
  </locationRequest>
  <locationRequest preferredLanguage="de" preferredSrs="" maxLocationNum="100" xmlns="delfi5eng">
    <trans providerCode="9" providerType="delfi" sequence="184221119" />
    <entry name="Museum" state="initial">
      <key providerCode="1" key="" />
      <location topLevelRegionName="" topLevelRegionId="" name="Museum" regionName="" regionId="" />
      <type>null</type>
    </entry>
    <type>stop</type>
    <type>address</type>
    <type>poi</type>
  </locationRequest>
  <transitionRequest preferredLanguage="de" preferredSrs="" searchDirection="departure"
xmlns="delfi5eng">
    <trans providerCode="9" providerType="delfi" sequence="184221119" />
    <startLocation name="Stuttgart Theater Rampe" state="complete">
      <key providerCode="1" key="8111000:-1:968:3512616:757066:NBWT" />
      <location topLevelRegionName="" topLevelRegionId="" name="Stuttgart Theater Rampe"
regionName="Stuttgart" regionId="8111000" />
      <type>poi</type>
    </startLocation>
    <destLocation name="Dresden Theater Oben" state="complete">
      <key providerCode="13" key="14262000:-1:130:4621554:501103:NAV4" />
      <location topLevelRegionName="" topLevelRegionId="" name="Theater Oben" regionName="Dresden"
regionId="14262000" />
      <type>poi</type>
    </destLocation>
  </transitionRequest>
  <connectionRequest preferredLanguage="de" preferredSrs="" searchtime="2009-01-14T09:00:00"
requestDirection="departure" xmlns="delfi5eng">
    <trans providerCode="9" providerType="delfi" sequence="184221119" />
    <fromLocation name="Stuttgart Theater Rampe" state="complete">
      <key providerCode="1" key="8111000:-1:968:3512616:757066:NBWT" />
      <location topLevelRegionName="" topLevelRegionId="" name="Stuttgart Theater Rampe"
regionName="Stuttgart" regionId="8111000" />
      <type>poi</type>
    </fromLocation>
    <toLocation name="Wanderweg" state="complete">
      <key providerCode="1" key="8116078:-1:5007105:3509790:764910:NBWT" />
      <location topLevelRegionName="" topLevelRegionId="" name="Wanderweg" regionName="Musberg"
regionId="8116078" />
      <type>stop</type>
    </toLocation>
    <handicapOptions />
  </connectionRequest>
</MultiRequestType>
```

Figure 71: Example Multi() request

```
<?xml version="1.0" encoding="utf-8"?>
<MultiResponseType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" calcTime="96.921875" returnCode="OK" apiVersion="5.0.0.5">
  <infoCode xmlns="delfi5eng">ok</infoCode>
  <infoMessage xmlns="delfi5eng" />
  <trans providerCode="9" providerType="delfi" sequence="184221119" xmlns="delfi5eng" />
  <locationResponse calcTime="15.78125" returnCode="OK" apiVersion="5.0.0.5" xmlns="delfi5eng">
    <infoCode>ok</infoCode>
    <infoMessage />
```

```
    <trans providerCode="9" providerType="delfi" sequence="184221119" />
    <location name="Stuttgart Kelley Theater" state="complete">
      <key providerCode="1" key="8111000:-1:3743:3513131:761631:NBWT" />
      <location name="Kelley Theater" regionName="Stuttgart" regionId="8111000">
        <coord x="9177440" y="48722740" />
      </location>
      <type>poi</type>
    </location>
    <location name="Stuttgart Theater Tredeschin" state="complete">
      <key providerCode="1" key="8111000:-1:3519:3514891:754695:NBWT" />
      <location name="Theater Tredeschin" regionName="Stuttgart" regionId="8111000">
        <coord x="9201610" y="48785070" />
      </location>
      <type>poi</type>
    </location>
    <location name="Stuttgart Theater im Wirtshaus" state="complete">
      <key providerCode="1" key="8111000:-1:3034:3511903:756253:NBWT" />
      <location name="Theater im Wirtshaus" regionName="Stuttgart" regionId="8111000">
        <coord x="9160910" y="48771120" />
      </location>
      <type>poi</type>
    </location>
    <location name="Stuttgart Theater Tri-Bühne" state="complete">
      <key providerCode="1" key="8111000:-1:971:3513034:756031:NBWT" />
      <location name="Theater Tri-Bühne" regionName="Stuttgart" regionId="8111000">
        <coord x="9176300" y="48773100" />
      </location>
      <type>poi</type>
    </location>
    <location name="Ulm Altes Theater" state="complete">
      <key providerCode="1" key="8421000:-1:14057:3572686:797691:NBWT" />
      <location name="Altes Theater" regionName="Ulm" regionId="8421000">
        <coord x="9980500" y="48394410" />
      </location>
      <type>poi</type>
    </location>
    <location name="Heidelberg Theaterstraße" state="complete">
      <key providerCode="1" key="8221000:-1:1000001107:3478583:685054:NBWT" />
      <location name="Theaterstraße" regionName="Heidelberg" regionId="8221000">
        <coord x="8703840" y="49411070" />
      </location>
      <type>address</type>
    </location>
  </locationResponse>
  <locationResponse calcTime="19.046875" returnCode="OK" apiVersion="5.0.0.5" xmlns="delfi5eng">
    <infoCode>ok</infoCode>
    <infoMessage />
    <trans providerCode="9" providerType="delfi" sequence="184221119" />
    <location name="Creglingen Fingerhut Museum" state="complete">
      <key providerCode="1" key="8128020:-1:5687:3574808:679223:NBWT" />
      <location name="Fingerhut Museum" regionName="Creglingen" regionId="8128020">
        <coord x="10030860" y="49459280" />
      </location>
      <type>poi</type>
    </location>
    <location name="Waldenbuch Museum Ritter" state="complete">
      <key providerCode="1" key="8115048:-1:4367:3509132:770968:NBWT" />
      <location name="Museum Ritter" regionName="Waldenbuch" regionId="8115048">
        <coord x="9122890" y="48638850" />
      </location>
      <type>poi</type>
    </location>
    <location name="Stuttgart Mercedes-Benz Museum" state="complete">
      <key providerCode="1" key="8111000:-1:887:3517245:754311:NBWT" />
      <location name="Mercedes-Benz Museum" regionName="Stuttgart" regionId="8111000">
        <coord x="9233660" y="48788460" />
      </location>
      <type>poi</type>
    </location>
    <location name="Tübingen Hotel Museum" state="complete">
      <key providerCode="1" key="8416041:-1:2147133375:3504356:783944:NBWT" />
      <location name="Hotel Museum" regionName="Tübingen" regionId="8416041">
        <coord x="9057950" y="48522210" />
      </location>
```

```
      <type>poi</type>
    </location>
    <location name="Tübingen Kino Museum" state="complete">
      <key providerCode="1" key="8416041:-1:2147133371:3504320:783927:NBWT" />
      <location name="Kino Museum" regionName="Tübingen" regionId="8416041">
        <coord x="9057460" y="48522360" />
      </location>
      <type>poi</type>
    </location>
  </locationResponse>
  <transitionResponse calcTime="16.453125" returnCode="OK" apiVersion="5.0.0.5" xmlns="delfi5eng">
    <infoCode>ok</infoCode>
    <infoMessage />
    <trans providerCode="9" providerType="delfi" sequence="184221119" />
    <transitions>
      <location name="Stuttgart ZOB Stuttgart Hbf" state="complete">
        <key providerCode="0" key="8089319" />
        <location name="ZOB Stuttgart Hbf" regionName="Stuttgart" regionId="08111000:51">
          <coord x="9186310" y="48784060" />
        </location>
        <type>stop</type>
      </location>
    </transitions>
    <transitions>
      <location name="Stuttgart Hauptbahnhof (oben)" state="complete">
        <key providerCode="0" key="8000096" />
        <location name="Hauptbahnhof (oben)" regionName="Stuttgart" regionId="08111000:51">
          <coord x="9183950" y="48783790" />
        </location>
        <type>stop</type>
      </location>
    </transitions>
    <transitions>
      <location name="Flughafen/Messe" state="complete">
        <key providerCode="0" key="8005768" />
        <location name="Flughafen/Messe" regionName="Echterdingen" regionId="08116078:1">
          <coord x="9194260" y="48691400" />
        </location>
        <type>stop</type>
      </location>
    </transitions>
    <transitions>
      <location name="Ludwigsburg" state="complete">
        <key providerCode="0" key="8000235" />
        <location name="Ludwigsburg" regionName="Ludwigsburg" regionId="08118048:4">
          <coord x="9188260" y="48891400" />
        </location>
        <type>stop</type>
      </location>
    </transitions>
    <transitions>
      <location name="Plochingen" state="complete">
        <key providerCode="0" key="8000302" />
        <location name="Plochingen" regionName="Plochingen" regionId="08116056:1">
          <coord x="9411470" y="48712880" />
        </location>
        <type>stop</type>
      </location>
    </transitions>
  <connectionResponse calcTime="0.984375" returnCode="OK" apiVersion="5.0.0.5" xmlns="delfi5eng">
    <infoCode>ok</infoCode>
    <infoMessage />
    <trans providerCode="9" providerType="delfi" sequence="184221119" />
    <schedules legs="3" durationMinutes="40">
      <elems elementDistance="159" durationMinutes="4">
        <from name="Stuttgart Theater Rampe" state="complete" departureTime="2009-01-14T09:04:00">
          <key providerCode="1" key="8111000:-1:968:3512616:757066:NBWT" />
          <location topLevelRegionName="" topLevelRegionId="" name="Stuttgart Theater Rampe"
regionName="Stuttgart" regionId="8111000" />
          <type>poi</type>
        </from>
        <passedLocations name="Stuttgart Marienplatz" state="complete">
          <key providerCode="1" key="8111000:51:5006019:3512405:756984:NBWT" />
          <location name="Marienplatz" regionName="Stuttgart Süd" regionId="08111000:51">
```

```
            <coord x="9167720" y="48764540" />
          </location>
          <type>stop</type>
        </passedLocations>
        <to name="Stuttgart Marienplatz" state="complete" arrivalTime="2009-01-14T09:08:00">
          <key providerCode="1" key="8111000:55:5006019:3512434:757004:NBWT" />
          <location name="Marienplatz" regionName="Stuttgart Süd" regionId="08111000:55">
            <coord x="9168110" y="48764360" />
          </location>
          <mapLink url="http://www2.vvs.de/Download/Envmaps/vvs/umarien.pdf" title="Haltestellenplan"
language="de" mimeType="application/pdf" />
          <mapLink url="http://192.168.32.18:8888/vvo/FILELOAD?Filename=bw-ww0_496E244C0.pdf"
title="Umgebungsplan" language="de" mimeType="application/pdf" />
          <type>stop</type>
        </to>
        <line operatorName="" product="Fussweg" trainName="" trainString="" direction="" lastStop=""
code="40">
          <sideInfo techtype="" businesstype="EFAEX" addtechspec="" addbusinessspec=""
content="100;Fussweg;;;;" />
        </line>
        <transfer transferType="road" name="Alte Weinsteige" duration="10" distance="12"
turnDirectionBefore="0" turnInstructionBefore="">
          <coord x="9170580" y="48763800" />
          <coord x="9170390" y="48763870" />
          <coord x="9170470" y="48763970" />
        </transfer>
        <transfer transferType="road" name="Filderstraße" duration="50" distance="68"
turnDirectionBefore="225" turnInstructionBefore="">
          <coord x="9170470" y="48763970" />
          <coord x="9169730" y="48763860" />
          <coord x="9169560" y="48763850" />
        </transfer>
        <transfer transferType="road" name="Böheimstraße" duration="50" distance="59"
turnDirectionBefore="0" turnInstructionBefore="">
          <coord x="9169560" y="48763850" />
          <coord x="9168950" y="48763780" />
          <coord x="9168770" y="48763760" />
        </transfer>
        <transfer transferType="road" name="Filderstraße" duration="28" distance="20"
turnDirectionBefore="0" turnInstructionBefore="">
          <coord x="9168770" y="48763760" />
          <coord x="9168320" y="48763690" />
          <coord x="9168110" y="48764360" />
        </transfer>
        <providerCodes>1</providerCodes>
      </elems>
      <elems durationMinutes="3">
        <from name="Stuttgart Marienplatz" state="complete" departureTime="2009-01-14T09:08:00">
          <key providerCode="1" key="8111000:55:5006019:3512434:757004:NBWT" />
          <location name="Marienplatz" regionName="Stuttgart Süd" regionId="08111000:55">
            <coord x="9168110" y="48764360" />
          </location>
          <mapLink url="http://www2.vvs.de/Download/Envmaps/vvs/umarien.pdf" title="Haltestellenplan"
language="de" mimeType="application/pdf" />
          <mapLink url="http://192.168.32.18:8888/vvo/FILELOAD?Filename=bw-ww0_496E244C1.pdf"
title="Umgebungsplan" language="de" mimeType="application/pdf" />
          <type>stop</type>
        </from>
        <passedLocations name="Stuttgart Marienplatz" state="complete" departureTime="2009-01-
14T09:08:00">
          <key providerCode="1" key="8111000:51:5006019:3512405:756984:NBWT" />
          <location name="Marienplatz" regionName="Stuttgart Süd" regionId="08111000:51">
            <coord x="9167720" y="48764540" />
          </location>
          <type>stop</type>
        </passedLocations>
        <passedLocations name="Stuttgart Mörikestraße" state="complete" departureTime="2009-01-
14T09:09:00" arrivalTime="2009-01-14T09:09:00">
          <key providerCode="1" key="8111000:51:5002409:3512491:756572:NBWT" />
          <location regionName="Stuttgart" regionId="08111000:51">
            <coord x="9168900" y="48768240" />
          </location>
          <type>stop</type>
        </passedLocations>
```

```
        <passedLocations name="Stuttgart Marien-/Silberburgstraße" state="complete"
departureTime="2009-01-14T09:10:00" arrivalTime="2009-01-14T09:10:00">
          <key providerCode="1" key="8111000:51:5002410:3512508:756315:NBWT" />
          <location regionName="Stuttgart" regionId="08111000:51">
            <coord x="9169140" y="48770550" />
          </location>
          <type>stop</type>
        </passedLocations>
        <passedLocations name="Stuttgart Feuersee" state="complete" arrivalTime="2009-01-14T09:11:00">
          <key providerCode="1" key="8111000:51:5006221:3512367:756017:NBWT" />
          <location name="Feuersee" regionName="Stuttgart West" regionId="08111000:51">
            <coord x="9167230" y="48773240" />
          </location>
          <type>stop</type>
        </passedLocations>
        <to name="Stuttgart Feuersee" state="complete" arrivalTime="2009-01-14T09:11:00">
          <key providerCode="1" key="8111000:56:5006221:3512281:756076:NBWT" />
          <location name="Feuersee" regionName="Stuttgart West" regionId="08111000:56">
            <coord x="9166060" y="48772710" />
          </location>
          <mapLink url="http://www2.vvs.de/Download/Envmaps/vvs/ufeuerse.pdf" title="Haltestellenplan"
language="de" mimeType="application/pdf" />
          <mapLink url="http://192.168.32.18:8888/vvo/FILELOAD?Filename=bw-ww0_496E244C2.pdf"
title="Umgebungsplan" language="de" mimeType="application/pdf" />
          <type>stop</type>
        </to>
        <line operatorName="SSB" product="Bus" trainName="41" trainString="0" direction="Stuttgart
Berliner Platz (Liederhalle)" lastStop="Stuttgart Berliner Platz (Liederhalle)" code="20">
          <sideInfo techtype="" businesstype="EFAEX" addtechspec="" addbusinessspec=""
content="3;Bus;;0;41;41" />
        </line>
        <providerCodes>1</providerCodes>
      </elems>
      <elems durationMinutes="17">
        <from name="Stuttgart Feuersee" state="complete" departureTime="2009-01-14T09:19:00">
          <key providerCode="1" key="8111000:56:5006221:3512281:756076:NBWT" />
          <location name="Feuersee" regionName="Stuttgart West" regionId="08111000:56">
            <coord x="9166060" y="48772710" />
          </location>
          <mapLink url="http://www2.vvs.de/Download/Envmaps/vvs/ufeuerse.pdf" title="Haltestellenplan"
language="de" mimeType="application/pdf" />
          <mapLink url="http://192.168.32.18:8888/vvo/FILELOAD?Filename=bw-ww0_496E244C3.pdf"
title="Umgebungsplan" language="de" mimeType="application/pdf" />
          <type>stop</type>
        </from>
        <passedLocations name="Stuttgart Feuersee" state="complete" departureTime="2009-01-
14T09:19:00">
          <key providerCode="1" key="8111000:51:5006221:3512258:756087:NBWT" />
          <location name="Feuersee" regionName="Stuttgart West" regionId="08111000:51">
            <coord x="9165740" y="48772610" />
          </location>
          <type>stop</type>
        </passedLocations>
        <passedLocations name="Stuttgart Schwabstraße" state="complete" departureTime="2009-01-
14T09:20:00" arrivalTime="2009-01-14T09:20:00">
          <key providerCode="1" key="8111000:51:5006052:3511588:756353:NBWT" />
          <location regionName="Stuttgart" regionId="08111000:51">
            <coord x="9156620" y="48770230" />
          </location>
          <type>stop</type>
        </passedLocations>
…
        <passedLocations name="Oberaichen" state="complete" departureTime="2009-01-14T09:34:00"
arrivalTime="2009-01-14T09:34:00">
          <key providerCode="1" key="8116078:3:5002105:3509453:763592:NBWT" />
          <location regionName="Leinfelden" regionId="08116078:3">
            <coord x="9127410" y="48705170" />
          </location>
          <type>stop</type>
        </passedLocations>
        <passedLocations name="Leinfelden" state="complete" arrivalTime="2009-01-14T09:36:00">
          <key providerCode="1" key="8116078:3:5000175:3510597:764569:NBWT" />
          <location name="Leinfelden" regionName="Leinfelden" regionId="08116078:3">
            <coord x="9142930" y="48696370" />
```

```
          </location>
          <type>stop</type>
        </passedLocations>
        <to name="Leinfelden" state="complete" arrivalTime="2009-01-14T09:36:00">
          <key providerCode="1" key="8116078:3:5000175:3510597:764569:NBWT" />
          <location name="Leinfelden" regionName="Leinfelden" regionId="08116078:3">
            <coord x="9142930" y="48696370" />
          </location>
          <mapLink url="http://www2.vvs.de/Download/Envmaps/vvs/uleinf.pdf" title="Haltestellenplan"
language="de" mimeType="application/pdf" />
          <mapLink url="http://192.168.32.18:8888/vvo/FILELOAD?Filename=bw-ww0_496E244C4.pdf"
title="Umgebungsplan" language="de" mimeType="application/pdf" />
          <type>stop</type>
        </to>
        <line operatorName="DB" product="S-Bahn" trainName="S3" trainString=""
direction="Flughafen/Messe" lastStop="Flughafen/Messe" code="11">
          <sideInfo techtype="" businesstype="EFAEX" addtechspec="" addbusinessspec="" content="2;S-
Bahn;;;S3;S3" />
        </line>
        <providerCodes>1</providerCodes>
      </elems>
      <elems durationMinutes="3">
        <from name="Leinfelden Bf" state="complete" departureTime="2009-01-14T09:41:00">
          <key providerCode="1" key="8116078:3:5000175:3510597:764569:NBWT" />
          <location name="Leinfelden" regionName="Leinfelden" regionId="08116078:3">
            <coord x="9142930" y="48696370" />
          </location>
          <mapLink url="http://www2.vvs.de/Download/Envmaps/vvs/uleinf.pdf" title="Haltestellenplan"
language="de" mimeType="application/pdf" />
          <type>stop</type>
        </from>
        <passedLocations name="Leinfelden Bf" state="complete" departureTime="2009-01-14T09:41:00">
          <key providerCode="1" key="8116078:3:5000175:3510515:764469:NBWT" />
          <location name="Leinfelden" regionName="Leinfelden" regionId="08116078:3">
            <coord x="9141820" y="48697270" />
          </location>
          <type>stop</type>
        </passedLocations>
        <passedLocations name="Leinfelden Schönbuchstr." state="complete" departureTime="2009-01-
14T09:43:00" arrivalTime="2009-01-14T09:43:00">
          <key providerCode="1" key="8116078:3:5007103:3510241:764881:NBWT" />
          <location regionName="Leinfelden" regionId="08116078:3">
            <coord x="9138080" y="48693570" />
          </location>
          <type>stop</type>
        </passedLocations>
        <passedLocations name="Leinfelden Riedweg" state="complete" departureTime="2009-01-14T09:44:00"
arrivalTime="2009-01-14T09:44:00">
          <key providerCode="1" key="8116078:3:5002099:3509789:764939:NBWT" />
          <location regionName="Leinfelden" regionId="08116078:3">
            <coord x="9131940" y="48693050" />
          </location>
          <type>stop</type>
        </passedLocations>
        <passedLocations name="Musberg Wanderweg" state="complete" arrivalTime="2009-01-14T09:44:00">
          <key providerCode="1" key="8116078:5:5007105:3509642:764994:NBWT" />
          <location name="Wanderweg" regionName="Musberg" regionId="08116078:5">
            <coord x="9129950" y="48692560" />
          </location>
          <type>stop</type>
        </passedLocations>
        <to name="Wanderweg" state="complete" arrivalTime="2009-01-14T09:44:00">
          <key providerCode="1" key="8116078:-1:5007105:3509790:764910:NBWT" />
          <location topLevelRegionName="" topLevelRegionId="" name="Wanderweg" regionName="Musberg"
regionId="8116078" />
          <mapLink url="http://192.168.32.18:8888/vvo/FILELOAD?Filename=bw-ww0_496E244C5.pdf"
title="Umgebungsplan" language="de" mimeType="application/pdf" />
          <type>stop</type>
        </to>
        <line operatorName="SSB" product="Bus" trainName="86" trainString="0" direction="Stuttgart
Vaihingen" lastStop="Stuttgart Vaihingen" code="20">
          <sideInfo techtype="" businesstype="EFAEX" addtechspec="" addbusinessspec=""
content="3;Bus;;0;86;86" />
        </line>
```

```
        <providerCodes>1</providerCodes>
      </elems>
    </schedules>
  </connectionResponse>
</MultiResponseType>
```

Figure 72: Example Multi() response

3.6 Miscellaneous Codes and Return Values

This chapter describes enumeration, return codes and restrictions of the API.

3.6.1 LocationTypeType

LocationTypeType	Description
null	Initial value (only allowed in entry of LocationRequestType)
stop	Station or Stop
address	Address
poi	Point of Interest (POI)
coord	Coordinate (typically as WGS84)

3.6.2 LocationType.state

LocationType.state	Description
initial	This status must be initialized when a search controller sets its first request (used only in request)
incomplete	The request is not completely answered, the search controller must run a further request after verification with the customer (used in request and response)
complete	the relevant stops or addresses are identified correctly and individually (used in request and response)

3.6.3 Operations Return Codes (*ReturnCodeType*):

The following list defines the values of return codes which are delivered and should be delivered by the functions of the interfaces.

ReturnCodeType	**Description**
OK	Success of the operation
ERROR_COMM	Communication failed (the problem of this return code may be that a more basic communication problem of the middleware will not be detected)
ERROR_SUPPORT	Function not supported
ERROR_NOTSERVE	Unable to answer this relation
ERROR_DATE	Date not during timetable period
ERROR_FORMAT	Format failure within request
ERROR_DEST	No transport at the destination station
ERROR_VIA	No transport at an intermediate stations
ERROR_START	No transport at the departure station
ERROR_OTHER	Internal failure
ERROR_LOCATION_UNKNOWN	No matching entry for the input
ERROR_LOCATION_TOO_MANY_HITS	Too many matching entries for the input (it should be avoided to deliver an empty list in this case, the returned list should cover maxlist entries)
ERROR_ORIG_EQUAL_DEST	The start location is identical to the destination location

3.6.4 Operation Info Codes (InfoCodeType)

InfoCodeType	**Description**
ok	No information available
less_than_requested	Less connections than requested
more_than_requested	More connections than requested

unknown_feature	A requested feature selection was ignored (this can be implemented by a server provider to deliver some connection instead of nothing) this code must be set if a server is acting that way
unknown_product	A requested product exclusion was ignored (this can be implemented by a server provider to deliver some connections instead of nothing) this code must be returned if a server is acting that way, even if a server can not differentiate between selected products
unknown_handicap_option	A handicap option was requested but is not supported
unknown_language	Language not supported (this is the case if a server can not provide the requested language)
origin_replaced	The origin location was replaced by a near location
destination_replaced	The destination location was replaced by a near location

4 Appendix

4.1 Implementation and Test Hints

This chapter summarizes experiences and definitions made during the development of DELFI. Definitions, like the minimum requirements, were made by DELFI providers starting at the functionality defined by their local existing system properties. This chapter can also be used in order to have a look on the different possibilities of implementations and problems of testing a distributed system.

4.1.1 Minimum Standards for the Interface

Starting from the actual functionality of existing local information systems, the steering committee defines a set of minimum requirements for participating systems. Every participating information system has to fulfil the following minimum requirements which are divided into two aspects:

- The active systems (active servers or search controllers, showing the combined information to the customer) have to grant that the provided information is complete and correct according to the received partial information.
- Existing information systems shall be able to keep the existing functional abilities and must not reduce it to a common minimum.

Item	**Output**	**Min. Req.**
start destination	repetition of input data	☑
spatial relation	city	☑
	suburb	⌫
	stop name	☑
	address / walk description	⌫
transport modes	type of vehicle	☑
	line identifier / transport company	☑
	train number	⌫
	direction	⌫
	destination stop name / displayed destination	⌫
	guaranteed transitions / transition durations	⌫
time information	departure time – arrival time	☑
	over all travel time	☑
	date	☑
	day	☑
	validity range of calendar	☑
transitions	stop name	☑
	arrival / departure time	☑
	walk / walking duration	☑
	number of transitions	☑

☑ = minimum requirement

⌫ = Incoming information (delivered by a passive server as partial information) has to be shown to the customer, but it is not a required necessary internal feature of a passive server

Some information items can be displayed optionally. The DELFI interface offers the possibility of submitting the desired presentation form and sequence for showing it to the customer interface.

A guideline for the information presentation on all information systems is that the content of information has to be independent from the platform.

None of the participating systems has to go beyond their actual features. Therefore, the following information content has been agreed to be submitted via the interface.

Item	**Input**	**Min. Req.**
Stop name	Complete name	☑
	String search	☑
	String similarity search	☑
	Shorted names search	↗
	Alternative stops (start/destination)	↗
Political separations	County/federal state	↗
	City	☑
	Suburb	⌫
	Address	☑
Local descriptions	Postal ID	↗
	License plate ID	↗
	Point of interest	↗
	Co-ordinates	☑
Time	Departure time	☑
	Arrival time	☑
	Time interval for dep./arr.	⌫
	Date	☑
Itinerary criteria	Product selection (include/exclude specific types of mode)	⌫
	Via stations	⌫
	Maximum number of partial trips	⌫
Search criteria	Fastest itinerary	☑
	Itinerary with lowest number of transitions	⌫
	Bike carrier possibility	↗
	Couchette/sleeper car	⌫

☑ = minimum requirement

⌫ = Incoming information (delivered by a passive server as partial information) have to be displayed to the customer, but it is not a required feature of a

passive server

↗ = intended for future steps

It is not necessary, that all user interfaces (web front ends) assists all data elements. Only the DELFI interface has to take care of all contents to be able to be passed. This requires that all products have been categorised in order to have comparable and selectable categories (see the meta tables).

4.1.2 Requirements for the Search Controller

The described framing architecture defines only the relation between a search controller (acting as a client) and passive servers (defined as servers). This allows an advanced distributed itinerary search incl. considerable advantages.

In the following, some variants of (future) designs for the realizations of a search controller are described. The comparison of extreme alternatives shows the bandwidth of possibilities which will not be restrained by the DELFI definition.

Today the search controller is linked directly with the existing local system. This implementation allows first an internal search within the local system for origin and destination and an internal implementation for the location resolving.

Figure 73 shows the typical solution of local implementations nowadays beside the DELFI servers at which the own web client has internally a direct access to the search controller- the search controller is normally only internal accessible. The local location resolver component is not shown in the figure.

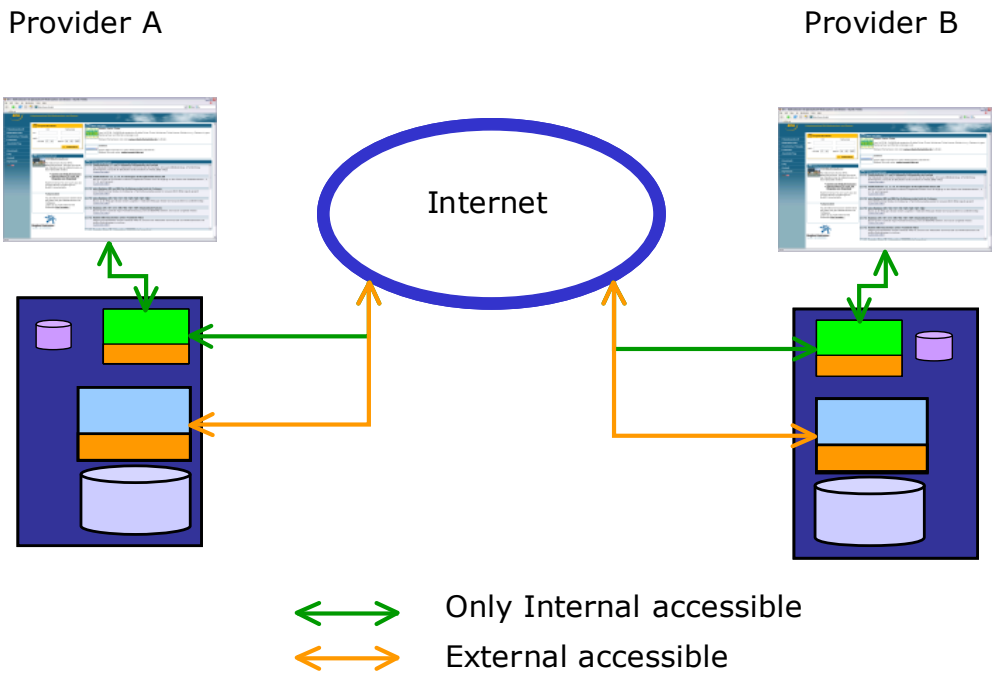

Figure 73: Typical actual implementation with search controller, DELFI server and web client

Another (future) possibility may be the provision of a search controller service for

other participants (Figure 74). In this schematic example, provider B has a DELFI server but no own realized search controller. He can use the search controller of provider A for distributed information. If all participants agree it is (theoretically) possible that providers without own server can access the service of DELFI partners (provider C).

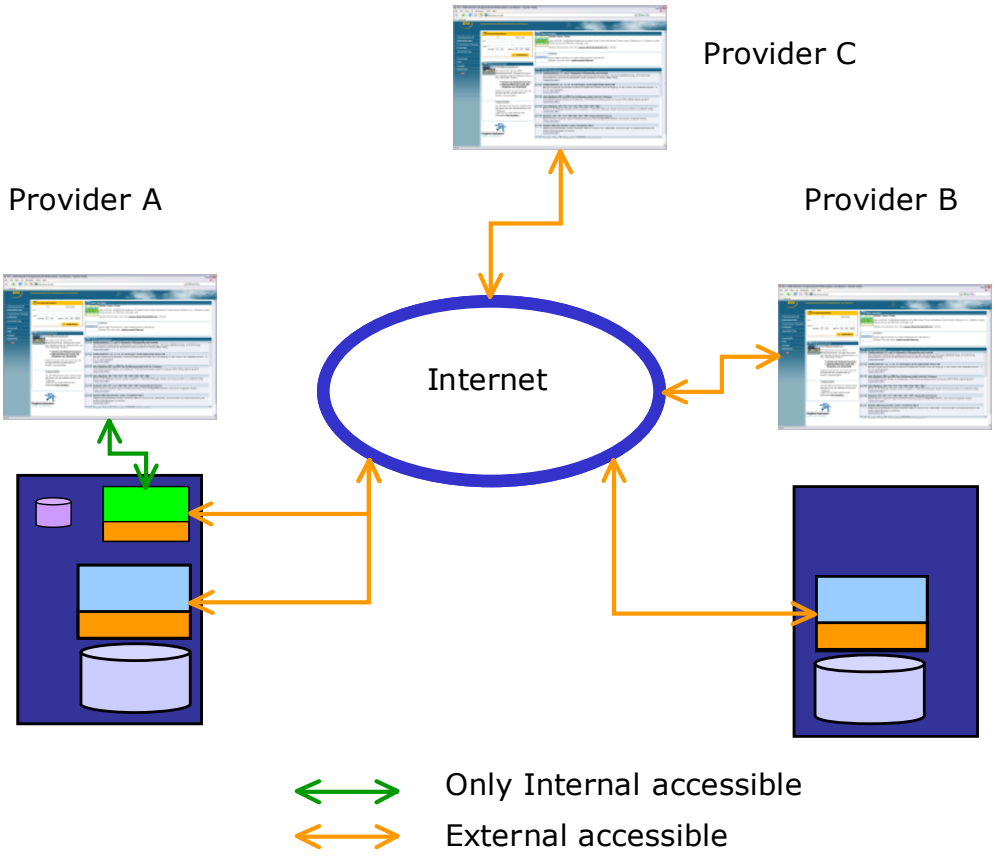

Figure 74: Implementation where search controller are open for other web clients

Another further possibility may be that the search controllers themselves are extended with the web service interface. This will allow to cascade systems (i.e. regional, national, international).

Other computing algorithms will allow to speed up the performance of the computation. This can happen if a three phase search must not be performed. This can happen if, first, the long-distance connection is first computed (in most cases this is the more critical part of the overall itinerary because the schedules of long-distance transports are more seldom than those of local transport) and then the local partial itineraries are computed as attachments. This can cause suboptimal solutions in those cases where the regions of origin and destination are connected, but it works well in separated regions.

4.1.3 Testing the Distributed System

A distributed system cannot be tested in the same way as integrated systems are usually tested. At a normal test, the data base and perform regress tests are used from time table data. At a distributed test, the calculated itineraries can only be compared with local knowledge of data administrators or with reference systems.

The following section repeats the used tests and testing methodologies during the setup of the DELFI system in a condensed form in order to show the problems of the test and possibilities for their solution. A method of evaluating the correctness of fastest itinerary compilations is also defined as a tool for a continuous test of the complete distributed system.

The evaluation of the distributed computed result is based on a comparison of the same computation within a pooled data base. This test method is able to show the quality and performance of the distributed system compared to a pooled system. The elements of evaluation are:

- quality of the distributed itinerary compared to the result of pooled systems
- time effort for the distributed compilation
- estimation of the communication data volume

Initial test of algorithm quality

During the set up of the DELFI system, the consortium and the steering committee decided to use a formal and satisfying test method which guarantees an objective and numerical sufficient test result. The extensive method has been chosen in order to qualify for the mathematical optimum of a fastest itinerary within a given time period.

The method used is based on the use and comparison of reference itineraries with the distributed computed itineraries. A first problem was the creation of reference itineraries. Figure 75 illustrates the process to develop reference itineraries used for the comparison. The regional time table data of two German federal states together with the long-distance time table of the DB AG are pooled for two different stand-alone information systems with slightly different computing algorithms. The problem of selecting was that the calculated results of itineraries may vary in the number of allowed transitions in order to catch the fastest path. The both used systems represent the most common information systems for public transport in Germany.

In order to have a numerous valid set of reference itineraries, both information systems performed an itinerary compilation for a thousand start-destination combinations of stops on the integrated time table data base. Both resulting itineraries sets of both computing systems were then compared with each other (both results per request) in the step of developing reference data.

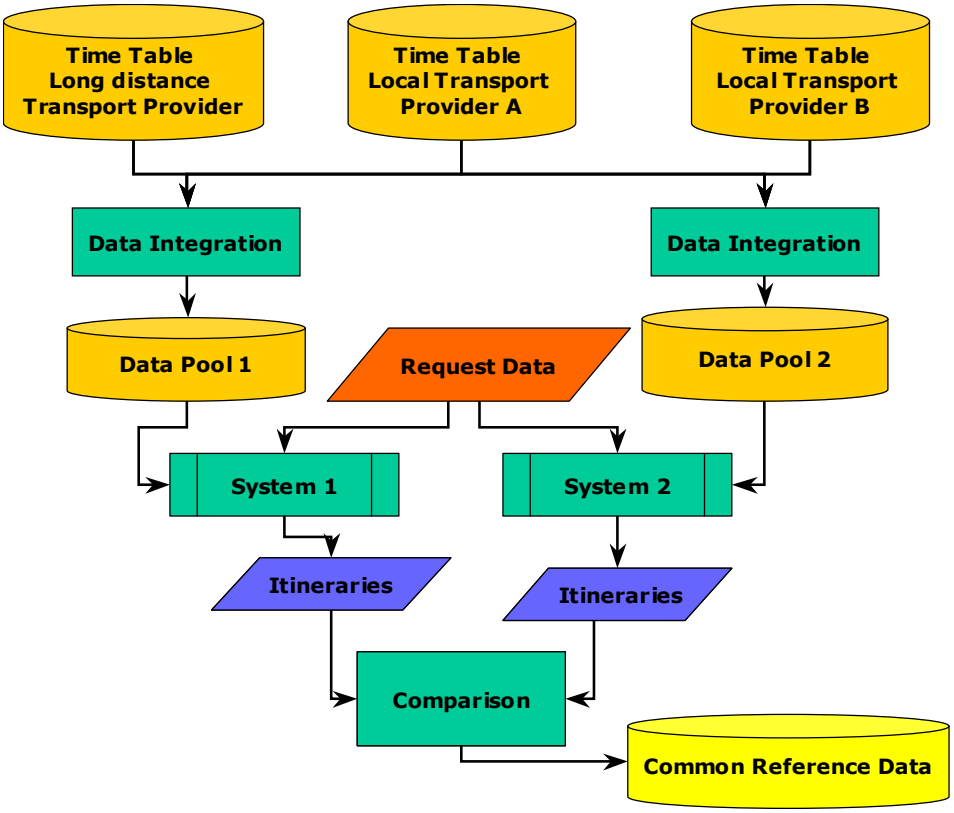

Figure 75: Flow diagram: developing of reference itineraries for evaluation

The following definition was fixed for this comparison process (see the example below):

Status	System	Departure time	Arrival time	Duration	No. of Transitions
Identical	System 1	22:33	6:54	8:21	5
	System 2	22:33	6:54	8:21	5
Same Duration	System 1	18:33	6:16	11:43	5
	System 2	18:33	6:16	11:43	6

Itinerary results which will not pass this comparison test were not used as reference data. The distributed system was polled by using these reference itineraries and the related request data and the resulting itineraries were compared with the reference data (see Figure 76).

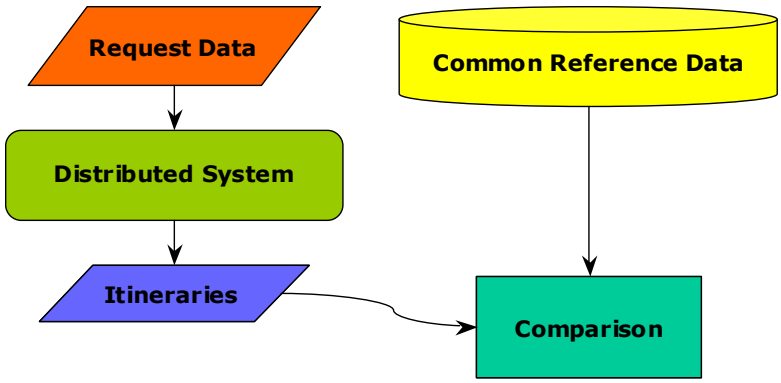

Figure 76: Flow diagram: Comparing distributed results with reference itineraries

A successful comparison process led to the result that it is approved. On the other hand, the steps towards this result detect some major mistakes possible specially at the transition points. This cognition improves the metadata process significantly.

This complex test method was used to grant the distributed system to be as qualitative high as the other existing systems.

Continuous Quality Insurance

A distributed system is much more vulnerable to a breakdown than classic stand-alone systems. On the other hand, if systems give partially overlapping data they can improve and behave as backup systems that improve the whole system. This is also true for the data quality within the whole system. While system providers deliver so called regression tools in order to test the quality of a set of new time table data, this method is not applicable in a distributed system.

Therefore, it was decided to set up a quality insurance system which behaves centrally and provides some central services which work partially in automatic mode and that provide a central information basis on the other side. Figure 77 and Figure 78 show schematic illustrations of some features of this quality management system. Automatic parts are online tests, test requests and specific requests with existing reference itineraries (see Figure 79).

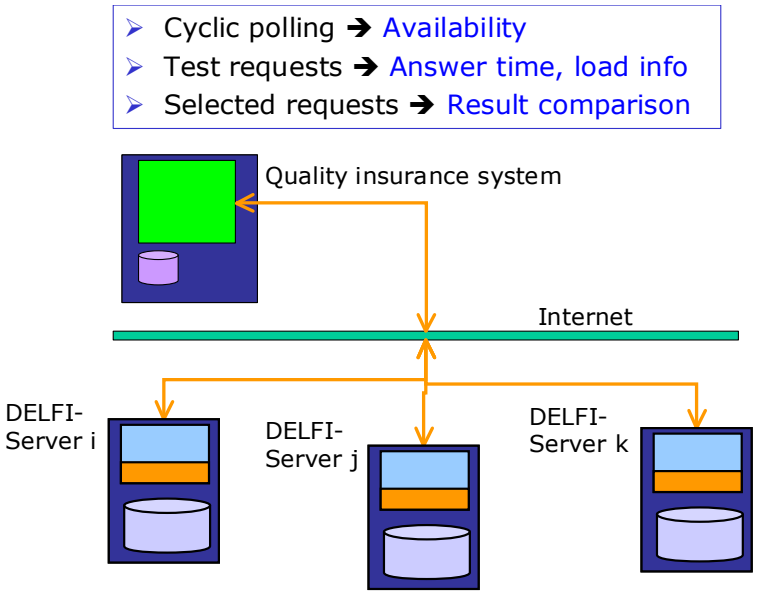

Figure 77: Quality management: central information and measurement system

Figure 78 illustrates the information part of the quality insurance system. Information basis for error logs, repair and success are also present as news or adjustment groups for data administrators i.e. for transition points.

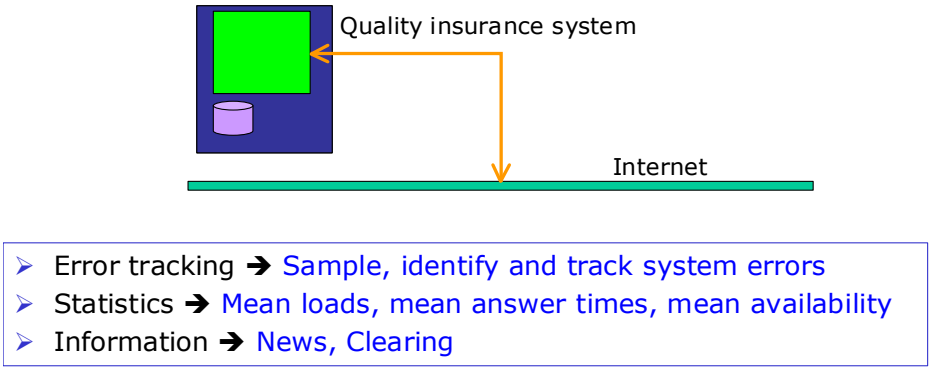

Figure 78: Quality management: task tracking and central information

Additionally, a method was defined for the task of itinerary quality proves which gives advice to data modifications (which can be mistakes or wanted ones). In Figure 79, the flow of this itinerary test method is illustrated. Such tests can be started at regular time intervals or when modifications in the system seem that it is necessary to test.

The test results are stored in a data base and can be used for warnings in case of discrepancies of new results. There were set up basic principles for testing: a test with randomly chosen origins and destinations and the repeat modus at which earlier requests from the data base will be used for new requests. The results from the last method will be compared with the stored results.

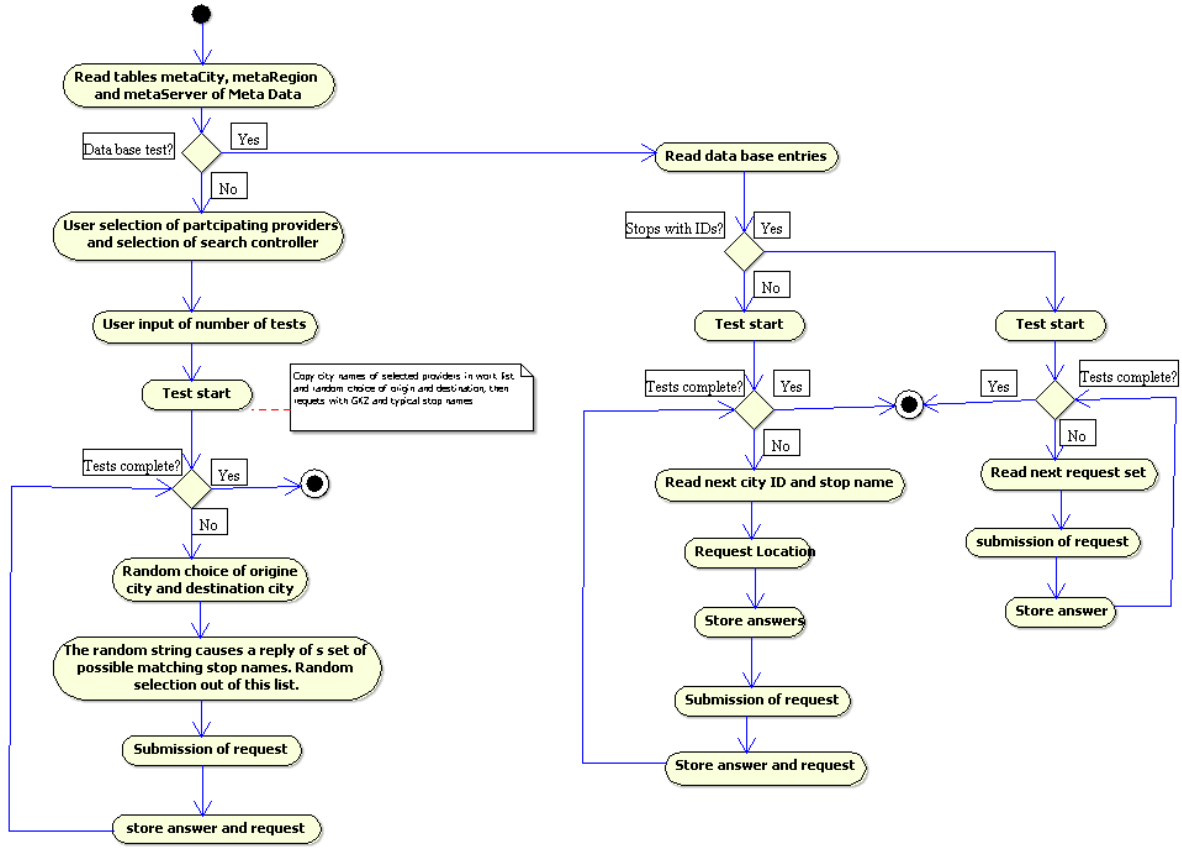

Figure 79: Overview of the flow of test methods in order to ensure continuous quality

Answer parameter are summary data of the itinerary (duration, number of transitions) also as communication parameter (answer time, error). Complete request and answer data are stored.

The quality insurance system is only accessible for the steering committee or data administrators of the federal states.

4.1.4 Test Examples of the DELFI System

During the set up of the DELFI system questions about the performance of the distributed system came up as well. After having done a series of tests, information about the development process in particular was collected but some extracts may also be of general interest. Those extracts are given in the following:

A method of estimation of system and communication requirements for a practical use of the system was introduced which was used before the development of DELFI was set up and which can be used in other cases for similar situations. Basically, the number of requests per time interval of existing systems or of similar situations is necessary for the evaluation of expected communication load. In Germany, the results of the usage of the Internet portal of the DB AG was additionally used (www.bahn.de).

For the estimation of required system layouts and capacities of communication connections the regular transaction load has to be determined und itemized on the

separate server. It has to be reminded on the fact that not only the own requests cause stress on the server but also the requests from active systems (search controllers) of other systems. At normal use the DELFI information is expected to be an additional service for regional information systems and will not replace the long-distance information of the DB AG in near future.

The DELFI partners additionally estimated to what extend the request volume of their systems will grow if DELFI will be installed. The volume – most time estimated on monthly base – then has to be splitted down to an estimated maximum stress in a specified short time interval (quarter of an hour). In most cases, this was nearly 10% of the daily volume. The other estimations as follows:

- One stop of a request (origin or destination) is within the 'own' system
- The other stop is a distributed using estimation data or – in the case of Germany – existing data of a long-distance transport information system
- The long-distance transport server will be part of a computing process in every case

For every participating passive server the load results from the sum of its direct requests (from the own Internet portal) and from all other requests done by the search controllers of all other systems.

All these assumptions can be used now for the estimation of computing performance and communication bandwidth.

The load is mainly determined by the operation *PartialConnections* because this operation is the most complex parts regarding the computing performance and amount of data.

The following mean data sizes have been determined during tests: For every answer of a passive server about 3.5 kB data were submitted. This results in a mean volume of about 10 kB per (successful) information between the search controller and the passive servers (a server for origin and destination and the long-distance transport server). Additionally, about 1 kB data between the search controller and the web interface will be submitted. The number of this data may vary highly depending on the real situation. During the test, the used systems (which are not fully extended production servers) achieved the following mean answer times per API function:

Operation	**Average Duration**
Transitions	0,1 sec
PartialConnections	0,4 sec

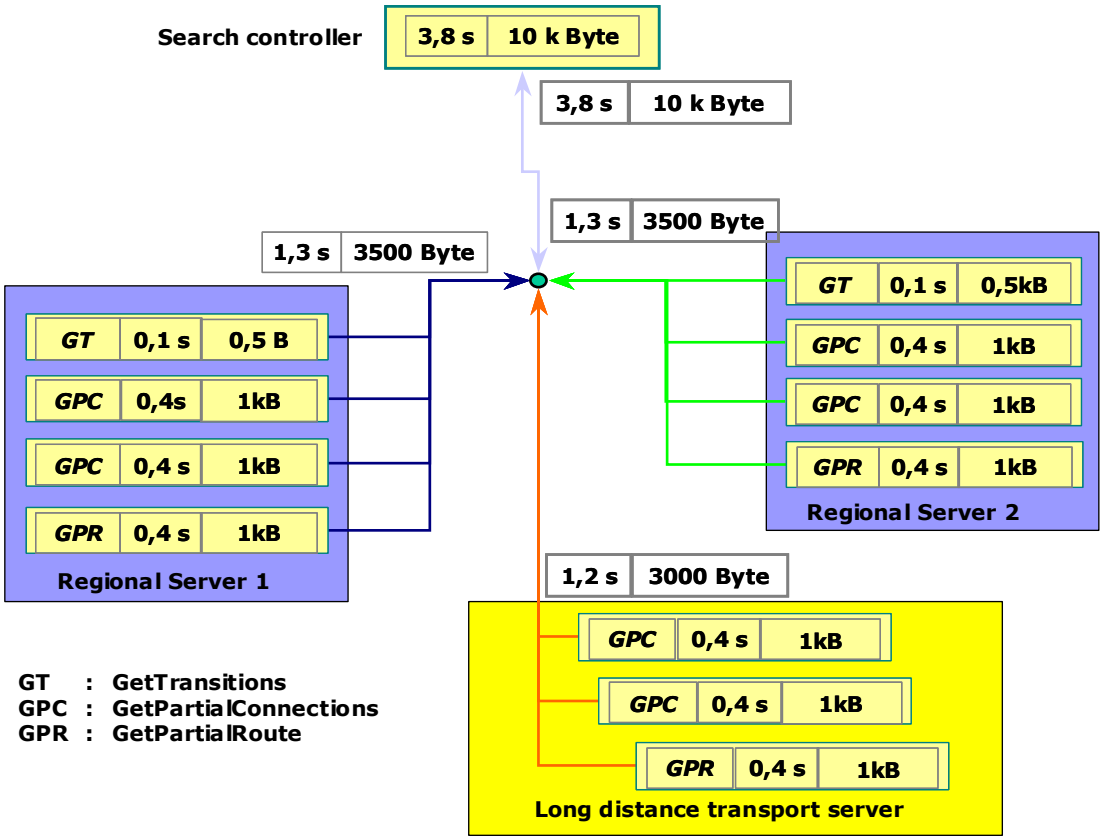

Figure 80: Dataflow, timing and volume during distributed computing process

Those data are also part of the quality information system described above. A further problem may be that bad matches of *Locations()* result in large lists with more traffic than necessary.

NOTE: The above image contains the naming of the CORBA interface that DELFI has used up to API version 4. GT (GetTransitions) should be read as Transitions(), GPC (GetPartialConnections) and GPR (GetPartialRoutes) should be read as PartialConnections().

4.2 Glossary and References

Passive (DELFI) server	A standard itinerary calculation system working in time table data accessible in an internal data base, extended with the DELFI communication interface for partial or complete itinerary requests
Active (DELFI) Server	A special component necessary for DELFI itinerary calculation; The active system or search controller acts between a web front end and passive DELFI systems. The active system performs the itinerary combination out of delivered partial itineraries.

Search controller	Another name for an active DELFI server.
Provider	In this document, used for acting information providers in the case of electronic time table information for public transport in Germany
Time table data	The basic data contains the planned trips of means of public transport systems; which mode when at what station.
Pooling	Integrated storing of time table in the data base direct accessible by the calculation algorithm
System	In general, means used here for all complete components of which an electronic time table information systems consists of
Partial information / itinerary	A part of the complete itinerary from the origin (see origin) to the destination (see destination). The combination of the partial itineraries represents the complete itinerary
Query	A request submitted by a customer or an active DELFI component to another DELFI component
Origin	The desired start point of the customer's trip. It can be an address, a stop/station or a co-ordinate or a specific point of interest.
Destination	The desired end point of the customer's trip; it can be an address, a stop/station or a co-ordinate or a specific point of interest.
Start node	In general terms, a start node for a calculation or search can be the origin, a transition point, or only for systematic explanations.
Target node	In general terms, an end node for a calculation or search can be the destination, a transition point or only for systematic explanations.
Connection	The term connection is used in general (especially in the theory chapter, the appendix) for a sequence of rides between stations with or without a physical transition of the person between trains. The expression of a connection is often used (chapter 2 and chapter 3) to define the arrival at a location in time but not the route to arrive at this point.
Route	A route is nearly the same as an itinerary. A route describes a series of rides in reality or edges in graph theory. Differences may come up in terms of completeness of the route if the request does not contain the first part of the trip from the activity location to the station or stop.

Itinerary	An itinerary is a complete journey of a person (or persons) from an origin to a destination. If a route contains even the non-motorized parts the route is the same as the itinerary but may contain no additional information hints as the itinerary should have.

4.3 Distributed Connection Calculation – Algorithmic Possibilities and Limits

In 1997 Professor Möhring has published a paper concerning the problem of the general solvability of distributed computation of optimal itineraries. The paper was published in German. Concerning the importance of this paper related to the distributed computation a very rough summary with most relevant advices is given here. So the following is a translation of parts of the original text. The complete text can be found in [19].

4.3.1 Introduction and Graph Theory

For the following text a basic knowledge of searching connections between two points and a given supply of public transport is assumed. The possibilities of modelling a connection supply with graphs and the modelling of the problem as shortest path search even with more than one criterium should be available. A starting point is time table data provided by one or more transport operators. The set of connections leads to a directed line graph (see also [5]). For the basic connection request time table data of the form

(location, location, departure time, arrival time, kind of transport mode,...)

is necessary. Additionally, further information as a number of transitions (changes from one transport element – i.e. from tram to train - to another, usually by walking), price information etc. can be provided. Such a data set shall be named as *elementary connection*. All participating *partial systems have to provide these data as a minimum*. The sequence of elementary connections and transitions figured in a time-space-diagram are shown in Figure 81.

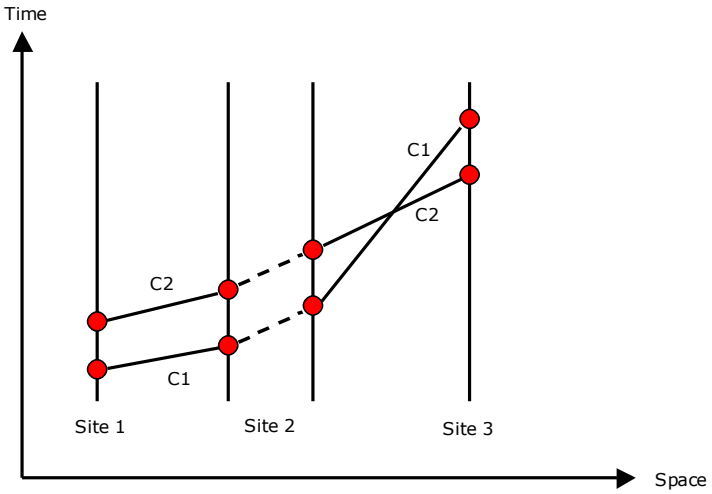

Figure 81: Time-Space-Diagram of Trip Connections

Solid lines represent trips, dotted lines transitions. This can be modelled directly into a graph named *space-time graph*. *Nodes* of this graph are locations with given departure and arrival times. Every location is represented as a node, *edges* are on the one hand elementary connections and transitions on the other. The related edge connects the arrival node with the relating departure node (see Figure 82).

For a specific connection request additionally, a start node *Origin* and a target node *Destination* will be introduced and connected with edges to all nodes corresponding to the origin. Within the so defined graph G all directed paths from origin to destination represent all possible connections. Therefore, connection requests can be done by the search of a best path in *G*.

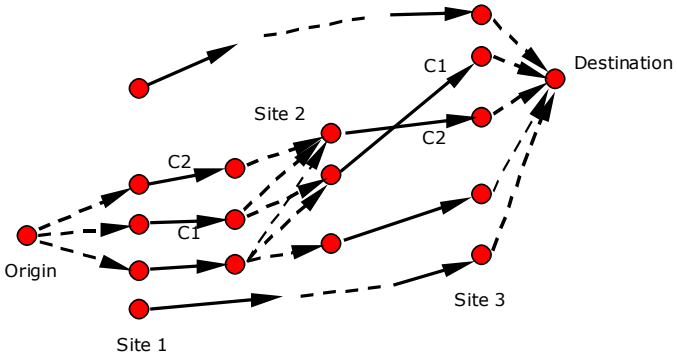

Figure 82: Section of the time-space-graph for the time-space-diagram Figure 81

The time interval in which departure times will be considered is not defined. The simple case is that all connections are synchronized in intervals. The search can be concentrated on an interval which represents the greatest interval. Otherwise, the time interval can only be chosen heuristically.

The difference between paths lies in their different weight criteria. Usual weight criteria are

- fastest connection
- shortest connection (i.e. kilometre)
- frequency of transitions
- balanced time during transitions
- kind of mode (surcharges, equipment)
- validity of special rates (weekend, trips within one rate area)
- cheapest connection
- minimum travel times (i.e. night trains)
- touristic attractivities
- multi criteria combination (i.e. fastest connection with minimum number of transitions)

Most of above single criteria can be used in addition with the space-time graph and the path can be computed with the Dijkstra algorithm.

multi criteria target functions result from combinations of elementary criteria. In this case, all *pareto-optimal connections* for these criteria have to be computed which are those combinations of travel time and number of transitions which cannot be undercut by any other combination. This problem can be solved also by the Dijkstra algorithm, but the complexity rises proportionally to the number of possible pareto-optima.

In every node v instead of d(v), all two-dimensional weights of pareto-optimal partial paths have to be stored. This number can theoretically be very large, in reality the number of transitions will be kept small (maximum 3 to 5) which leads to a storage of only a few weights per node.

With time spares of transitions, parts of the graph can be masked even by a selection of specific transport modes.

The definition of pareto-optimal paths in graphs can be summarized with a basic model. For every edge *e* of the graph $\lambda(e) = (\lambda_1(e),....,\lambda_r(e))$ is a r-dimensional weight vector (e.g. travel time, number of transitions, price,...). For simplicity we assume that the weight of a path *W* related to the *k*th component (*k = 1....r*) is additive, which is $\lambda_k = \Sigma_{e \in W} \lambda_k(e)$ (see [17]).

A path W from s to t with a weight $\lambda_k = (\lambda_1,\ldots,\lambda_r)$ is pareto-optimal or efficient, if no other path W' exists from s to t with a weight $\lambda' = (\lambda'_1,\ldots\lambda'_r)$, with $\lambda'_k \leq \lambda_k$ for k=1...r and $\lambda'_l < \lambda_l$ for any $l \in \{1,\ldots,k\}$.

4.3.2 The Dijkstra Algorithm for Pareto-optimal Paths

The algorithm of Dijkstra is the classic (and best) algorithm for the calculation of shortest paths in direct graphs with non-negative edge weights if the weight of a path is additive of the weights of its edges.

Instead of the stored distance d(v) in node v in case of multi criteria r-dimensional vectors will be used $d[v] = (d_1[v],\ldots,d_r[v]$, where $d_k[v]$represents the distance from s to v related to the k-th edge weight $\lambda_k(e)$. For a node a set of such vectors will be

stored in general. They represent the values of so far calculated pareto-optimal s, v -paths. At start of the algorithm is

$$d(v) = \begin{cases} (0,0,\ldots,0) & falls\, v = s \\ \lambda(s,v) & falls\, die\, Kante\,(s,v)\, existiert \\ (\infty,\infty,\ldots,\infty) & sonst \end{cases}$$

The choice of the unmarked node with lowest d[v] is substituted by the lexicographical *smalles unmarked vector* d[v], which is then marked. Starting from related node v for all edges e of v to w the stored pareto-optima has to be actualised by adding the vector d[v] + λ(e) to the stored vectors at w and eliminating all non pareto-optimal vectors. The correctness of this extended Dijkstra algorithm is shown in [17]. The complexity dominates the whole runtime and can be calculated by

$$\sum_{v \in V} \text{pareto}(v) \cdot \text{outdegree}(v)$$

Where pareto(v) is the number of all in v stored pareto-optima and outdegree(v) is the number of outgoing edges. The algorithm is indeed per pareto-optima as fast as the normal Dijkstra algorithm, but the number of pareto-optima can be exponentially large. Indeed the calculation of all pareto-optimal s, t-paths in general is weak NP complete. (see [18]). In Figure 83 is an example for an exponential number of pareto-optimal paths.

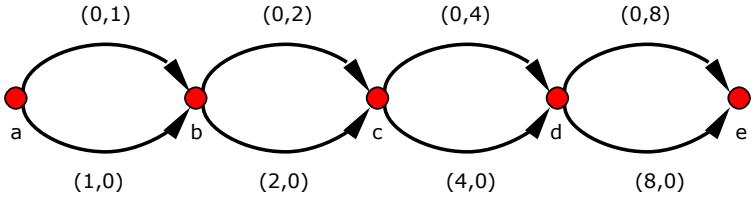

Figure 83: Example calculation pareto-optimal paths

The Dijkstra-Algorithm will provide the following pareto-optima (in the nodes)

a	b	c	d	e
(0,0)	(0,1)	(0,3)	(0,7)	(0,15)
	(1,0)	(2,1)	(4,3)	(8,7)
		(1,2)	(1,6)	(1,14)
		(3,0)	(5,2)	(9,6)
			(2,5)	(2,13)
			(6,1)	(10,5)
			(3,4)	(3,12)
			(7,0)	(11,4)
				(4,11)
				(12,3)
				(5,10)
				(13,2)
				(6,9)

				(14,1)
				(7,8)
				(15,0)

For n nodes in the last node 2^{n-1} pareto-optimal solutions will be generated. A simple upper limit for the maximum number of pareto-optimal solutions in a node v can be derived from the maximum path length $L_k(v)$ from s to v for the k-th criteria, if $\lambda_k(e)$ are integers and non negative. Then is

$$pareto(v) \leq (L_k(v))^r$$

Therefore, the number pareto(v) stays *polynomial* in number of number count n, if edge weights $\lambda_k(e)$ are polynomial in n and r is fixed.

4.3.3 Approximation of Pareto-optimal Paths

Abandoning the exact determination of all pareto-optimal paths, several methods for the estimation of approximations are known. In literature, three different approaches can be differentiated: In [8] the approach of weighted sums of the criteria is considered. In general, only bad approximations can be reached. With many criteria the solutions can differ by the factor r. This is too large for practical applications.

In [17, S. 171 ff] the approach is considered if simple scaling of lengths. It scales *the* (assumed to be integers) edge weights $\lambda(e) = (\lambda_1(e),\ldots,\lambda_r(e))$ on the less important criteria k=2,...,r about the factor $\alpha > 1$ and leads to

$$\lambda'(e) = \left(\lambda_1(e), \left\lfloor \frac{\lambda_2(e)}{\alpha} \right\rfloor, \ldots, \left\lfloor \frac{\lambda_r(e)}{\alpha} \right\rfloor \right)$$

As they are reduced by the factor α, the number of pareto-optimal solutions is per node v smaller, because the path length $L_k(v)$ is shorter. A detailed analysis is given in [17, S. 172 ff and S. 203 ff].

Warburton [18] developed a polynomial approximation scheme for the calculation of pareto-optimal paths. The basic idea of his algorithm is multiple scaling and for every fixed scaling the resolving of a validity problem. This leads to the problem that every calculation of validity is as complex as a single Dijkstra calculation. Therefore, the approach of Warburton is theoretically relevant but can be assumed to be unacceptable in reality.

Lexicographical orders can much simplify multi criteria problems. Exemplified: A relatively small number of transitions is desired and the second criteria is the travel time differentiating between connections with the same number of transitions. In this case, the Dijkstra algorithm can be used directly and the addition is replaced by the componential addition of both criteria and the minimum is replaced by the lexicographical minimum.

4.3.4 Distributed Search

The general case within the distributed connection search will be that partial net are only connected at a few connection points. For this reason, we use the line graph

which shall consist of two partial networks. The locations where a transition is possible establish a so called *separator* which is the set of nodes. If this set of nodes does not exist, the graphs are no longer connected together. Is the number of transition points smaller compared to the number of all nodes we talk about *weak connected systems or networks*. Figure 84 illustrates this situation.

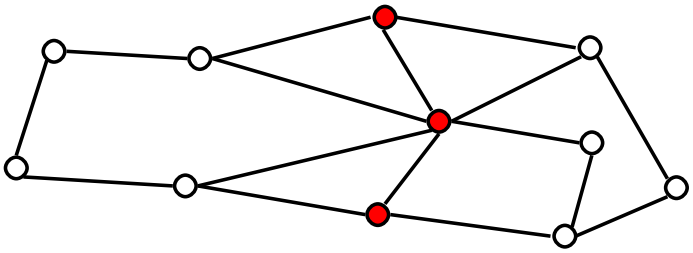

Figure 84: Unidirectional graph with separators (dark nodes)

In Figure 85, the partial networks to the separators of Figure 84 are illustrated.

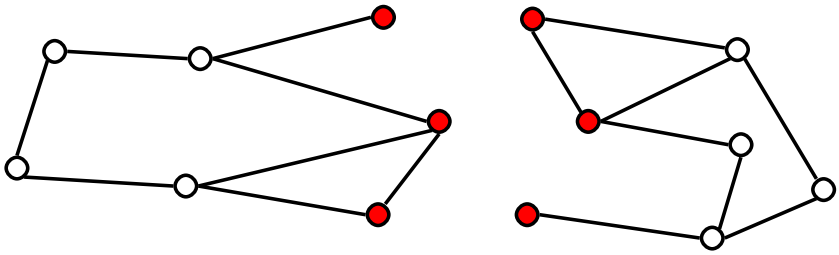

Figure 85: Partial networks of separators of the previous net

4.3.5 Weak Connected Systems

Most up-to-date might be the case of three weak connected networks. It represents the most common standard request starting at an origin in net 1 (i.e. local public transport in a city) to a bigger transport node (i.e. airport, main station), starting there into an over regional network (train, flights) to another bigger transport node in the region of the destination and then travelling within another local system to the desired destination (see Figure 86).

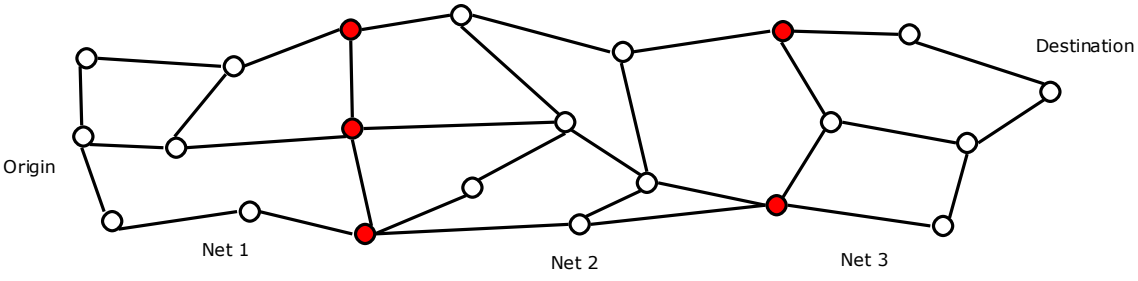

Figure 86: Network for a distributed standard search

Such a connection request shall be done by a superior algorithm (further called *master*). At the beginning, the master has only knowledge about origin and destination. The master has to locate the partial networks in which origin and destination are combined via the intermediate network (or a set of intermediate networks). This study does not focus on the determination of those networks but on the access to geographical information (covering areas or similars) as well as meta knowledge about connections between networks (i.e. train and flight, train and ferry) or about sequences of transport modes between destinations (local public transport – train – ferry – train – local transport) is imaginable.

If starting network, target network, and intermediate network(s) are known by the master the next information needed is the separators between the different networks. Even they are provided by meta knowledge. In order to process a shortest-path-search, the master is able to perform requests for shortest connections to the single partial networks but at this stage he knows only target nodes and transition nodes. If the master only receives answers of single connection requests but no information about time tables, line graphs, space-time graphs or other internal information of the partial networks.

The master has to process a shortest-path search on a graph which only consists of these nodes and the transitions. For the three networks of Figure 86 the graph is shown in Figure 87.

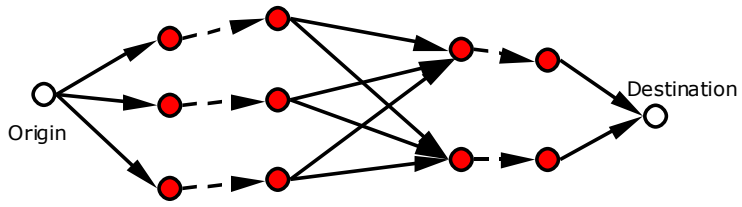

Figure 87: Graph for the search of master algorithm for the previous net with one transition for every separator (dotted lines)

Beginning at the origin node, there are, at first, transition edges of relevant departures in time (delivered by partial system for network 1). From those, there exist edges to every possible transition node between network 1 and 2. Within those transition nodes temporal transitions are displayed as dotted edges from where edges to the transition node between network 2 and 3 exist, there, connected again with temporal transitions to edges which lead, at last, to the destination. This kind of graph will be named the *master-graph*.

The master provides a shortest-path search on this graph for which he needs solid lines information about connections of the single partial systems.

The master must ask the connection for every combination of visible nodes because every of these edges can be part of a shortest path. The method of shortest path algorithms during the decomposition of graphs by separators has been analysed in detail at [4]. Further information for the calculation of shortest paths by decomposition with separators especially for road graphs has been documented in detail at [14].

To conclude, it can be stated that in weak connected systems all requests can be performed which can be done in a global system. The necessary number of requests is proportionally to the product of the biggest number of departure times with the number of the connection nodes between the single networks.

4.3.6 Pareto-optimal Paths in Distributed Searches

For a multi criteria search in the master-graph it is necessary for the determination of *all* pareto-optimal s, t-paths that the partial systems provide *all* pareto-optimal solutions of connections of the partial networks.

Now, one edge of the master-graph represents a set of connections, namely one for every pareto-optima. At the model of the master-graph, it can be considered by introducing as many parallel edges relating to the number of pareto-optima as possible. Every one of the edges represents a pareto-optimal path. By modelling the master-graph, it is again a graph with multi criteria edge weights, so that all methods are applicable. Especially the extended Dijkstra algorithm for the calculation of all pareto-optima in the master-graph is available. We illustrate this for the graph of Figure 84 which will be divided into 2 partial networks. The calculation in both partial networks results in the pareto-optima (0,3), (1,2), (2,1), (3,0) in node c and (0,12), (4,8), (8,4), (12,0) in node e. The master-graph results from there, in **Fehler! Verweisquelle konnte nicht gefunden werden.**. The example shows also that *no* pareto-optima can be abandoned, if *all* pareto-optima of a whole network have to be calculated.

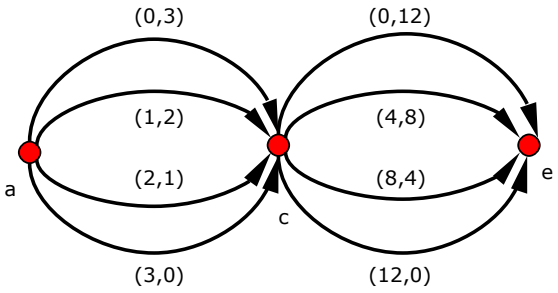

Figure 88: Master graph of the example of Figure 83 with c as separator

The question of approximation pareto-optimal paths in the distributed connection search is covered in [19]. All in all, it can be noticed that approximation techniques are compatible with the distributed search and the possibility of control is available if existing.

The complete algorithm can be more efficient if specific knowledge about the used algorithm in the partial networks is available. If it is known that in partial network 1 the calculation is done with the Dijkstra algorithm, one Dijkstra calculation is enough for every temporal alternative at the origin, which calculates (in fact of the *single source multiple target* condition of the Dijkstra algorithm) the shortest travel times to all transition nodes between network 1 and 2. That information could be provided for the master that causes him much less Dijkstra calculations.

This process including all partial networks results in a maximum of r temporal alternatives at a location

r	Dijkstra calculations at destination
$r \cdot a$	Dijkstra calculations at the transition nodes between network 1 to 2
$r \cdot b$	Dijkstra calculations at the transition nodes between network 2 to 3,

in summary

$r \cdot (a + b)$	Dijkstra calculations.

Compared to the base model we have

$r \cdot (a \cdot b)$	Dijkstra calculations

less.

These essential speed-ups predict that all partial systems deliver a submitted pair of origin and time necessary information (travel time, number of transitions, ...) for every transition point to the next partial network. We will name this as single source multiple target information.

While shortest-path algorithm, if they are not specialised for a search of a specific target node, have that ability this kind of information submitting should be considered very seriously.

Generally can be stated: all other criteria which can be formulated as shortest path problem can be calculated by using this model.

4.3.7 Strong Connected Systems

Strong connected systems are characterised by very many transition points between different partial networks in the line graph. They operate on a common node set and only differ in the edge sets which are the lines of the networks. In this case, no small separators exist which divide the partial systems. A typical example is the tram network and the subway of a city.

In this case, mechanism for a distributed calculation as described would fail. The only chance for the master is to create a complete overview over all partial systems via a series of requests.

Exists a *hierarchy of transport modes* in a strong connected network (i.e. bus, train, airplane) and are rides *unimodal* in the sense of consisting of a series of used transport modes which start with short-distance modes stepping continuously to longer distance modes and then again towards short-distance modes, than it is possible to perform a kind of distributed connection search with attributed edges because in this kind of network the transition points consist of a few nodes only. In this case, the described methods are applicable.

4.3.8 Summary

The original study in [19] proves that the usual search criteria can be modelled as

shortest path problems. Furthermore, it is shown that shortest-path calculations in general can be performed in distributed systems but weak connected systems should be preferred related to the complexity.

In summary, it can be fixed: a distributed connection search is possible and reasonable. The criteria to interfaces result from the request criteria and attributes side by side and can be delivered exactly in that case if all participating partial systems can perform this criteria and attributes.

5 Bibliography

5.1 Bibliography- the Algorithm Theory

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows. Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] B. V. Cherkassy, A. V. Goldberg, and T. Radzik. Shortest path algorithms: Theory and experimental evaluation. *Math. Programming*, 73:129–174, 1996.

[3] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of . *J. Assoc. Comp. Mach.*, 32(3):505–536, 1985.

[4] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs with applications. *SIAM J. Comput.*, 16:1004–1022, 1987.

[5] M. Gondran and M. Minoux. *Graphs and Algorithms*. John Wiley & Sons, New York, 1984.

[6] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Systems, Science and Cybernetics, SCC-4(2):100–107, 1968.

[7] U. Huckenbeck and D. Ruland. A generalized best-first search method in graphs. In R. H. Möhring, editor, *Proceedings 16th International Workshop on Graph*R. H. Möhring Verteilte Verbindungssuche 41 *Theoretic Concepts in Computer Science WG'90*, pages 41–60. Springer-Verlag, Lecture Notes in Computer Science, vol. 484, 1990.

[8] J. M. Jaffe. Algorithms for finding paths with multiple constraints. *Networks*, 14:95–116, 1984.

[9] E. L. Lawler, M. G. Luby, and B. Parker. Finding shortest paths in very large networks. In M. Nagl and J. Perl, editors, *Proceedings 9th InternationalWorkshop on Graph-Theoretic Concepts in Computer Science WG'83*, pages 184–199. Trauner Verlag, Linz, 1983.

[10] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley-Teubner, 1990.

[11] T. Lengauer and E. Wanke. Efficient solutions of connectivity problems on hierarchically defined graphs. *SIAM J. Comput.*, 17:1063–1080, 1988.

[12] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers, 1980.

[13] I. Pohl. Bi-directional search. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence*, pages 127–140. Edinburgh University Press, Edinburgh, 1971.

[14] B. Riedhofer. Hierarchische Straßengraphen. Master's thesis, Universität Stuttgart, Fakultät Informatik, 1997.

[15] G. Rote. Path problems in graphs. In G. Tinhofer, E. Mayr, H. Noltemeier, and M. Sysło, editors, *Computational Graph Theory*, Computing Supplementum 7,

pages 155–190. Springer-Verlag Wien, 1990.

[16] R. Sedgewick and J. S. Vitter. Shortest paths in Euclidean graphs. *Algorithmica*, 1:31–48, 1986.

[17] D. Theune. *Robuste und effiziente Methoden zur Lösung von Wegproblemen*. Teubner Verlag, Stuttgart, 1995.

[18] A. Warburton. Approximation of Pareto optima in multiple-objective, shortestpath problems. *Oper. Res.*, 35(1):70–79, 1987.

[19] Möhring, R.H. Verteilte Verbindungssuche – Algorithmische Möglichkeiten und Grenzen, Abschlussbericht DELFI-II, Bundesministerium für Verkehr, Bau- und Wohnungswesen, Berlin, FE-Nr. 70.532/1997

[20] T. Kämpke and M. Schaal: Distributed Generation of Fastest Paths, Proceedings of the 10th IASTED International Conference „Parallel and Distributed Computing and Systems", October 28-31, 1998, Las Vegas, Nevada, USA, p-172-177

5.2 Preferred Literature of the DELFI Project

[21] DELFI – deutschlandweite elektronische Fahrplaninformation, Konzeptstudie Teil 1, Wiesbaden, Ulm, 1997, Im Auftrag des Bundesministerium für Verkehr

[22] DELFI – deutschlandweite elektronische Fahrplaninformation, Konzeptstudie Teil 2 (Verteilte Verbindungssuche über eine Systemschnittstelle), Wiesbaden, Ulm, 1997, Im Auftrag des Bundesministerium für Verkehr

[23] DELFI-II, Abschlussbericht zum FE-Nr. 70.532/1997 im Auftrag des Bundesministerium für Verkehr, Bau- und Wohnungswesen, Berlin, Ulm, 1998

[24] DELFI-III, Abschlussbericht zum FE-Nr. 70.604/1999 im Auftrag des Bundesministerium für Verkehr, Bau- und Wohnungswesen, Berlin, Karlsruhe/Ulm, 2002

[25] Auswertung des produktiven DELFI-Testbetriebs, Abschlussbericht zum DELFI-Testbetrieb vom 1. Februar bis zum 30. April 2002, Frankfurt, 2002

[26] DELFI – Funktionale Erweiterung der DELFI-Schnittstellen gemäß Anforderungsprofil zur Fußball-WM 2006. Abschlussbericht zum FE-Nr. 70.743/2004 im Auftrag des Bundesministerium für Verkehr, Bau- und Wohnungswesen, Berlin, Karlsruhe/Ulm,

[27] DELFI - Funktionale Erweiterung der DELFI-Schnittstellen gemäß Anforderungsprofil zur Fußball-WM 2006. Abschlussbericht zum FE-Nr. 70.744/2004 im Auftrag des Bundesministerium für Verkehr, Bau- und Wohnungswesen, Berlin, Karlsruhe/Ulm,

5.3 Online Documentation

[28] see www.delfi.de

6 XSD-Specification

The following pages are containing the content of the actual XSD File.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- edited with XMLSPY v5 rel. 3 U (http://www.xmlspy.com) by Bastian Guddat (HaCon) -->
<!-- edited with XMLSpy v2007 sp2 (http://www.altova.com) by se (Mentz Datenverarbeitung GmbH) -->
<!-- edited by Stefan Engelhardt (Mentz Datenverarbeitung Gmbh (www.mentzdv.de)) -->
<xs:schema xmlns="delfi5eng" xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="delfi5eng" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="0.1">
  <xs:element name="AllTransitionsRequest" type="AllTransitionsRequestType">
    <xs:annotation>
      <xs:documentation>request to determine all transitions of a passive server</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="AllTransitionsRequestType">
    <xs:annotation>
      <xs:documentation>type of AllTransitionRequest</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="BaseRequestType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="AllTransitionsResponse" type="AllTransitionsResponseType">
    <xs:annotation>
      <xs:documentation>response of request all transitions</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="AllTransitionsResponseType">
    <xs:annotation>
      <xs:documentation>type of AllTransitionResponse</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="BaseResponseType">
        <xs:sequence>
          <xs:element name="transitions" type="TransitionType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="AttributeType">
    <xs:annotation>
      <xs:documentation>generic attribute type</xs:documentation>
    </xs:annotation>
```

```
    <xs:attribute name="attrMandatory" type="xs:boolean" use="required">
      <xs:annotation>
        <xs:documentation>Whether this text has to be displayed mandatory or not</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="sortOrder" type="xs:short" use="optional" default="-1">
      <xs:annotation>
        <xs:documentation>If sortOrder is less than 0, than all texts can be displayed in any order, in other cases the sortOrder contains positive values for
the order which have to be displayed from low to high</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="textCode" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>if filled, textcode contains a language independent text code, defined in the meta data tables</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="plainString" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>the desired text string can be stored in a direct way </xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="language" type="LanguageType" use="optional">
      <xs:annotation>
        <xs:documentation>the language of the information</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="BaseRequestType">
    <xs:annotation>
      <xs:documentation>base class of all requests</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="trans" type="TransactionType"/>
    </xs:sequence>
    <xs:attribute name="preferredLanguage" type="xs:language" use="optional">
      <xs:annotation>
        <xs:documentation>preferred language of text </xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="preferredSrs" type="xs:string" use="optional" default="EPSG:4326">
      <xs:annotation>
        <xs:documentation>preferred srs of coordinates</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="fallbackLanguage" type="xs:language" use="optional">
      <xs:annotation>
        <xs:documentation>fallback language if preferred language is not supported</xs:documentation>
      </xs:annotation>
```

```
    </xs:attribute>
    <xs:attribute name="apiVersion" type="VersionInfoType" use="optional">
      <xs:annotation>
        <xs:documentation>describes the api version of the calling system</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="BaseResponseType">
    <xs:annotation>
      <xs:documentation>base class of all responses</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="infoCode" type="InfoCodeType" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>ok (can be omiitted), less_than_requested (less trips returned than requested), more_than_requested (more trips returned than
requested), unknown_feature (a requested feature is not supported), unknown_product (a requested product is not supported), unknown_language (a requested
language is not supported), unknown_handicap_option (a requested handicap option is not supporterd), origin_replaced (the origin location has been replaced by
a near location), destination_replaced (the destination location has been replaced by a near location)</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="infoMessage" type="xs:string" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>additional information messages of the passive server</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="trans" type="TransactionType">
        <xs:annotation>
          <xs:documentation>the transaction of the corresponding request</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="calcTime" type="CalcTimeType" use="required">
      <xs:annotation>
        <xs:documentation>The time the calculation needed- must be the difference between the time stamp when the passive servers receive the request and the
time stamp whenever the answer is sent; time shown in seconds (with fraction)</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="returnCode" type="ReturnCodeType" use="required"/>
    <xs:attribute name="returnOtherText" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>if returnCode is ERROR_OTHER, this field should contain a description of the abnormal state</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="apiVersion" type="VersionInfoType" use="optional">
      <xs:annotation>
        <xs:documentation>describes the api version of the responding system</xs:documentation>
      </xs:annotation>
    </xs:attribute>
```

```
</xs:complexType>
<xs:simpleType name="CalcTimeType">
  <xs:annotation>
    <xs:documentation>calculation time of request in seconds</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:float"/>
</xs:simpleType>
<xs:element name="CapabilitiesRequest" type="CapabilitiesRequestType">
  <xs:annotation>
    <xs:documentation>request to determine capabilities of passive server</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:complexType name="CapabilitiesRequestType">
  <xs:annotation>
    <xs:documentation>type of CababilitiesRequest</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="BaseRequestType"/>
  </xs:complexContent>
</xs:complexType>
<xs:element name="CapabilitiesResponse" type="CapabilitiesResponseType">
  <xs:annotation>
    <xs:documentation>response with capabilities of passive server</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:complexType name="CapabilitiesResponseType">
  <xs:annotation>
    <xs:documentation>type of CapabilitiesResponse</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="BaseResponseType">
      <xs:sequence>
        <xs:element name="capabilities" type="ServerCapabilitiesType" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>list of capabitlities of all known servers</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="ConnectionInfoType">
  <xs:annotation>
    <xs:documentation>type for info at connection</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="schedule" type="ScheduleType" minOccurs="0">
      <xs:annotation>
```

```
          <xs:documentation>the schedule information (for calcRoutesBefore/calcRoutesAfter)</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="absoluteTime" type="xs:dateTime" use="optional">
      <xs:annotation>
        <xs:documentation>contains an absolute time to specify departure or arrival times (used in DELFI)</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="relativeTime" type="xs:int" use="optional">
      <xs:annotation>
        <xs:documentation>contains a relative offset in minutes to the time in grand-parents baseDepartureTime/baseArrivalTime (used in EU-
Spirit)</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="wait" type="xs:int" use="optional">
      <xs:annotation>
        <xs:documentation>wait will be interpreted as waiting time until the trip can continue in the next system. This value must be 0 if the node was reached
by foot, otherwise it must contain the appropriate change margin for this node. (Every system must set this value appropriately. The search controller will
decide whether this value is sent to the next system or not.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="legs" type="xs:unsignedShort" use="required">
      <xs:annotation>
        <xs:documentation>The number of rides without walks</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="line" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>information about the service</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:element name="ConnectionsRequest" type="ConnectionsRequestType">
    <xs:annotation>
      <xs:documentation>request for non-distributed connection</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="ConnectionsRequestType">
    <xs:annotation>
      <xs:documentation>type of ConnectionRequest</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="BaseRequestType">
        <xs:sequence>
          <xs:element name="fromLocation" type="LocationType">
            <xs:annotation>
              <xs:documentation>Start location (must be completely resolved)</xs:documentation>
```

```
          </xs:annotation>
        </xs:element>
        <xs:element name="toLocation" type="LocationType">
          <xs:annotation>
            <xs:documentation>Destination location (must be completely resolved)</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="fromIT" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>The sequence of types of IT mode at origin location, each element in the sequence represents one activated individual transport
mode. Transport modes not mentioned in the list are deactivated. An empty list causes the server to use his default mode. Each transport mode will occur only
once otherwise the behaviour is not defined.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="toIT" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>Type of IT mode at destination location, see from IT</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="gains" type="OptimizeMode" default="fastest" minOccurs="0">
          <xs:annotation>
            <xs:documentation>Optimizing criteria for route search</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="exclProducts" type="ProductType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>Excluded products (mode types) per partial itinerary, by default no products are excluded</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="features" type="FeatureType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>requested features (attributes) per partial itinerary, by default no features are selected</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="handicapOptions" type="HandicapRequestOptionsType" minOccurs="0">
          <xs:annotation>
            <xs:documentation>requested handicap options to influence individual and public transport usage</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="searchtime" type="xs:dateTime" use="required">
        <xs:annotation>
          <xs:documentation>Requested time (input time of customer)</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="requestDirection" type="RequestDirectionType" use="required">
        <xs:annotation>
```

```
            <xs:documentation>Request mode (search direction for route search), defines whether the search time is a departure or an arrival
</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="useRealtimeData" type="xs:boolean" use="optional" default="false">
          <xs:annotation>
            <xs:documentation>the the calculating system use realtime data or only theoretic (schedule) data</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="connectionsBefore" type="xs:unsignedShort" use="optional">
          <xs:annotation>
            <xs:documentation>Number of desired itineraries before requested time</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="connectionsAfter" type="xs:unsignedShort" use="optional">
          <xs:annotation>
            <xs:documentation>Number of desired itineraries after requested time</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="interval" type="xs:int" use="optional">
          <xs:annotation>
            <xs:documentation>Length of search interval in minutes</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="changeSpeedFactor" type="xs:unsignedShort" use="optional" default="100">
          <xs:annotation>
            <xs:documentation>accelaration/reduction for changes as percent of default speed. Less than 100 means slower changing, more than 100 means faster
changing.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="ConnectionsResponse" type="ConnectionsResponseType">
    <xs:annotation>
      <xs:documentation>Response for non-distributed connection request</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="ConnectionsResponseType">
    <xs:annotation>
      <xs:documentation>type of ConnectionResponse</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="BaseResponseType">
        <xs:sequence>
          <xs:element name="schedules" type="ScheduleType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="dataVersion" type="xs:string" use="optional">
```

```
          <xs:annotation>
            <xs:documentation>indicates the version of realtime data used. Can be used to detect change of data between consecutive requests for a transaction
(unique per passive server)</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="realTimeDataUsed" type="xs:boolean" use="optional" default="false">
          <xs:annotation>
            <xs:documentation>inidicates if passive server used real time info for calculation</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="realTimeDataAvailable" type="xs:boolean" use="optional" default="false">
          <xs:annotation>
            <xs:documentation>inidicates if passive server has real time info</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="CoordType">
    <xs:annotation>
      <xs:documentation>coordinates of locations</xs:documentation>
    </xs:annotation>
    <xs:attribute name="x" type="xs:int" use="required">
      <xs:annotation>
        <xs:documentation>longitude value</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="y" type="xs:int" use="required">
      <xs:annotation>
        <xs:documentation>latitude value</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="z" type="xs:int" use="optional">
      <xs:annotation>
        <xs:documentation>elevation in meters</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="srs" type="xs:string" use="optional" default="EPSG:4326">
      <xs:annotation>
        <xs:documentation>spatial reference system as EPSG coding (default is EPSG:4326 = WGS84)</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:simpleType name="CurrencyType">
    <xs:annotation>
      <xs:documentation>currency type according to ISO 4217</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string"/>
```

```
</xs:simpleType>
<xs:complexType name="OnDemandType">
  <xs:annotation>
    <xs:documentation>type for on-demand traffics</xs:documentation>
  </xs:annotation>
  <xs:attribute name="description" type="xs:string" use="required">
    <xs:annotation>
      <xs:documentation>description text of traffic</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="phoneNumber" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>phone number for booking</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="infoUrl" type="xs:anyURI" use="optional">
    <xs:annotation>
      <xs:documentation>urll to additional information wqebsite</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="bookingUrl" type="xs:anyURI" use="optional">
    <xs:annotation>
      <xs:documentation>url to booking website</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="bookingForerunMinutes" type="xs:short" use="optional">
    <xs:annotation>
      <xs:documentation>time of booking delay</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="FareElementType">
  <xs:annotation>
    <xs:documentation>describes the tariffs between two locations of a journey</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="from" type="LocationType">
      <xs:annotation>
        <xs:documentation>Price valid from location</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="to" type="LocationType">
      <xs:annotation>
        <xs:documentation>Price valid to location</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="tariffs" type="TariffType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
```

```
        <xs:documentation>Price information to this part</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="FeatureType">
  <xs:annotation>
    <xs:documentation>requested feature (see InfoFeature)</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>
<xs:complexType name="HandicapRequestOptionsType">
  <xs:annotation>
    <xs:documentation>request options for handicaps</xs:documentation>
  </xs:annotation>
  <xs:attribute name="noSingleStep" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>single steps should not be part of transfers</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="noStairs" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>stairs shoould not be part of transfers</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="noEscalator" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>escelators should not be part of transfers</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="noElevator" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>elevators should not be part of transfers</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="noRamp" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>ramps should not be part of transfers (ramps have an elevation of 3-12%)</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="needFlatEntrance" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>flat entrances are needed</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="needNearlyFlatEntrance" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>nearly flat entrances are needed</xs:documentation>
```

```
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="IndividualTransport">
    <xs:annotation>
      <xs:documentation>parameters for IT elements of connection</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="params" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>it parameter values</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="InfoCodeType">
    <xs:annotation>
      <xs:documentation>enumeration of info codes</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="ok"/>
      <xs:enumeration value="less_than_requested"/>
      <xs:enumeration value="more_than_requested"/>
      <xs:enumeration value="unknown_feature"/>
      <xs:enumeration value="unknown_product"/>
      <xs:enumeration value="unknown_language"/>
      <xs:enumeration value="unknown_handicap_option"/>
      <xs:enumeration value="origin_replaced"/>
      <xs:enumeration value="destination_replaced"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ITModeType">
    <xs:annotation>
      <xs:documentation>means of individual transport. walk = pedestrian mode, bike = biking mode, taxi = rides by taxi, car = kiss+ride (without parking),
parkRide = park+ride</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="walk"/>
      <xs:enumeration value="bike"/>
      <xs:enumeration value="taxi"/>
      <xs:enumeration value="car"/>
      <xs:enumeration value="parkRide"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="ITParameterType">
    <xs:annotation>
      <xs:documentation>parameters for IT elements of one specific IT means</xs:documentation>
    </xs:annotation>
```

```
    <xs:attribute name="mode" type="ITModeType" use="required">
      <xs:annotation>
        <xs:documentation>The type of individual transport mode</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="minDistance" type="xs:nonNegativeInteger" use="optional">
      <xs:annotation>
        <xs:documentation>Minimal distance for this transport mode, defined in meter</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="maxDistance" type="xs:nonNegativeInteger" use="optional">
      <xs:annotation>
        <xs:documentation>Maximum distance for this transport mode, defined in meter</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="minDuration" type="xs:nonNegativeInteger" use="optional">
      <xs:annotation>
        <xs:documentation>Minimal duration for this transport mode, defined in minutes</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="maxDuration" type="xs:nonNegativeInteger" use="optional">
      <xs:annotation>
        <xs:documentation>maxmum duration for this transport mode, defined in minutes</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="speedFactor" type="xs:float" use="optional">
      <xs:annotation>
        <xs:documentation>factor to reduce/increase speed 1.0=neutral, less 1.0 = decrease, more tham 1.0 = increase. The percentage is a relative term of the
absolute speed for the mode. This absolute speed is set inside the server and not directly available. Negative values cause the server to act as
100%.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="JourneyLocationType">
    <xs:annotation>
      <xs:documentation>journey specific data for a location</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="LocationType">
        <xs:attribute name="departureTime" type="xs:dateTime" use="optional">
          <xs:annotation>
            <xs:documentation>the scheduled  departure time at location</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="arrivalTime" type="xs:dateTime" use="optional">
          <xs:annotation>
            <xs:documentation>the scheduled arrival time at location</xs:documentation>
          </xs:annotation>
```

```
      </xs:attribute>
      <xs:attribute name="departureDelay" type="xs:int" use="optional">
        <xs:annotation>
          <xs:documentation>the delay of departure in minutes</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="arrivalDelay" type="xs:int" use="optional">
        <xs:annotation>
          <xs:documentation>the delay of arrival in minutes</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="track" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>the scheduled departure track</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="actualTrack" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>the real-time departure track</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="departureEstimated" type="xs:boolean" use="optional">
        <xs:annotation>
          <xs:documentation>the departure time is only estimated</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="arrivalEstimated" type="xs:boolean" use="optional">
        <xs:annotation>
          <xs:documentation>the arrival time is only estimated</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="isCancelled" type="xs:boolean" use="optional">
        <xs:annotation>
          <xs:documentation>this stop is cancelled in real time</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:simpleType name="LanguageType">
  <xs:annotation>
    <xs:documentation>language code as ISO-639 (resp. RFC 1766)</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:language"/>
</xs:simpleType>
<xs:element name="LocationsRequest" type="LocationsRequestType">
  <xs:annotation>
    <xs:documentation>request for identification of origin and destination points</xs:documentation>
```

```
    </xs:annotation>
  </xs:element>
  <xs:complexType name="LocationsRequestType">
    <xs:annotation>
      <xs:documentation>type of a single location request</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="BaseRequestType">
        <xs:sequence>
          <xs:element name="entry" type="LocationType">
            <xs:annotation>
              <xs:documentation>contains already known elements of the locations or search patterns like name and region as given by the
user</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="type" type="LocationTypeType" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>list of location types in which the passive server should search</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="maxLocationNum" type="xs:int" use="optional">
          <xs:annotation>
            <xs:documentation>The maximum number of matches which will be returned in case of multiple matches</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="usage" type="LocationUsageType" use="optional">
          <xs:annotation>
            <xs:documentation>usage of the requested location (origin, destination, via)</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="LocationsResponse" type="LocationsResponseType">
    <xs:annotation>
      <xs:documentation>response of a LocationRequest</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="LocationsResponseType">
    <xs:annotation>
      <xs:documentation>type of a single location response</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="BaseResponseType">
        <xs:sequence>
          <xs:element name="location" type="LocationType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
```

```
            <xs:documentation>the matching locations</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="LocationType">
  <xs:annotation>
    <xs:documentation>an origin/destination point or a passed location</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="key" type="LocationKeyType">
      <xs:annotation>
        <xs:documentation>Location key: individual identifier within the requested system, the output of the requested system, usually not referable outside
the requested system</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="location" type="LocationEntryType">
      <xs:annotation>
        <xs:documentation>Location element</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="mapLink" type="MapLinkType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>explicit map links for this location</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="sideInfo" type="SideInfoType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>A list of additional information like local descriptions </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="attrs" type="AttributeType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>More additional information </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="type" type="LocationTypeType">
      <xs:annotation>
        <xs:documentation>the type of the location (can not be LocationTypeType.null)</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="realTimeMessages" type="RealTimeMessageType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>realtime messages concerning the location</xs:documentation>
      </xs:annotation>
    </xs:element>
```

```
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>output name of the requested type, this must be the complete name (i.e. â€˜Stuttgart Hauptbahnhof). </xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="state" use="required">
      <xs:annotation>
        <xs:documentation>state of location. Can be initial (must be initialized when a search controller sets its first request), incomplete (not completely
answered, the search controller must run a further request after verification with the customer) or complete (identified correctly and
individually)</xs:documentation>
      </xs:annotation>
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="initial"/>
          <xs:enumeration value="incomplete"/>
          <xs:enumeration value="complete"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="LocationEntryType">
    <xs:annotation>
      <xs:documentation>describes a location</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="coord" type="CoordType" minOccurs="0">
        <xs:annotation>
          <xs:documentation>coordinates of location</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="topLevelRegionName" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>Can be used to provide additional information about the region i.e. the state or region name</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="topLevelRegionId" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>The key of the top level region</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="name" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>name of location (not necessary containing region name)</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="regionName" type="xs:string" use="optional">
```

```
      <xs:annotation>
        <xs:documentation>name of the region/city/place</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="regionId" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>id of the region/city/place (for german regions the AGS should be used)</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="zip" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>zip code of region/city/place</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="houseno" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>House or street number</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="LocationKeyType">
    <xs:annotation>
      <xs:documentation>key of a location</xs:documentation>
    </xs:annotation>
    <xs:attribute name="providerCode" type="ProviderCodeType" use="required">
      <xs:annotation>
        <xs:documentation>provider code of source system. For transition points this must be 0</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="key" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>alphanumeric id of location. For transition points this must be a known id, for local points it can be a internal only
id</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:simpleType name="LocationTypeType">
    <xs:annotation>
      <xs:documentation>type of location (stop, address, poi, coord, ...)</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="null"/>
      <xs:enumeration value="stop"/>
      <xs:enumeration value="address"/>
      <xs:enumeration value="poi"/>
      <xs:enumeration value="coord"/>
    </xs:restriction>
  </xs:simpleType>
```

```
<xs:simpleType name="LocationUsageType">
  <xs:annotation>
    <xs:documentation>usage of the requested location (origin, destination, via)</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="origin"/>
    <xs:enumeration value="destination"/>
    <xs:enumeration value="via"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="MapLinkType">
  <xs:annotation>
    <xs:documentation>type for a map link</xs:documentation>
  </xs:annotation>
  <xs:attribute name="url" type="xs:anyURI" use="required">
    <xs:annotation>
      <xs:documentation>complete url to map</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="title" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>title of the map in language language</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="language" type="LanguageType" use="optional">
    <xs:annotation>
      <xs:documentation>language of the map and the title</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="mimeType" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>mime type of the map behind the url</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:element name="MultiRequest" type="MultiRequestType">
  <xs:annotation>
    <xs:documentation>request for multiple locations</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:complexType name="MultiRequestType">
  <xs:annotation>
    <xs:documentation>type of multi location request</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="BaseRequestType">
      <xs:sequence>
        <xs:element name="locationRequest" type="LocationsRequestType" minOccurs="0" maxOccurs="unbounded">
```

```
          <xs:annotation>
            <xs:documentation>list of location requests</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="transitionRequest" type="TransitionsRequestType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>list of transition requests</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="partialConnectionRequest" type="PartialConnectionsRequestType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>list of partial connection requests</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="connectionRequest" type="ConnectionsRequestType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>list of connection requests</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="refineRequest" type="RefineConnectionsRequestType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>list of refine connection requests</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="MultiResponse" type="MultiResponseType">
  <xs:annotation>
    <xs:documentation>response for multiple locations</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:complexType name="MultiResponseType">
  <xs:annotation>
    <xs:documentation>type of multi location response</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="BaseResponseType">
      <xs:sequence>
        <xs:element name="locationResponse" type="LocationsResponseType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>list of location responses</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="transitionResponse" type="TransitionsResponseType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>list of transition responses</xs:documentation>
```

```
          </xs:annotation>
        </xs:element>
        <xs:element name="partialConnectionResponse" type="PartialConnectionsResponseType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>list of partial connection responses</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="connectionResponse" type="ConnectionsResponseType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>list of connection responses</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="refineResponse" type="RefineConnectionsResponseType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>list of refine connection responses</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:simpleType name="OptimizeMode">
  <xs:annotation>
    <xs:documentation>type of optimization requested (fastest=optimizing criteria for the fastest itinerary, cheapest=optimizing criteria for the cheapest
itinerary, comfortable=optimizing criteria for a comfortable itinerary)</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="fastest"/>
    <xs:enumeration value="cheapest"/>
    <xs:enumeration value="comfortable"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="PartialConnectionsRequest" type="PartialConnectionsRequestType">
  <xs:annotation>
    <xs:documentation>request for distributed connection calculation</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:complexType name="PartialConnectionsRequestType">
  <xs:annotation>
    <xs:documentation>type of partial connection request</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="BaseRequestType">
      <xs:sequence>
        <xs:element name="origNodes" type="StationNodeType" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>Start nodes</xs:documentation>
          </xs:annotation>
```

```
    </xs:element>
    <xs:element name="destNodes" type="StationNodeType" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Destination nodes</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="startIT" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>optional parameters of IT usage from start location to first stop/station</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="destIT" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>optional parameters of IT usage from last stop/station to destination location</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="exclProducts" type="ProductType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>excluded product classes for the search</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="features" type="FeatureType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>included features for rhe search</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="handicapOptions" type="HandicapRequestOptionsType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>describes hadicaps of user</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="requestDirection" type="RequestDirectionType" use="required">
    <xs:annotation>
      <xs:documentation>request direction of the overall request (selected by the user in the interface)</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="baseDepartureTime" type="xs:dateTime" use="optional">
    <xs:annotation>
      <xs:documentation>absolute departure time as common base of all relative times in departure connection infos</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="baseArrivalTime" type="xs:dateTime" use="optional">
    <xs:annotation>
      <xs:documentation>absolute arrival time as common base of all relative times in arrival connection infos</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="useRealtimeData" type="xs:boolean" use="optional" default="false">
```

```
          <xs:annotation>
            <xs:documentation>the active server can determine if the passive server should use only theoretic (schedule) data or realtime
data</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="calcConnectionArrivalTimes" type="xs:boolean" use="optional" default="false">
          <xs:annotation>
            <xs:documentation>server must calculate arrival times for connections. (departure times are known in connections)</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="calcConnectionDepartureTimes" type="xs:boolean" use="optional" default="false">
          <xs:annotation>
            <xs:documentation>server must calculate departure times for connections. (arrival times are known in connections)</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="calcRouteArrivalTimes" type="xs:boolean" use="optional" default="false">
          <xs:annotation>
            <xs:documentation>server must calculate arrival times for routes. (departure times are known in connections)</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="calcRouteDepartureTimes" type="xs:boolean" use="optional" default="false">
          <xs:annotation>
            <xs:documentation>server must calculate departure times for routes. (arrival times are known in connections)</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="connectionsBefore" type="xs:unsignedShort" use="optional">
          <xs:annotation>
            <xs:documentation>number of connections/routes to calculate before the given time</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="connectionsAfter" type="xs:unsignedShort" use="optional">
          <xs:annotation>
            <xs:documentation>number of connections/routes to calculate after the given time </xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="changeSpeedFactor" type="xs:unsignedShort" use="optional" default="100">
          <xs:annotation>
            <xs:documentation>accelaration/reduction for changes as percent of default speed. Less than 100 means slower changing, more than 100 means faster
changing.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="PartialConnectionsResponse" type="PartialConnectionsResponseType">
    <xs:annotation>
      <xs:documentation>Response for distributed connection calculation request</xs:documentation>
    </xs:annotation>
```

```
  </xs:element>
  <xs:complexType name="PartialConnectionsResponseType">
    <xs:annotation>
      <xs:documentation>type of partial connection response</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="BaseResponseType">
        <xs:sequence>
          <xs:element name="origNodes" type="StationNodeType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>orig nodes of partial connections request. nodes without information will be omitted</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="destNodes" type="StationNodeType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>dest nodes of partial connections request. nodes without information will be omitted</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="dataVersion" type="xs:string" use="optional">
          <xs:annotation>
            <xs:documentation>optional identification of the  data version that the passive server has used for the calculation. Can be used by the active
server to detect data updates between consequetive partial calls.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="realTimeDataUsed" type="xs:boolean" use="optional" default="false">
          <xs:annotation>
            <xs:documentation>indicates if passive server has used real time information</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="realTimeDataAvailable" type="xs:boolean" use="optional">
          <xs:annotation>
            <xs:documentation>indicates if passive server has real time information</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="baseDepartureTime" type="xs:dateTime" use="optional">
          <xs:annotation>
            <xs:documentation>base time for relative departure times in connectionInfos of origNodes</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="baseArrivalTime" type="xs:dateTime" use="optional">
          <xs:annotation>
            <xs:documentation>base time for relative arrival times in connectionInfos of destNodes</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

```
<xs:complexType name="PolygonType">
  <xs:annotation>
    <xs:documentation>type of polygon as sequence of coordinates</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="coords" type="CoordType" minOccurs="3" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>coordinates of polygon. at least 3 coordinates are required</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="isHole" type="xs:boolean" use="optional" default="false">
    <xs:annotation>
      <xs:documentation>the polygon is a hole</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:simpleType name="ProductType">
  <xs:annotation>
    <xs:documentation>allowed products (the list of valid products are defined in the meta data file  InfoProducts)</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>
<xs:simpleType name="ProviderCodeType">
  <xs:annotation>
    <xs:documentation>provider code type (the list of valid provider codes are defined in the meta data. 999 = test provider code)</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:short"/>
</xs:simpleType>
<xs:simpleType name="ProviderTypeType">
  <xs:annotation>
    <xs:documentation>provider type (delfi or eu-spirit) to destinguish the calling system type</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="delfi"/>
    <xs:enumeration value="spirit"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="RealTimeMessageType">
  <xs:annotation>
    <xs:documentation>a real time message provided for a journey or a station</xs:documentation>
  </xs:annotation>
  <xs:attribute name="message" type="xs:string" use="required">
    <xs:annotation>
      <xs:documentation>the message text to be presented to the user</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="sortOrder" type="xs:nonNegativeInteger" use="required">
```

```
      <xs:annotation>
        <xs:documentation>a higher nnumber indicates higher importance</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="code" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>a classification code for the type of the message that can be defined in the meta data</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="title" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>title/summary of the message</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="detailURL" type="xs:anyURI" use="optional">
      <xs:annotation>
        <xs:documentation>an URL to a HTML page with details of this message</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:element name="RefineConnectionsRequest" type="RefineConnectionsRequestType">
    <xs:annotation>
      <xs:documentation>request to refine connections</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="RefineConnectionsRequestType">
    <xs:annotation>
      <xs:documentation>type of refine connections request</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="BaseRequestType">
        <xs:sequence>
          <xs:element name="schedules" type="ScheduleType" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>schedules which should be refined by the called server</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="RefineConnectionsResponse" type="RefineConnectionsResponseType">
    <xs:annotation>
      <xs:documentation>response of refined connections</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="RefineConnectionsResponseType">
    <xs:annotation>
```

```
      <xs:documentation>type of refine connections response</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="BaseResponseType">
        <xs:sequence>
          <xs:element name="schedules" type="ScheduleType" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>the (possibly) refined schedules</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:simpleType name="RequestDirectionType">
    <xs:annotation>
      <xs:documentation>direction of search (departure=search forward and starting at defined time, arrival=search backward and defined time is desired arrival
time)</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="departure"/>
      <xs:enumeration value="arrival"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ReturnCodeType">
    <xs:annotation>
      <xs:documentation>enumeration of valid return codes. OK=successful, ERROR_COM=communication failed, ERROR_SUPPORT=function not supported,
ERROR_NOTSERVE=unable to answer the request, ERROR_DATE=date not during the timetable data period, ERRROR_FORMAT=a format failure of the request, ERROR_DEST=no
transport to destination available, ERROR_VIA=no transport to a transfer station, ERROR_START=no transport from the station of departure,
ERROR_LOCATION_UNKNOWN=unknown location, ERROR_LOCATION_TOO_MANY_HITS=the input was not specific, ERROR_ORIG_EQUAL_DEST=the start location is the same as the
dest locaiton
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="OK"/>
      <xs:enumeration value="ERROR_COMM"/>
      <xs:enumeration value="ERROR_SUPPORT"/>
      <xs:enumeration value="ERROR_NOTSERVE"/>
      <xs:enumeration value="ERROR_DATE"/>
      <xs:enumeration value="ERROR_FORMAT"/>
      <xs:enumeration value="ERROR_START"/>
      <xs:enumeration value="ERROR_DEST"/>
      <xs:enumeration value="ERROR_VIA"/>
      <xs:enumeration value="ERROR_OTHER"/>
      <xs:enumeration value="ERROR_LOCATION_UNKNOWN"/>
      <xs:enumeration value="ERROR_LOCATION_TOO_MANY_HITS"/>
      <xs:enumeration value="ERROR_ORIG_EQUAL_DEST"/>
    </xs:restriction>
```

```
  </xs:simpleType>
  <xs:complexType name="ScheduleElementType">
    <xs:annotation>
      <xs:documentation>leg of a connection</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="from" type="JourneyLocationType">
        <xs:annotation>
          <xs:documentation>departure location of leg</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="passedLocations" type="JourneyLocationType" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>passed locations of PT leg. Must either be empty or filled with all in-between locations including start and destination of this
schedule element, if filled it must have at least two elements (start, destination)</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="to" type="JourneyLocationType">
        <xs:annotation>
          <xs:documentation>arrival location of leg</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="line" type="ServiceType">
        <xs:annotation>
          <xs:documentation>line of PT leg</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="attrs" type="AttributeType" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>additional attributes of leg</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="transfer" type="TransferElementType" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>transfer elements of IT leg</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="providerCodes" type="ProviderCodeType" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>one or more providerCodes to determine which servers have calculated this request</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="onDemandInfo" type="OnDemandType" minOccurs="0">
        <xs:annotation>
          <xs:documentation>additional infos for transport on demand. otherwise omitted.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="realTimeMessages" type="RealTimeMessageType" minOccurs="0" maxOccurs="unbounded">
```

```
      <xs:annotation>
        <xs:documentation>realtime messages concerning the service</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="mapLink" type="MapLinkType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>explicit map links for this schedule element</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="visibleElement" type="xs:boolean" use="optional" default="true">
    <xs:annotation>
      <xs:documentation>should this element be presented to user?</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="isCancelled" type="xs:boolean" use="optional" default="false">
    <xs:annotation>
      <xs:documentation>the schedule element is cancelled in real-time</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="elementDistance" type="xs:int" use="optional">
    <xs:annotation>
      <xs:documentation>distance between from and to location in meters respecting the routes of the vehicles</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="handicapOptionsInData" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>indicates if the passive server has handicap option data for this schedule element</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="roughDelayValues" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>the delays in the journey locations are only approximated values and should be indicated as this by the interface</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="durationMinutes" type="xs:int" use="required">
    <xs:annotation>
      <xs:documentation>duration of the schedule element in minutes</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="ScheduleType">
  <xs:annotation>
    <xs:documentation>a complete journey</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="elems" type="ScheduleElementType" maxOccurs="unbounded">
      <xs:annotation>
```

```
        <xs:documentation>patial schedule elements</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="mapLink" type="MapLinkType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>explicit map link for complete  journey</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="sideInfo" type="SideInfoType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>additional informations for this schedule</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="fares" type="FareElementType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>tariff information of schedule</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="infoMessage" type="xs:int" use="optional">
    <xs:annotation>
      <xs:documentation>optional message text</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="legs" type="xs:unsignedShort" use="required">
    <xs:annotation>
      <xs:documentation>number of rides without walks</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="durationMinutes" type="xs:int" use="required">
    <xs:annotation>
      <xs:documentation>duration of complete schedule in minutes</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="distanceMeters" type="xs:int" use="optional">
    <xs:annotation>
      <xs:documentation>distance of complete schedule in meters</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="journeyNotAvailable" type="xs:boolean" use="optional" default="false">
    <xs:annotation>
      <xs:documentation>indicates if the schedule is not possible (because one of the schedule elements is cancelled)</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="ServerCapabilitiesType">
  <xs:annotation>
    <xs:documentation>type of CapabilitiesResponse</xs:documentation>
```

```
</xs:annotation>
<xs:sequence>
  <xs:element name="supportedLanguages" type="LanguageType" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
      <xs:documentation>contains the supported languages</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="supportedLocationTypes" type="LocationTypeType" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
      <xs:documentation>contains the supported location types (that can be requested in the method Locations)</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="supportedFeatures" type="FeatureType" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
      <xs:documentation>contains the supported features (that can be used as requested features)</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="supportedProducts" type="ProductType" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
      <xs:documentation>contains the supported product codes (that can be used as excluded products)</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="supportedHandicapOptions" type="HandicapRequestOptionsType" minOccurs="0">
    <xs:annotation>
      <xs:documentation>contains the supported handicap options of the server</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="supportedSrs" type="xs:string" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
      <xs:documentation>list of all supported coordinate systems</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="supportedRequests" type="xs:string" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
      <xs:documentation>list of all supported requests of this servers (as method names)</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="supportedCoordRanges" type="PolygonType" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
      <xs:documentation>list of polygons to define the ranges in which the passive server can use locations of type coordinate </xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:sequence>
<xs:attribute name="providerCode" type="ProviderCodeType" use="required">
  <xs:annotation>
    <xs:documentation>the provider code of this server</xs:documentation>
  </xs:annotation>
</xs:attribute>
```

```
    <xs:attribute name="minAPIVersion" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>minimum API version understood by the server</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="maxAPIVersion" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>maximum API version understood by the server</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="dataBeginTime" type="xs:dateTime" use="required">
      <xs:annotation>
        <xs:documentation>begin date of availabe timetable data</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="dataEndTime" type="xs:dateTime" use="required">
      <xs:annotation>
        <xs:documentation>end date of availabe timetable data</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="hasRealTimeData" type="xs:boolean" use="optional">
      <xs:annotation>
        <xs:documentation>idicates if the passive server has  real time data available for calculation</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="ServiceType">
    <xs:annotation>
      <xs:documentation>service attributes of a PT leg</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="sideInfo" type="SideInfoType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="operatorName" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>operator code of the service</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="product" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>textual description of the product. not set for footpaths</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="trainName" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>train name/line number of the service. not set for footpaths</xs:documentation>
      </xs:annotation>
    </xs:attribute>
```

```
  <xs:attribute name="trainString" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>additional train name</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="direction" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>direction of the service</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="lastStop" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>last scheduled stop of the service</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="code" type="SymbolCodeType" use="required">
    <xs:annotation>
      <xs:documentation>symbol code of the service (see InfoSymbol)</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="co2Factor" type="xs:double" use="optional">
    <xs:annotation>
      <xs:documentation>average CO2 factor in g/km</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="particulateMatterFactor" type="xs:double" use="optional">
    <xs:annotation>
      <xs:documentation>average Particulate Matter factor in mg/km</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="tpesFactor" type="xs:double" use="optional">
    <xs:annotation>
      <xs:documentation>average total primary energy supply factor in kWH/100km</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="frequencyInfo" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>textual description of the frequency of the serrvice (e.g. "all 3-5 minutes")</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="SideInfoType">
  <xs:annotation>
    <xs:documentation>additional information</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="additionalInfo" type="xs:string" minOccurs="0">
      <xs:annotation>
```

```
        <xs:documentation>free additional information</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="techtype" type="xs:string" use="required">
    <xs:annotation>
      <xs:documentation>Physical type of element in MIME notation</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="businesstype" type="xs:string" use="required">
    <xs:annotation>
      <xs:documentation>Type of information</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="addtechspec" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>Additional information of this techtype </xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="addbusinessspec" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>Additional information of the businesstype, i.e. name of the map</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="content" type="xs:string" use="required">
    <xs:annotation>
      <xs:documentation>The URI of the element </xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="StationNodeType">
  <xs:annotation>
    <xs:documentation>type of a station node</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="location" type="LocationType">
      <xs:annotation>
        <xs:documentation>the fully identified location of the station node</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="connectionInfos" type="ConnectionInfoType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>contains the connection times and/or schedules for this node</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="wait" type="xs:int" use="optional" default="-1">
    <xs:annotation>
```

```
        <xs:documentation>changing time from one ride to the other in minutes. -1 in a request signals that the passive server should add the changing
time</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="StatisticsValueType">
    <xs:annotation>
      <xs:documentation>type of a statistics value</xs:documentation>
    </xs:annotation>
    <xs:attribute name="methodName" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>name of the API method for this value</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="indicatorName" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>additional indicator name of the statistics value (e.g. active or passive)</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="value" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>(numeric) value of the statistics value</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="type" use="required">
      <xs:annotation>
        <xs:documentation>type of the statistics value. (sum=sum of single values, avg=average of single values, min=minimum of single values, max=maximum of
single values, var=variance of single values, totalDuration=sum of all durations in type sum)</xs:documentation>
      </xs:annotation>
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="sum"/>
          <xs:enumeration value="avg"/>
          <xs:enumeration value="min"/>
          <xs:enumeration value="max"/>
          <xs:enumeration value="var"/>
          <xs:enumeration value="totalDuration"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="providerCode" type="ProviderCodeType" use="optional">
      <xs:annotation>
        <xs:documentation>provider code (active server: provider code of passive server, passive server: provider code of active server). If omitted a summary
of all providers is given.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:simpleType name="SymbolCodeType">
```

```
    <xs:annotation>
      <xs:documentation>numeric code of means as defined in the meta data file InfoCode</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:int"/>
  </xs:simpleType>
  <xs:complexType name="TariffType">
    <xs:annotation>
      <xs:documentation>a single tariff information</xs:documentation>
    </xs:annotation>
    <xs:attribute name="amount" type="xs:decimal" use="required">
      <xs:annotation>
        <xs:documentation>price in currency units</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="currency" type="CurrencyType" use="required">
      <xs:annotation>
        <xs:documentation>currency name like Euro</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="textCode" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>if filled, textcode contains a language independent text code, defined in the meta data tables</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="plainString" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>i.e. 1st class trip or short trip price</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="language" type="LanguageType" use="optional">
      <xs:annotation>
        <xs:documentation>the language of the information</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="TimeSpanType">
    <xs:annotation>
      <xs:documentation>period between two times</xs:documentation>
    </xs:annotation>
    <xs:attribute name="from" type="xs:dateTime" use="required">
      <xs:annotation>
        <xs:documentation>the begin of the time interval</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="to" type="xs:dateTime" use="required">
      <xs:annotation>
        <xs:documentation>the end of the time interval (must not be before from)</xs:documentation>
      </xs:annotation>
```

```
  </xs:attribute>
</xs:complexType>
<xs:complexType name="TransferElementType">
  <xs:annotation>
    <xs:documentation>part of a complex transfer between to legs</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="coord" type="CoordType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>sequence of coordinates to describe the transfer track</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="specificInfo" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation>provider specific additional informations of transfer element</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="transferType" type="TransferTypeType" use="required">
    <xs:annotation>
      <xs:documentation>type of transfer element </xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="name" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>describing name of element</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="class" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>class name of transferType (e.g. road type for roads)</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="duration" type="xs:short" use="optional">
    <xs:annotation>
      <xs:documentation>duration of this part in seconds</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="distance" type="xs:short" use="optional">
    <xs:annotation>
      <xs:documentation>distance of this part in meters</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="isCancelled" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>describes a cancelled element</xs:documentation>
    </xs:annotation>
  </xs:attribute>
```

```
    <xs:attribute name="turnDirectionBefore" type="xs:short" use="optional">
      <xs:annotation>
        <xs:documentation>turn direction at the beginning of this part in degree (0=north, 90=east, 180=south, 270=west)</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="turnInstructionBefore" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>textual description of turn at the beginning of this part</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="visibleElement" type="xs:boolean" use="optional" default="true">
      <xs:annotation>
        <xs:documentation>should this  element be presented to the user?</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:simpleType name="TransferTypeType">
    <xs:annotation>
      <xs:documentation>type of transfer elements</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="road"/>
      <xs:enumeration value="step"/>
      <xs:enumeration value="stairs"/>
      <xs:enumeration value="ramp"/>
      <xs:enumeration value="elevator"/>
      <xs:enumeration value="escalator"/>
      <xs:enumeration value="move-to-infoSymbol-check-in"/>
      <xs:enumeration value="move-to-infoSymbol-check-out"/>
      <xs:enumeration value="already-in-infoSymbol-assuredConnection"/>
      <xs:enumeration value="already-in-infoSymbol-walk"/>
      <xs:enumeration value="already-in-infoSymbol-cycle"/>
      <xs:enumeration value="already-in-infoSymbol-car"/>
      <xs:enumeration value="already-in-infoSymbol-taxi"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="TransactionType">
    <xs:annotation>
      <xs:documentation>transaction (given from caller in request and returned in response)</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="specific" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>provider specific attributes</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="providerCode" type="ProviderCodeType" use="required">
```

```
      <xs:annotation>
        <xs:documentation>provider code of the requesting system</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="providerType" type="ProviderTypeType" use="required">
      <xs:annotation>
        <xs:documentation>provider type of the requesting system</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="sequence" type="xs:unsignedInt" use="required">
      <xs:annotation>
        <xs:documentation>identifiying number of the transaction, typically time_t value</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="subSequence" type="xs:unsignedInt" use="optional">
      <xs:annotation>
        <xs:documentation>optional sub sequence number for logical sub-transactions </xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="TransitionType">
    <xs:annotation>
      <xs:documentation>transition point type</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="location" type="LocationType">
        <xs:annotation>
          <xs:documentation>the fully identified location of the transition point</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="timeDiff" type="xs:int" use="optional">
      <xs:annotation>
        <xs:documentation>limits the time depth of the search: It can end timediff minutes after arriving at the first node.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="wait" type="xs:int" use="optional">
      <xs:annotation>
        <xs:documentation>only EU-Spirit: wait is also used only for transition points. It contains the  time needed for changing the means of transportation
at that station. It is set by the RCC for a GetConnections call and used by a passive server. The value is taken from a startWait or stopWait of a tSchedule
structure. If a transition point is left or reached directly by a train, etc. (without a footpath), wait has to be added. A value of -1 indicates that the
passive server has to determine the change margin by itself.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:element name="TransitionsRequest" type="TransitionsRequestType">
    <xs:annotation>
      <xs:documentation>request of transitions</xs:documentation>
```

```
    </xs:annotation>
  </xs:element>
  <xs:complexType name="TransitionsRequestType">
    <xs:annotation>
      <xs:documentation>type of TransitionRequest</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="BaseRequestType">
        <xs:sequence>
          <xs:element name="startLocation" type="LocationType">
            <xs:annotation>
              <xs:documentation>if search direction is forward, this is the location of the requested server. If search direction is backward, this is the
location of the foreign server.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="destLocation" type="LocationType">
            <xs:annotation>
              <xs:documentation>if search direction is forward, this is the location of the foreign server. If search direction is backward, this is the
location of the requested server.</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="startIT" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>optional parameters of IT usage from start location to first stop/station</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="destIT" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>optional parameters of IT usage from last stop/station to destination location</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="exclProducts" type="ProductType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>Excluded products (mode types) per partial itinerary, by default no products are excluded</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="features" type="FeatureType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>requested features (attributes) per partial itinerary, by default no features are selected</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="handicapOptions" type="HandicapRequestOptionsType" minOccurs="0">
            <xs:annotation>
              <xs:documentation>requested handicap options to influence individual and public transport usage</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="requestTime" type="xs:dateTime" use="optional">
```

```
        <xs:annotation>
          <xs:documentation>Requested time (input time of customer)</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="searchDirection" type="RequestDirectionType" use="required">
        <xs:annotation>
          <xs:documentation>search direction of the overall  itinerary request</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="TransitionsResponse" type="TransitionsResponseType">
  <xs:annotation>
    <xs:documentation>response of transitions</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:complexType name="TransitionsResponseType">
  <xs:annotation>
    <xs:documentation>type of TransitionResponse</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="BaseResponseType">
      <xs:sequence>
        <xs:element name="transitions" type="TransitionType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>identified transition point locations </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="UsageRequest" type="UsageRequestType">
  <xs:annotation>
    <xs:documentation>request of Usage</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:complexType name="UsageRequestType">
  <xs:annotation>
    <xs:documentation>type of UsageRequest</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="BaseRequestType"/>
  </xs:complexContent>
</xs:complexType>
<xs:element name="UsageResponse" type="UsageResponseType">
  <xs:annotation>
```

```
      <xs:documentation>response of Usage</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="UsageResponseType">
    <xs:annotation>
      <xs:documentation>type of UsageResponse</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="BaseResponseType">
        <xs:sequence>
          <xs:element name="statistics" type="StatisticsValueType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>sequence of statistics values</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="capturePeriod" type="TimeSpanType">
            <xs:annotation>
              <xs:documentation>period of statistics data that is returned in statistics elements</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="startupTime" type="xs:dateTime" use="required">
          <xs:annotation>
            <xs:documentation>contains the startup time of the requested server</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:simpleType name="VersionInfoType">
    <xs:annotation>
      <xs:documentation>describes the API version as a version string with dotted notation.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="5.0.0.5"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

WSDL-Specification

The following pages are containing the content of the actual WSDL File.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:schema="delfi5eng" xmlns:tns="delfi5eng" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="delfi5" targetNamespace="delfi5eng">
  <wsdl:types>
    <xs:schema xmlns="delfi5eng" xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="delfi5eng" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="0.1">
      <xs:element name="AllTransitionsRequest" type="AllTransitionsRequestType">
        <xs:annotation>
          <xs:documentation>request to determine all transitions of a passive server</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:complexType name="AllTransitionsRequestType">
        <xs:annotation>
          <xs:documentation>type of AllTransitionRequest</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
          <xs:extension base="BaseRequestType"/>
        </xs:complexContent>
      </xs:complexType>
      <xs:element name="AllTransitionsResponse" type="AllTransitionsResponseType">
        <xs:annotation>
          <xs:documentation>response of request all transitions</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:complexType name="AllTransitionsResponseType">
        <xs:annotation>
          <xs:documentation>type of AllTransitionResponse</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
          <xs:extension base="BaseResponseType">
            <xs:sequence>
              <xs:element name="transitions" type="TransitionType" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
```

```
    <xs:complexType name="AttributeType">
      <xs:annotation>
        <xs:documentation>generic attribute type</xs:documentation>
      </xs:annotation>
      <xs:attribute name="attrMandatory" type="xs:boolean" use="required">
        <xs:annotation>
          <xs:documentation>Whether this text has to be displayed mandatory or not</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="sortOrder" type="xs:short" use="optional" default="-1">
        <xs:annotation>
          <xs:documentation>If sortOrder is less than 0, than all texts can be displayed in any order, in other cases the sortOrder contains positive values
for the order which have to be displayed from low to high</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="textCode" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>if filled, textcode contains a language independent text code, defined in the meta data tables</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="plainString" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>the desired text string can be stored in a direct way </xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="language" type="LanguageType" use="optional">
        <xs:annotation>
          <xs:documentation>the language of the information</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
    <xs:complexType name="BaseRequestType">
      <xs:annotation>
        <xs:documentation>base class of all requests</xs:documentation>
      </xs:annotation>
      <xs:sequence>
        <xs:element name="trans" type="TransactionType"/>
      </xs:sequence>
      <xs:attribute name="preferredLanguage" type="xs:language" use="optional">
        <xs:annotation>
          <xs:documentation>preferred language of text </xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="preferredSrs" type="xs:string" use="optional" default="EPSG:4326">
        <xs:annotation>
          <xs:documentation>preferred srs of coordinates</xs:documentation>
        </xs:annotation>
      </xs:attribute>
```

```
        <xs:attribute name="fallbackLanguage" type="xs:language" use="optional">
          <xs:annotation>
            <xs:documentation>fallback language if preferred language is not supported</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="apiVersion" type="VersionInfoType" use="optional">
          <xs:annotation>
            <xs:documentation>describes the api version of the calling system</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:complexType>
      <xs:complexType name="BaseResponseType">
        <xs:annotation>
          <xs:documentation>base class of all responses</xs:documentation>
        </xs:annotation>
        <xs:sequence>
          <xs:element name="infoCode" type="InfoCodeType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>ok (can be omiitted), less_than_requested (less trips returned than requested), more_than_requested (more trips returned than
requested), unknown_feature (a requested feature is not supported), unknown_product (a requested product is not supported), unknown_language (a requested
language is not supported), unknown_handicap_option (a requested handicap option is not supporterd), origin_replaced (the origin location has been replaced by
a near location), destination_replaced (the destination location has been replaced by a near location)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="infoMessage" type="xs:string" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>additional information messages of the passive server</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="trans" type="TransactionType">
            <xs:annotation>
              <xs:documentation>the transaction of the corresponding request</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="calcTime" type="CalcTimeType" use="required">
          <xs:annotation>
            <xs:documentation>The time the calculation needed- must be the difference between the time stamp when the passive servers receive the request and
the time stamp whenever the answer is sent; time shown in seconds (with fraction)</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="returnCode" type="ReturnCodeType" use="required"/>
        <xs:attribute name="returnOtherText" type="xs:string" use="optional">
          <xs:annotation>
            <xs:documentation>if returnCode is ERROR_OTHER, this field should contain a description of the abnormal state</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="apiVersion" type="VersionInfoType" use="optional">
```

```
    <xs:annotation>
      <xs:documentation>describes the api version of the responding system</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:simpleType name="CalcTimeType">
  <xs:annotation>
    <xs:documentation>calculation time of request in seconds</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:float"/>
</xs:simpleType>
<xs:element name="CapabilitiesRequest" type="CapabilitiesRequestType">
  <xs:annotation>
    <xs:documentation>request to determine capabilities of passive server</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:complexType name="CapabilitiesRequestType">
  <xs:annotation>
    <xs:documentation>type of CababilitiesRequest</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="BaseRequestType"/>
  </xs:complexContent>
</xs:complexType>
<xs:element name="CapabilitiesResponse" type="CapabilitiesResponseType">
  <xs:annotation>
    <xs:documentation>response with capabilities of passive server</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:complexType name="CapabilitiesResponseType">
  <xs:annotation>
    <xs:documentation>type of CapabilitiesResponse</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="BaseResponseType">
      <xs:sequence>
        <xs:element name="capabilities" type="ServerCapabilitiesType" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>list of capabitlities of all known servers</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="ConnectionInfoType">
  <xs:annotation>
    <xs:documentation>type for info at connection</xs:documentation>
```

```
        </xs:annotation>
        <xs:sequence>
          <xs:element name="schedule" type="ScheduleType" minOccurs="0">
            <xs:annotation>
              <xs:documentation>the schedule information (for calcRoutesBefore/calcRoutesAfter)</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="absoluteTime" type="xs:dateTime" use="optional">
          <xs:annotation>
            <xs:documentation>contains an absolute time to specify departure or arrival times (used in DELFI)</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="relativeTime" type="xs:int" use="optional">
          <xs:annotation>
            <xs:documentation>contains a relative offset in minutes to the time in grand-parents baseDepartureTime/baseArrivalTime (used in EU-
Spirit)</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="wait" type="xs:int" use="optional">
          <xs:annotation>
            <xs:documentation>wait will be interpreted as waiting time until the trip can continue in the next system. This value must be 0 if the node was
reached by foot, otherwise it must contain the appropriate change margin for this node. (Every system must set this value appropriately. The search controller
will decide whether this value is sent to the next system or not.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="legs" type="xs:unsignedShort" use="required">
          <xs:annotation>
            <xs:documentation>The number of rides without walks</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="line" type="xs:string" use="optional">
          <xs:annotation>
            <xs:documentation>information about the service</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:complexType>
      <xs:element name="ConnectionsRequest" type="ConnectionsRequestType">
        <xs:annotation>
          <xs:documentation>request for non-distributed connection</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:complexType name="ConnectionsRequestType">
        <xs:annotation>
          <xs:documentation>type of ConnectionRequest</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
          <xs:extension base="BaseRequestType">
```

```
            <xs:sequence>
              <xs:element name="fromLocation" type="LocationType">
                <xs:annotation>
                  <xs:documentation>Start location (must be completely resolved)</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="toLocation" type="LocationType">
                <xs:annotation>
                  <xs:documentation>Destination location (must be completely resolved)</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="fromIT" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>The sequence of types of IT mode at origin location, each element in the sequence represents one activated individual
transport mode. Transport modes not mentioned in the list are deactivated. An empty list causes the server to use his default mode. Each transport mode will
occur only once otherwise the behaviour is not defined.</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="toIT" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>Type of IT mode at destination location, see from IT</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="gains" type="OptimizeMode" default="fastest" minOccurs="0">
                <xs:annotation>
                  <xs:documentation>Optimizing criteria for route search</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="exclProducts" type="ProductType" minOccurs="0" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>Excluded products (mode types) per partial itinerary, by default no products are excluded</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="features" type="FeatureType" minOccurs="0" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>requested features (attributes) per partial itinerary, by default no features are selected</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="handicapOptions" type="HandicapRequestOptionsType" minOccurs="0">
                <xs:annotation>
                  <xs:documentation>requested handicap options to influence individual and public transport usage</xs:documentation>
                </xs:annotation>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="searchtime" type="xs:dateTime" use="required">
              <xs:annotation>
                <xs:documentation>Requested time (input time of customer)</xs:documentation>
              </xs:annotation>
```

```
            </xs:attribute>
            <xs:attribute name="requestDirection" type="RequestDirectionType" use="required">
              <xs:annotation>
                <xs:documentation>Request mode (search direction for route search), defines whether the search time is a departure or an arrival
</xs:documentation>
              </xs:annotation>
            </xs:attribute>
            <xs:attribute name="useRealtimeData" type="xs:boolean" use="optional" default="false">
              <xs:annotation>
                <xs:documentation>the the calculating system use realtime data or only theoretic (schedule) data</xs:documentation>
              </xs:annotation>
            </xs:attribute>
            <xs:attribute name="connectionsBefore" type="xs:unsignedShort" use="optional">
              <xs:annotation>
                <xs:documentation>Number of desired itineraries before requested time</xs:documentation>
              </xs:annotation>
            </xs:attribute>
            <xs:attribute name="connectionsAfter" type="xs:unsignedShort" use="optional">
              <xs:annotation>
                <xs:documentation>Number of desired itineraries after requested time</xs:documentation>
              </xs:annotation>
            </xs:attribute>
            <xs:attribute name="interval" type="xs:int" use="optional">
              <xs:annotation>
                <xs:documentation>Length of search interval in minutes</xs:documentation>
              </xs:annotation>
            </xs:attribute>
            <xs:attribute name="changeSpeedFactor" type="xs:unsignedShort" use="optional" default="100">
              <xs:annotation>
                <xs:documentation>accelaration/reduction for changes as percent of default speed. Less than 100 means slower changing, more than 100 means
faster changing.</xs:documentation>
              </xs:annotation>
            </xs:attribute>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
      <xs:element name="ConnectionsResponse" type="ConnectionsResponseType">
        <xs:annotation>
          <xs:documentation>Response for non-distributed connection request</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:complexType name="ConnectionsResponseType">
        <xs:annotation>
          <xs:documentation>type of ConnectionResponse</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
          <xs:extension base="BaseResponseType">
            <xs:sequence>
```

```
            <xs:element name="schedules" type="ScheduleType" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="dataVersion" type="xs:string" use="optional">
            <xs:annotation>
              <xs:documentation>indicates the version of realtime data used. Can be used to detect change of data between consecutive requests for a
transaction (unique per passive server)</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="realTimeDataUsed" type="xs:boolean" use="optional" default="false">
            <xs:annotation>
              <xs:documentation>inidicates if passive server used real time info for calculation</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="realTimeDataAvailable" type="xs:boolean" use="optional" default="false">
            <xs:annotation>
              <xs:documentation>inidicates if passive server has real time info</xs:documentation>
            </xs:annotation>
          </xs:attribute>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="CoordType">
      <xs:annotation>
        <xs:documentation>coordinates of locations</xs:documentation>
      </xs:annotation>
      <xs:attribute name="x" type="xs:int" use="required">
        <xs:annotation>
          <xs:documentation>longitude value</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="y" type="xs:int" use="required">
        <xs:annotation>
          <xs:documentation>latitude value</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="z" type="xs:int" use="optional">
        <xs:annotation>
          <xs:documentation>elevation in meters</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="srs" type="xs:string" use="optional" default="EPSG:4326">
        <xs:annotation>
          <xs:documentation>spatial reference system as EPSG coding (default is EPSG:4326 = WGS84)</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
    <xs:simpleType name="CurrencyType">
      <xs:annotation>
```

```
      <xs:documentation>currency type according to ISO 4217</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:complexType name="OnDemandType">
    <xs:annotation>
      <xs:documentation>type for on-demand traffics</xs:documentation>
    </xs:annotation>
    <xs:attribute name="description" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>description text of traffic</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="phoneNumber" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>phone number for booking</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="infoUrl" type="xs:anyURI" use="optional">
      <xs:annotation>
        <xs:documentation>urll to additional information wqebsite</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="bookingUrl" type="xs:anyURI" use="optional">
      <xs:annotation>
        <xs:documentation>url to booking website</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="bookingForerunMinutes" type="xs:short" use="optional">
      <xs:annotation>
        <xs:documentation>time of booking delay</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="FareElementType">
    <xs:annotation>
      <xs:documentation>describes the tariffs between two locations of a journey</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="from" type="LocationType">
        <xs:annotation>
          <xs:documentation>Price valid from location</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="to" type="LocationType">
        <xs:annotation>
          <xs:documentation>Price valid to location</xs:documentation>
        </xs:annotation>
```

```
    </xs:element>
    <xs:element name="tariffs" type="TariffType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Price information to this part</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="FeatureType">
  <xs:annotation>
    <xs:documentation>requested feature (see InfoFeature)</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>
<xs:complexType name="HandicapRequestOptionsType">
  <xs:annotation>
    <xs:documentation>request options for handicaps</xs:documentation>
  </xs:annotation>
  <xs:attribute name="noSingleStep" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>single steps should not be part of transfers</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="noStairs" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>stairs shoould not be part of transfers</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="noEscalator" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>escelators should not be part of transfers</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="noElevator" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>elevators should not be part of transfers</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="noRamp" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>ramps should not be part of transfers (ramps have an elevation of 3-12%)</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="needFlatEntrance" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>flat entrances are needed</xs:documentation>
    </xs:annotation>
  </xs:attribute>
```

```
        <xs:attribute name="needNearlyFlatEntrance" type="xs:boolean" use="optional">
          <xs:annotation>
            <xs:documentation>nearly flat entrances are needed</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:complexType>
      <xs:complexType name="IndividualTransport">
        <xs:annotation>
          <xs:documentation>parameters for IT elements of connection</xs:documentation>
        </xs:annotation>
        <xs:sequence>
          <xs:element name="params" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>it parameter values</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
      <xs:simpleType name="InfoCodeType">
        <xs:annotation>
          <xs:documentation>enumeration of info codes</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
          <xs:enumeration value="ok"/>
          <xs:enumeration value="less_than_requested"/>
          <xs:enumeration value="more_than_requested"/>
          <xs:enumeration value="unknown_feature"/>
          <xs:enumeration value="unknown_product"/>
          <xs:enumeration value="unknown_language"/>
          <xs:enumeration value="unknown_handicap_option"/>
          <xs:enumeration value="origin_replaced"/>
          <xs:enumeration value="destination_replaced"/>
        </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="ITModeType">
        <xs:annotation>
          <xs:documentation>means of individual transport. walk = pedestrian mode, bike = biking mode, taxi = rides by taxi, car = kiss+ride (without parking),
parkRide = park+ride</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
          <xs:enumeration value="walk"/>
          <xs:enumeration value="bike"/>
          <xs:enumeration value="taxi"/>
          <xs:enumeration value="car"/>
          <xs:enumeration value="parkRide"/>
        </xs:restriction>
      </xs:simpleType>
      <xs:complexType name="ITParameterType">
```

```
        <xs:annotation>
          <xs:documentation>parameters for IT elements of one specific IT means</xs:documentation>
        </xs:annotation>
        <xs:attribute name="mode" type="ITModeType" use="required">
          <xs:annotation>
            <xs:documentation>The type of individual transport mode</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="minDistance" type="xs:nonNegativeInteger" use="optional">
          <xs:annotation>
            <xs:documentation>Minimal distance for this transport mode, defined in meter</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="maxDistance" type="xs:nonNegativeInteger" use="optional">
          <xs:annotation>
            <xs:documentation>Maximum distance for this transport mode, defined in meter</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="minDuration" type="xs:nonNegativeInteger" use="optional">
          <xs:annotation>
            <xs:documentation>Minimal duration for this transport mode, defined in minutes</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="maxDuration" type="xs:nonNegativeInteger" use="optional">
          <xs:annotation>
            <xs:documentation>maxmum duration for this transport mode, defined in minutes</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="speedFactor" type="xs:float" use="optional">
          <xs:annotation>
            <xs:documentation>factor to reduce/increase speed 1.0=neutral, less 1.0 = decrease, more tham 1.0 = increase. The percentage is a relative term of
the absolute speed for the mode. This absolute speed is set inside the server and not directly available. Negative values cause the server to act as
100%.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:complexType>
      <xs:complexType name="JourneyLocationType">
        <xs:annotation>
          <xs:documentation>journey specific data for a location</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
          <xs:extension base="LocationType">
            <xs:attribute name="departureTime" type="xs:dateTime" use="optional">
              <xs:annotation>
                <xs:documentation>the scheduled  departure time at location</xs:documentation>
              </xs:annotation>
            </xs:attribute>
            <xs:attribute name="arrivalTime" type="xs:dateTime" use="optional">
```

```
        <xs:annotation>
          <xs:documentation>the scheduled arrival time at location</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="departureDelay" type="xs:int" use="optional">
        <xs:annotation>
          <xs:documentation>the delay of departure in minutes</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="arrivalDelay" type="xs:int" use="optional">
        <xs:annotation>
          <xs:documentation>the delay of arrival in minutes</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="track" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>the scheduled departure track</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="actualTrack" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>the real-time departure track</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="departureEstimated" type="xs:boolean" use="optional">
        <xs:annotation>
          <xs:documentation>the departure time is only estimated</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="arrivalEstimated" type="xs:boolean" use="optional">
        <xs:annotation>
          <xs:documentation>the arrival time is only estimated</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="isCancelled" type="xs:boolean" use="optional">
        <xs:annotation>
          <xs:documentation>this stop is cancelled in real time</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:simpleType name="LanguageType">
  <xs:annotation>
    <xs:documentation>language code as ISO-639 (resp. RFC 1766)</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:language"/>
</xs:simpleType>
```

```
      <xs:element name="LocationsRequest" type="LocationsRequestType">
        <xs:annotation>
          <xs:documentation>request for identification of origin and destination points</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:complexType name="LocationsRequestType">
        <xs:annotation>
          <xs:documentation>type of a single location request</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
          <xs:extension base="BaseRequestType">
            <xs:sequence>
              <xs:element name="entry" type="LocationType">
                <xs:annotation>
                  <xs:documentation>contains already known elements of the locations or search patterns like name and region as given by the
user</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="type" type="LocationTypeType" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>list of location types in which the passive server should search</xs:documentation>
                </xs:annotation>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="maxLocationNum" type="xs:int" use="optional">
              <xs:annotation>
                <xs:documentation>The maximum number of matches which will be returned in case of multiple matches</xs:documentation>
              </xs:annotation>
            </xs:attribute>
            <xs:attribute name="usage" type="LocationUsageType" use="optional">
              <xs:annotation>
                <xs:documentation>usage of the requested location (origin, destination, via)</xs:documentation>
              </xs:annotation>
            </xs:attribute>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
      <xs:element name="LocationsResponse" type="LocationsResponseType">
        <xs:annotation>
          <xs:documentation>response of a LocationRequest</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:complexType name="LocationsResponseType">
        <xs:annotation>
          <xs:documentation>type of a single location response</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
          <xs:extension base="BaseResponseType">
```

```
            <xs:sequence>
              <xs:element name="location" type="LocationType" minOccurs="0" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>the matching locations</xs:documentation>
                </xs:annotation>
              </xs:element>
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="LocationType">
        <xs:annotation>
          <xs:documentation>an origin/destination point or a passed location</xs:documentation>
        </xs:annotation>
        <xs:sequence>
          <xs:element name="key" type="LocationKeyType">
            <xs:annotation>
              <xs:documentation>Location key: individual identifier within the requested system, the output of the requested system, usually not referable
outside the requested system</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="location" type="LocationEntryType">
            <xs:annotation>
              <xs:documentation>Location element</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="mapLink" type="MapLinkType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>explicit map links for this location</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="sideInfo" type="SideInfoType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>A list of additional information like local descriptions </xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="attrs" type="AttributeType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>More additional information </xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="type" type="LocationTypeType">
            <xs:annotation>
              <xs:documentation>the type of the location (can not be LocationTypeType.null)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="realTimeMessages" type="RealTimeMessageType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
```

```
            <xs:documentation>realtime messages concerning the location</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required">
        <xs:annotation>
          <xs:documentation>output name of the requested type, this must be the complete name (i.e. â€˜Stuttgart Hauptbahnhof). </xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="state" use="required">
        <xs:annotation>
          <xs:documentation>state of location. Can be initial (must be initialized when a search controller sets its first request), incomplete (not
completely answered, the search controller must run a further request after verification with the customer) or complete (identified correctly and
individually)</xs:documentation>
        </xs:annotation>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="initial"/>
            <xs:enumeration value="incomplete"/>
            <xs:enumeration value="complete"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
    <xs:complexType name="LocationEntryType">
      <xs:annotation>
        <xs:documentation>describes a location</xs:documentation>
      </xs:annotation>
      <xs:sequence>
        <xs:element name="coord" type="CoordType" minOccurs="0">
          <xs:annotation>
            <xs:documentation>coordinates of location</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="topLevelRegionName" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>Can be used to provide additional information about the region i.e. the state or region name</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="topLevelRegionId" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>The key of the top level region</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="name" type="xs:string" use="required">
        <xs:annotation>
          <xs:documentation>name of location (not necessary containing region name)</xs:documentation>
```

```
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="regionName" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>name of the region/city/place</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="regionId" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>id of the region/city/place (for german regions the AGS should be used)</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="zip" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>zip code of region/city/place</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="houseno" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>House or street number</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
    <xs:complexType name="LocationKeyType">
      <xs:annotation>
        <xs:documentation>key of a location</xs:documentation>
      </xs:annotation>
      <xs:attribute name="providerCode" type="ProviderCodeType" use="required">
        <xs:annotation>
          <xs:documentation>provider code of source system. For transition points this must be 0</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="key" type="xs:string" use="required">
        <xs:annotation>
          <xs:documentation>alphanumeric id of location. For transition points this must be a known id, for local points it can be a internal only
id</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
    <xs:simpleType name="LocationTypeType">
      <xs:annotation>
        <xs:documentation>type of location (stop, address, poi, coord, ...)</xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string">
        <xs:enumeration value="null"/>
        <xs:enumeration value="stop"/>
        <xs:enumeration value="address"/>
        <xs:enumeration value="poi"/>
```

```
      <xs:enumeration value="coord"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="LocationUsageType">
    <xs:annotation>
      <xs:documentation>usage of the requested location (origin, destination, via)</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="origin"/>
      <xs:enumeration value="destination"/>
      <xs:enumeration value="via"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="MapLinkType">
    <xs:annotation>
      <xs:documentation>type for a map link</xs:documentation>
    </xs:annotation>
    <xs:attribute name="url" type="xs:anyURI" use="required">
      <xs:annotation>
        <xs:documentation>complete url to map</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="title" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>title of the map in language language</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="language" type="LanguageType" use="optional">
      <xs:annotation>
        <xs:documentation>language of the map and the title</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="mimeType" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>mime type of the map behind the url</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:element name="MultiRequest" type="MultiRequestType">
    <xs:annotation>
      <xs:documentation>request for multiple locations</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="MultiRequestType">
    <xs:annotation>
      <xs:documentation>type of multi location request</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
```

```
      <xs:extension base="BaseRequestType">
        <xs:sequence>
          <xs:element name="locationRequest" type="LocationsRequestType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>list of location requests</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="transitionRequest" type="TransitionsRequestType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>list of transition requests</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="partialConnectionRequest" type="PartialConnectionsRequestType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>list of partial connection requests</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="connectionRequest" type="ConnectionsRequestType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>list of connection requests</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="refineRequest" type="RefineConnectionsRequestType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>list of refine connection requests</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="MultiResponse" type="MultiResponseType">
    <xs:annotation>
      <xs:documentation>response for multiple locations</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="MultiResponseType">
    <xs:annotation>
      <xs:documentation>type of multi location response</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="BaseResponseType">
        <xs:sequence>
          <xs:element name="locationResponse" type="LocationsResponseType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>list of location responses</xs:documentation>
            </xs:annotation>
          </xs:element>
```

```
            <xs:element name="transitionResponse" type="TransitionsResponseType" minOccurs="0" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>list of transition responses</xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="partialConnectionResponse" type="PartialConnectionsResponseType" minOccurs="0" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>list of partial connection responses</xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="connectionResponse" type="ConnectionsResponseType" minOccurs="0" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>list of connection responses</xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="refineResponse" type="RefineConnectionsResponseType" minOccurs="0" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>list of refine connection responses</xs:documentation>
              </xs:annotation>
            </xs:element>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:simpleType name="OptimizeMode">
      <xs:annotation>
        <xs:documentation>type of optimization requested (fastest=optimizing criteria for the fastest itinerary, cheapest=optimizing criteria for the
cheapest itinerary, comfortable=optimizing criteria for a comfortable itinerary)</xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string">
        <xs:enumeration value="fastest"/>
        <xs:enumeration value="cheapest"/>
        <xs:enumeration value="comfortable"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:element name="PartialConnectionsRequest" type="PartialConnectionsRequestType">
      <xs:annotation>
        <xs:documentation>request for distributed connection calculation</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:complexType name="PartialConnectionsRequestType">
      <xs:annotation>
        <xs:documentation>type of partial connection request</xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="BaseRequestType">
          <xs:sequence>
            <xs:element name="origNodes" type="StationNodeType" maxOccurs="unbounded">
```

```
      <xs:annotation>
        <xs:documentation>Start nodes</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="destNodes" type="StationNodeType" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Destination nodes</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="startIT" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>optional parameters of IT usage from start location to first stop/station</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="destIT" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>optional parameters of IT usage from last stop/station to destination location</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="exclProducts" type="ProductType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>excluded product classes for the search</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="features" type="FeatureType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>included features for rhe search</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="handicapOptions" type="HandicapRequestOptionsType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>describes hadicaps of user</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="requestDirection" type="RequestDirectionType" use="required">
    <xs:annotation>
      <xs:documentation>request direction of the overall request (selected by the user in the interface)</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="baseDepartureTime" type="xs:dateTime" use="optional">
    <xs:annotation>
      <xs:documentation>absolute departure time as common base of all relative times in departure connection infos</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="baseArrivalTime" type="xs:dateTime" use="optional">
    <xs:annotation>
      <xs:documentation>absolute arrival time as common base of all relative times in arrival connection infos</xs:documentation>
```

```
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="useRealtimeData" type="xs:boolean" use="optional" default="false">
            <xs:annotation>
              <xs:documentation>the active server can determine if the passive server should use only theoretic (schedule) data or realtime
data</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="calcConnectionArrivalTimes" type="xs:boolean" use="optional" default="false">
            <xs:annotation>
              <xs:documentation>server must calculate arrival times for connections. (departure times are known in connections)</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="calcConnectionDepartureTimes" type="xs:boolean" use="optional" default="false">
            <xs:annotation>
              <xs:documentation>server must calculate departure times for connections. (arrival times are known in connections)</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="calcRouteArrivalTimes" type="xs:boolean" use="optional" default="false">
            <xs:annotation>
              <xs:documentation>server must calculate arrival times for routes. (departure times are known in connections)</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="calcRouteDepartureTimes" type="xs:boolean" use="optional" default="false">
            <xs:annotation>
              <xs:documentation>server must calculate departure times for routes. (arrival times are known in connections)</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="connectionsBefore" type="xs:unsignedShort" use="optional">
            <xs:annotation>
              <xs:documentation>number of connections/routes to calculate before the given time</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="connectionsAfter" type="xs:unsignedShort" use="optional">
            <xs:annotation>
              <xs:documentation>number of connections/routes to calculate after the given time </xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="changeSpeedFactor" type="xs:unsignedShort" use="optional" default="100">
            <xs:annotation>
              <xs:documentation>accelaration/reduction for changes as percent of default speed. Less than 100 means slower changing, more than 100 means
faster changing.</xs:documentation>
            </xs:annotation>
          </xs:attribute>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:element name="PartialConnectionsResponse" type="PartialConnectionsResponseType">
```

```
      <xs:annotation>
        <xs:documentation>Response for distributed connection calculation request</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:complexType name="PartialConnectionsResponseType">
      <xs:annotation>
        <xs:documentation>type of partial connection response</xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="BaseResponseType">
          <xs:sequence>
            <xs:element name="origNodes" type="StationNodeType" minOccurs="0" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>orig nodes of partial connections request. nodes without information will be omitted</xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="destNodes" type="StationNodeType" minOccurs="0" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>dest nodes of partial connections request. nodes without information will be omitted</xs:documentation>
              </xs:annotation>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="dataVersion" type="xs:string" use="optional">
            <xs:annotation>
              <xs:documentation>optional identification of the  data version that the passive server has used for the calculation. Can be used by the active
server to detect data updates between consequetive partial calls.</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="realTimeDataUsed" type="xs:boolean" use="optional" default="false">
            <xs:annotation>
              <xs:documentation>indicates if passive server has used real time information</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="realTimeDataAvailable" type="xs:boolean" use="optional">
            <xs:annotation>
              <xs:documentation>indicates if passive server has real time information</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="baseDepartureTime" type="xs:dateTime" use="optional">
            <xs:annotation>
              <xs:documentation>base time for relative departure times in connectionInfos of origNodes</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="baseArrivalTime" type="xs:dateTime" use="optional">
            <xs:annotation>
              <xs:documentation>base time for relative arrival times in connectionInfos of destNodes</xs:documentation>
            </xs:annotation>
          </xs:attribute>
```

```
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="PolygonType">
  <xs:annotation>
    <xs:documentation>type of polygon as sequence of coordinates</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="coords" type="CoordType" minOccurs="3" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>coordinates of polygon. at least 3 coordinates are required</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="isHole" type="xs:boolean" use="optional" default="false">
    <xs:annotation>
      <xs:documentation>the polygon is a hole</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:simpleType name="ProductType">
  <xs:annotation>
    <xs:documentation>allowed products (the list of valid products are defined in the meta data file  InfoProducts)</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>
<xs:simpleType name="ProviderCodeType">
  <xs:annotation>
    <xs:documentation>provider code type (the list of valid provider codes are defined in the meta data. 999 = test provider code)</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:short"/>
</xs:simpleType>
<xs:simpleType name="ProviderTypeType">
  <xs:annotation>
    <xs:documentation>provider type (delfi or eu-spirit) to destinguish the calling system type</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="delfi"/>
    <xs:enumeration value="spirit"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="RealTimeMessageType">
  <xs:annotation>
    <xs:documentation>a real time message provided for a journey or a station</xs:documentation>
  </xs:annotation>
  <xs:attribute name="message" type="xs:string" use="required">
    <xs:annotation>
      <xs:documentation>the message text to be presented to the user</xs:documentation>
```

```
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="sortOrder" type="xs:nonNegativeInteger" use="required">
      <xs:annotation>
        <xs:documentation>a higher number indicates higher importance</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="code" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>a classification code for the type of the message that can be defined in the meta data</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="title" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>title/summary of the message</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="detailURL" type="xs:anyURI" use="optional">
      <xs:annotation>
        <xs:documentation>an URL to a HTML page with details of this message</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:element name="RefineConnectionsRequest" type="RefineConnectionsRequestType">
    <xs:annotation>
      <xs:documentation>request to refine connections</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="RefineConnectionsRequestType">
    <xs:annotation>
      <xs:documentation>type of refine connections request</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="BaseRequestType">
        <xs:sequence>
          <xs:element name="schedules" type="ScheduleType" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>schedules which should be refined by the called server</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="RefineConnectionsResponse" type="RefineConnectionsResponseType">
    <xs:annotation>
      <xs:documentation>response of refined connections</xs:documentation>
    </xs:annotation>
```

```
    </xs:element>
    <xs:complexType name="RefineConnectionsResponseType">
      <xs:annotation>
        <xs:documentation>type of refine connections response</xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="BaseResponseType">
          <xs:sequence>
            <xs:element name="schedules" type="ScheduleType" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>the (possibly) refined schedules</xs:documentation>
              </xs:annotation>
            </xs:element>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:simpleType name="RequestDirectionType">
      <xs:annotation>
        <xs:documentation>direction of search (departure=search forward and starting at defined time, arrival=search backward and defined time is desired
arrival time)</xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string">
        <xs:enumeration value="departure"/>
        <xs:enumeration value="arrival"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="ReturnCodeType">
      <xs:annotation>
        <xs:documentation>enumeration of valid return codes. OK=successful, ERROR_COM=communication failed, ERROR_SUPPORT=function not supported,
ERROR_NOTSERVE=unable to answer the request, ERROR_DATE=date not during the timetable data period, ERRROR_FORMAT=a format failure of the request, ERROR_DEST=no
transport to destination available, ERROR_VIA=no transport to a transfer station, ERROR_START=no transport from the station of departure,
ERROR_LOCATION_UNKNOWN=unknown location, ERROR_LOCATION_TOO_MANY_HITS=the input was not specific, ERROR_ORIG_EQUAL_DEST=the start location is the same as the
dest locaiton
    </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string">
        <xs:enumeration value="OK"/>
        <xs:enumeration value="ERROR_COMM"/>
        <xs:enumeration value="ERROR_SUPPORT"/>
        <xs:enumeration value="ERROR_NOTSERVE"/>
        <xs:enumeration value="ERROR_DATE"/>
        <xs:enumeration value="ERROR_FORMAT"/>
        <xs:enumeration value="ERROR_START"/>
        <xs:enumeration value="ERROR_DEST"/>
        <xs:enumeration value="ERROR_VIA"/>
        <xs:enumeration value="ERROR_OTHER"/>
        <xs:enumeration value="ERROR_LOCATION_UNKNOWN"/>
```

```
          <xs:enumeration value="ERROR_LOCATION_TOO_MANY_HITS"/>
          <xs:enumeration value="ERROR_ORIG_EQUAL_DEST"/>
        </xs:restriction>
      </xs:simpleType>
      <xs:complexType name="ScheduleElementType">
        <xs:annotation>
          <xs:documentation>leg of a connection</xs:documentation>
        </xs:annotation>
        <xs:sequence>
          <xs:element name="from" type="JourneyLocationType">
            <xs:annotation>
              <xs:documentation>departure location of leg</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="passedLocations" type="JourneyLocationType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>passed locations of PT leg. Must either be empty or filled with all in-between locations including start and destination of
this schedule element, if filled it must have at least two elements (start, destination)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="to" type="JourneyLocationType">
            <xs:annotation>
              <xs:documentation>arrival location of leg</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="line" type="ServiceType">
            <xs:annotation>
              <xs:documentation>line of PT leg</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="attrs" type="AttributeType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>additional attributes of leg</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="transfer" type="TransferElementType" minOccurs="0" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>transfer elements of IT leg</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="providerCodes" type="ProviderCodeType" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>one or more providerCodes to determine which servers have calculated this request</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="onDemandInfo" type="OnDemandType" minOccurs="0">
            <xs:annotation>
              <xs:documentation>additional infos for transport on demand. otherwise omitted.</xs:documentation>
```

```
          </xs:annotation>
        </xs:element>
        <xs:element name="realTimeMessages" type="RealTimeMessageType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>realtime messages concerning the service</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="mapLink" type="MapLinkType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>explicit map links for this schedule element</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="visibleElement" type="xs:boolean" use="optional" default="true">
        <xs:annotation>
          <xs:documentation>should this element be presented to user?</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="isCancelled" type="xs:boolean" use="optional" default="false">
        <xs:annotation>
          <xs:documentation>the schedule element is cancelled in real-time</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="elementDistance" type="xs:int" use="optional">
        <xs:annotation>
          <xs:documentation>distance between from and to location in meters respecting the routes of the vehicles</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="handicapOptionsInData" type="xs:boolean" use="optional">
        <xs:annotation>
          <xs:documentation>indicates if the passive server has handicap option data for this schedule element</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="roughDelayValues" type="xs:boolean" use="optional">
        <xs:annotation>
          <xs:documentation>the delays in the journey locations are only approximated values and should be indicated as this by the
interface</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="durationMinutes" type="xs:int" use="required">
        <xs:annotation>
          <xs:documentation>duration of the schedule element in minutes</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
    <xs:complexType name="ScheduleType">
      <xs:annotation>
        <xs:documentation>a complete journey</xs:documentation>
```

```
</xs:annotation>
<xs:sequence>
  <xs:element name="elems" type="ScheduleElementType" maxOccurs="unbounded">
    <xs:annotation>
      <xs:documentation>patial schedule elements</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="mapLink" type="MapLinkType" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
      <xs:documentation>explicit map link for complete  journey</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="sideInfo" type="SideInfoType" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
      <xs:documentation>additional informations for this schedule</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="fares" type="FareElementType" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
      <xs:documentation>tariff information of schedule</xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:sequence>
<xs:attribute name="infoMessage" type="xs:int" use="optional">
  <xs:annotation>
    <xs:documentation>optional message text</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="legs" type="xs:unsignedShort" use="required">
  <xs:annotation>
    <xs:documentation>number of rides without walks</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="durationMinutes" type="xs:int" use="required">
  <xs:annotation>
    <xs:documentation>duration of complete schedule in minutes</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="distanceMeters" type="xs:int" use="optional">
  <xs:annotation>
    <xs:documentation>distance of complete schedule in meters</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="journeyNotAvailable" type="xs:boolean" use="optional" default="false">
  <xs:annotation>
    <xs:documentation>indicates if the schedule is not possible (because one of the schedule elements is cancelled)</xs:documentation>
  </xs:annotation>
</xs:attribute>
```

```
</xs:complexType>
<xs:complexType name="ServerCapabilitiesType">
  <xs:annotation>
    <xs:documentation>type of CapabilitiesResponse</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="supportedLanguages" type="LanguageType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>contains the supported languages</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="supportedLocationTypes" type="LocationTypeType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>contains the supported location types (that can be requested in the method Locations)</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="supportedFeatures" type="FeatureType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>contains the supported features (that can be used as requested features)</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="supportedProducts" type="ProductType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>contains the supported product codes (that can be used as excluded products)</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="supportedHandicapOptions" type="HandicapRequestOptionsType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>contains the supported handicap options of the server</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="supportedSrs" type="xs:string" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>list of all supported coordinate systems</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="supportedRequests" type="xs:string" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>list of all supported requests of this servers (as method names)</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="supportedCoordRanges" type="PolygonType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>list of polygons to define the ranges in which the passive server can use locations of type coordinate </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="providerCode" type="ProviderCodeType" use="required">
```

```
      <xs:annotation>
        <xs:documentation>the provider code of this server</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="minAPIVersion" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>minimum API version understood by the server</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="maxAPIVersion" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>maximum API version understood by the server</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="dataBeginTime" type="xs:dateTime" use="required">
      <xs:annotation>
        <xs:documentation>begin date of availabe timetable data</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="dataEndTime" type="xs:dateTime" use="required">
      <xs:annotation>
        <xs:documentation>end date of availabe timetable data</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="hasRealTimeData" type="xs:boolean" use="optional">
      <xs:annotation>
        <xs:documentation>idicates if the passive server has  real time data available for calculation</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="ServiceType">
    <xs:annotation>
      <xs:documentation>service attributes of a PT leg</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="sideInfo" type="SideInfoType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="operatorName" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>operator code of the service</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="product" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>textual description of the product. not set for footpaths</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="trainName" type="xs:string" use="optional">
```

```
    <xs:annotation>
      <xs:documentation>train name/line number of the service. not set for footpaths</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="trainString" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>additional train name</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="direction" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>direction of the service</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="lastStop" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>last scheduled stop of the service</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="code" type="SymbolCodeType" use="required">
    <xs:annotation>
      <xs:documentation>symbol code of the service (see InfoSymbol)</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="co2Factor" type="xs:double" use="optional">
    <xs:annotation>
      <xs:documentation>average CO2 factor in g/km</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="particulateMatterFactor" type="xs:double" use="optional">
    <xs:annotation>
      <xs:documentation>average Particulate Matter factor in mg/km</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="tpesFactor" type="xs:double" use="optional">
    <xs:annotation>
      <xs:documentation>average total primary energy supply factor in kWH/100km</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="frequencyInfo" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>textual description of the frequency of the serrvice (e.g. "all 3-5 minutes")</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="SideInfoType">
  <xs:annotation>
    <xs:documentation>additional information</xs:documentation>
```

```
    </xs:annotation>
    <xs:sequence>
      <xs:element name="additionalInfo" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>free additional information</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="techtype" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>Physical type of element in MIME notation</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="businesstype" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>Type of information</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="addtechspec" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>Additional information of this techtype </xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="addbusinessspec" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>Additional information of the businesstype, i.e. name of the map</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="content" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>The URI of the element </xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="StationNodeType">
    <xs:annotation>
      <xs:documentation>type of a station node</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="location" type="LocationType">
        <xs:annotation>
          <xs:documentation>the fully identified location of the station node</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="connectionInfos" type="ConnectionInfoType" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>contains the connection times and/or schedules for this node</xs:documentation>
        </xs:annotation>
```

```
          </xs:element>
        </xs:sequence>
        <xs:attribute name="wait" type="xs:int" use="optional" default="-1">
          <xs:annotation>
            <xs:documentation>changing time from one ride to the other in minutes. -1 in a request signals that the passive server should add the changing
time</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:complexType>
      <xs:complexType name="StatisticsValueType">
        <xs:annotation>
          <xs:documentation>type of a statistics value</xs:documentation>
        </xs:annotation>
        <xs:attribute name="methodName" type="xs:string" use="required">
          <xs:annotation>
            <xs:documentation>name of the API method for this value</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="indicatorName" type="xs:string" use="required">
          <xs:annotation>
            <xs:documentation>additional indicator name of the statistics value (e.g. active or passive)</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="value" type="xs:string" use="required">
          <xs:annotation>
            <xs:documentation>(numeric) value of the statistics value</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="type" use="required">
          <xs:annotation>
            <xs:documentation>type of the statistics value. (sum=sum of single values, avg=average of single values, min=minimum of single values, max=maximum
of single values, var=variance of single values, totalDuration=sum of all durations in type sum)</xs:documentation>
          </xs:annotation>
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="sum"/>
              <xs:enumeration value="avg"/>
              <xs:enumeration value="min"/>
              <xs:enumeration value="max"/>
              <xs:enumeration value="var"/>
              <xs:enumeration value="totalDuration"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="providerCode" type="ProviderCodeType" use="optional">
          <xs:annotation>
            <xs:documentation>provider code (active server: provider code of passive server, passive server: provider code of active server). If omitted a
summary of all providers is given.</xs:documentation>
```

```
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:simpleType name="SymbolCodeType">
  <xs:annotation>
    <xs:documentation>numeric code of means as defined in the meta data file InfoCode</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:int"/>
</xs:simpleType>
<xs:complexType name="TariffType">
  <xs:annotation>
    <xs:documentation>a single tariff information</xs:documentation>
  </xs:annotation>
  <xs:attribute name="amount" type="xs:decimal" use="required">
    <xs:annotation>
      <xs:documentation>price in currency units</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="currency" type="CurrencyType" use="required">
    <xs:annotation>
      <xs:documentation>currency name like Euro</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="textCode" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>if filled, textcode contains a language independent text code, defined in the meta data tables</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="plainString" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>i.e. 1st class trip or short trip price</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="language" type="LanguageType" use="optional">
    <xs:annotation>
      <xs:documentation>the language of the information</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="TimeSpanType">
  <xs:annotation>
    <xs:documentation>period between two times</xs:documentation>
  </xs:annotation>
  <xs:attribute name="from" type="xs:dateTime" use="required">
    <xs:annotation>
      <xs:documentation>the begin of the time interval</xs:documentation>
    </xs:annotation>
  </xs:attribute>
```

```
    <xs:attribute name="to" type="xs:dateTime" use="required">
      <xs:annotation>
        <xs:documentation>the end of the time interval (must not be before from)</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="TransferElementType">
    <xs:annotation>
      <xs:documentation>part of a complex transfer between to legs</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="coord" type="CoordType" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>sequence of coordinates to describe the transfer track</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="specificInfo" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>provider specific additional informations of transfer element</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="transferType" type="TransferTypeType" use="required">
      <xs:annotation>
        <xs:documentation>type of transfer element </xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="name" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>describing name of element</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="class" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>class name of transferType (e.g. road type for roads)</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="duration" type="xs:short" use="optional">
      <xs:annotation>
        <xs:documentation>duration of this part in seconds</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="distance" type="xs:short" use="optional">
      <xs:annotation>
        <xs:documentation>distance of this part in meters</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="isCancelled" type="xs:boolean" use="optional">
```

```
      <xs:annotation>
        <xs:documentation>describes a cancelled element</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="turnDirectionBefore" type="xs:short" use="optional">
      <xs:annotation>
        <xs:documentation>turn direction at the beginning of this part in degree (0=north, 90=east, 180=south, 270=west)</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="turnInstructionBefore" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>textual description of turn at the beginning of this part</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="visibleElement" type="xs:boolean" use="optional" default="true">
      <xs:annotation>
        <xs:documentation>should this  element be presented to the user?</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:simpleType name="TransferTypeType">
    <xs:annotation>
      <xs:documentation>type of transfer elements</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="road"/>
      <xs:enumeration value="step"/>
      <xs:enumeration value="stairs"/>
      <xs:enumeration value="ramp"/>
      <xs:enumeration value="elevator"/>
      <xs:enumeration value="escalator"/>
      <xs:enumeration value="move-to-infoSymbol-check-in"/>
      <xs:enumeration value="move-to-infoSymbol-check-out"/>
      <xs:enumeration value="already-in-infoSymbol-assuredConnection"/>
      <xs:enumeration value="already-in-infoSymbol-walk"/>
      <xs:enumeration value="already-in-infoSymbol-cycle"/>
      <xs:enumeration value="already-in-infoSymbol-car"/>
      <xs:enumeration value="already-in-infoSymbol-taxi"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="TransactionType">
    <xs:annotation>
      <xs:documentation>transaction (given from caller in request and returned in response)</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="specific" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>provider specific attributes</xs:documentation>
```

```
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="providerCode" type="ProviderCodeType" use="required">
        <xs:annotation>
          <xs:documentation>provider code of the requesting system</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="providerType" type="ProviderTypeType" use="required">
        <xs:annotation>
          <xs:documentation>provider type of the requesting system</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="sequence" type="xs:unsignedInt" use="required">
        <xs:annotation>
          <xs:documentation>identifiying number of the transaction, typically time_t value</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="subSequence" type="xs:unsignedInt" use="optional">
        <xs:annotation>
          <xs:documentation>optional sub sequence number for logical sub-transactions </xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
    <xs:complexType name="TransitionType">
      <xs:annotation>
        <xs:documentation>transition point type</xs:documentation>
      </xs:annotation>
      <xs:sequence>
        <xs:element name="location" type="LocationType">
          <xs:annotation>
            <xs:documentation>the fully identified location of the transition point</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="timeDiff" type="xs:int" use="optional">
        <xs:annotation>
          <xs:documentation>limits the time depth of the search: It can end timediff minutes after arriving at the first node.</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="wait" type="xs:int" use="optional">
        <xs:annotation>
          <xs:documentation>only EU-Spirit: wait is also used only for transition points. It contains the  time needed for changing the means of
transportation at that station. It is set by the RCC for a GetConnections call and used by a passive server. The value is taken from a startWait or stopWait of
a tSchedule structure. If a transition point is left or reached directly by a train, etc. (without a footpath), wait has to be added. A value of -1 indicates
that the passive server has to determine the change margin by itself.</xs:documentation>
        </xs:annotation>
      </xs:attribute>
```

```
      </xs:complexType>
      <xs:element name="TransitionsRequest" type="TransitionsRequestType">
        <xs:annotation>
          <xs:documentation>request of transitions</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:complexType name="TransitionsRequestType">
        <xs:annotation>
          <xs:documentation>type of TransitionRequest</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
          <xs:extension base="BaseRequestType">
            <xs:sequence>
              <xs:element name="startLocation" type="LocationType">
                <xs:annotation>
                  <xs:documentation>if search direction is forward, this is the location of the requested server. If search direction is backward, this is the
location of the foreign server.</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="destLocation" type="LocationType">
                <xs:annotation>
                  <xs:documentation>if search direction is forward, this is the location of the foreign server. If search direction is backward, this is the
location of the requested server.</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="startIT" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>optional parameters of IT usage from start location to first stop/station</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="destIT" type="ITParameterType" minOccurs="0" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>optional parameters of IT usage from last stop/station to destination location</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="exclProducts" type="ProductType" minOccurs="0" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>Excluded products (mode types) per partial itinerary, by default no products are excluded</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="features" type="FeatureType" minOccurs="0" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>requested features (attributes) per partial itinerary, by default no features are selected</xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="handicapOptions" type="HandicapRequestOptionsType" minOccurs="0">
                <xs:annotation>
                  <xs:documentation>requested handicap options to influence individual and public transport usage</xs:documentation>
```

```
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="requestTime" type="xs:dateTime" use="optional">
        <xs:annotation>
          <xs:documentation>Requested time (input time of customer)</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="searchDirection" type="RequestDirectionType" use="required">
        <xs:annotation>
          <xs:documentation>search direction of the overall  itinerary request</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="TransitionsResponse" type="TransitionsResponseType">
  <xs:annotation>
    <xs:documentation>response of transitions</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:complexType name="TransitionsResponseType">
  <xs:annotation>
    <xs:documentation>type of TransitionResponse</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="BaseResponseType">
      <xs:sequence>
        <xs:element name="transitions" type="TransitionType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>identified transition point locations </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="UsageRequest" type="UsageRequestType">
  <xs:annotation>
    <xs:documentation>request of Usage</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:complexType name="UsageRequestType">
  <xs:annotation>
    <xs:documentation>type of UsageRequest</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="BaseRequestType"/>
```

```
      </xs:complexContent>
    </xs:complexType>
    <xs:element name="UsageResponse" type="UsageResponseType">
      <xs:annotation>
        <xs:documentation>response of Usage</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:complexType name="UsageResponseType">
      <xs:annotation>
        <xs:documentation>type of UsageResponse</xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="BaseResponseType">
          <xs:sequence>
            <xs:element name="statistics" type="StatisticsValueType" minOccurs="0" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>sequence of statistics values</xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="capturePeriod" type="TimeSpanType">
              <xs:annotation>
                <xs:documentation>period of statistics data that is returned in statistics elements</xs:documentation>
              </xs:annotation>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="startupTime" type="xs:dateTime" use="required">
            <xs:annotation>
              <xs:documentation>contains the startup time of the requested server</xs:documentation>
            </xs:annotation>
          </xs:attribute>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:simpleType name="VersionInfoType">
      <xs:annotation>
        <xs:documentation>describes the API version as a version string with dotted notation.</xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string">
        <xs:enumeration value="5.0.0.4"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:schema>
</wsdl:types>
<wsdl:message name="AllTransitionsRequestMessage">
  <wsdl:part name="body" element="schema:AllTransitionsRequest"/>
</wsdl:message>
<wsdl:message name="AllTransitionsResponseMessage">
  <wsdl:part name="body" element="schema:AllTransitionsResponse"/>
```

```
</wsdl:message>
<wsdl:message name="CapabilitiesRequestMessage">
  <wsdl:part name="body" element="schema:CapabilitiesRequest"/>
</wsdl:message>
<wsdl:message name="CapabilitiesResponseMessage">
  <wsdl:part name="body" element="schema:CapabilitiesResponse"/>
</wsdl:message>
<wsdl:message name="ConnectionsRequestMessage">
  <wsdl:part name="body" element="schema:ConnectionsRequest"/>
</wsdl:message>
<wsdl:message name="ConnectionsResponseMessage">
  <wsdl:part name="body" element="schema:ConnectionsResponse"/>
</wsdl:message>
<wsdl:message name="LocationsRequestMessage">
  <wsdl:part name="body" element="schema:LocationsRequest"/>
</wsdl:message>
<wsdl:message name="LocationsResponseMessage">
  <wsdl:part name="body" element="schema:LocationsResponse"/>
</wsdl:message>
<wsdl:message name="MultiRequestMessage">
  <wsdl:part name="body" element="schema:MultiRequest"/>
</wsdl:message>
<wsdl:message name="MultiResponseMessage">
  <wsdl:part name="body" element="schema:MultiResponse"/>
</wsdl:message>
<wsdl:message name="PartialConnectionsRequestMessage">
  <wsdl:part name="body" element="schema:PartialConnectionsRequest"/>
</wsdl:message>
<wsdl:message name="PartialConnectionsResponseMessage">
  <wsdl:part name="body" element="schema:PartialConnectionsResponse"/>
</wsdl:message>
<wsdl:message name="RefineConnectionsRequestMessage">
  <wsdl:part name="body" element="schema:RefineConnectionsRequest"/>
</wsdl:message>
<wsdl:message name="RefineConnectionsResponseMessage">
  <wsdl:part name="body" element="schema:RefineConnectionsResponse"/>
</wsdl:message>
<wsdl:message name="TransitionsRequestMessage">
  <wsdl:part name="body" element="schema:TransitionsRequest"/>
</wsdl:message>
<wsdl:message name="TransitionsResponseMessage">
  <wsdl:part name="body" element="schema:TransitionsResponse"/>
</wsdl:message>
<wsdl:message name="UsageRequestMessage">
  <wsdl:part name="body" element="schema:UsageRequest"/>
</wsdl:message>
<wsdl:message name="UsageResponseMessage">
  <wsdl:part name="body" element="schema:UsageResponse"/>
```

```
</wsdl:message>
<wsdl:portType name="delfi5PortType">
  <wsdl:operation name="AllTransitions">
    <wsdl:input name="AllTransitionsInput" message="tns:AllTransitionsRequestMessage"/>
    <wsdl:output name="AllTransitionsOutput" message="tns:AllTransitionsResponseMessage"/>
  </wsdl:operation>
  <wsdl:operation name="Capabilities">
    <wsdl:input name="CapabilitiesInput" message="tns:CapabilitiesRequestMessage"/>
    <wsdl:output name="CapabilitiesOutput" message="tns:CapabilitiesResponseMessage"/>
  </wsdl:operation>
  <wsdl:operation name="Connections">
    <wsdl:input name="ConnectionsInput" message="tns:ConnectionsRequestMessage"/>
    <wsdl:output name="ConnectionsOutput" message="tns:ConnectionsResponseMessage"/>
  </wsdl:operation>
  <wsdl:operation name="Locations">
    <wsdl:input name="LocationsInput" message="tns:LocationsRequestMessage"/>
    <wsdl:output name="LocationsOutput" message="tns:LocationsResponseMessage"/>
  </wsdl:operation>
  <wsdl:operation name="Multi">
    <wsdl:input name="MultiInput" message="tns:MultiRequestMessage"/>
    <wsdl:output name="MultiOutput" message="tns:MultiResponseMessage"/>
  </wsdl:operation>
  <wsdl:operation name="PartialConnections">
    <wsdl:input name="PartialConnectionsInput" message="tns:PartialConnectionsRequestMessage"/>
    <wsdl:output name="PartialConnectionsOutput" message="tns:PartialConnectionsResponseMessage"/>
  </wsdl:operation>
  <wsdl:operation name="RefineConnections">
    <wsdl:input name="RefineConnectionsInput" message="tns:RefineConnectionsRequestMessage"/>
    <wsdl:output name="RefineConnectionsOutput" message="tns:RefineConnectionsResponseMessage"/>
  </wsdl:operation>
  <wsdl:operation name="Transitions">
    <wsdl:input name="TransitionsInput" message="tns:TransitionsRequestMessage"/>
    <wsdl:output name="TransitionsOutput" message="tns:TransitionsResponseMessage"/>
  </wsdl:operation>
  <wsdl:operation name="Usage">
    <wsdl:input name="UsageInput" message="tns:UsageRequestMessage"/>
    <wsdl:output name="UsageOutput" message="tns:UsageResponseMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="delfi5Binding" type="tns:delfi5PortType">
  <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="AllTransitions">
    <wsdlsoap:operation soapAction="delfi5eng/AllTransitions"/>
    <wsdl:input name="AllTransitionsInput">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="AllTransitionsOutput">
      <wsdlsoap:body use="literal"/>
```

```
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="Capabilities">
  <wsdlsoap:operation soapAction="delfi5eng/Capabilities"/>
  <wsdl:input name="CapabilitiesInput">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="CapabilitiesOutput">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="Connections">
  <wsdlsoap:operation soapAction="delfi5eng/Connections"/>
  <wsdl:input name="ConnectionsInput">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="ConnectionsOutput">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="Locations">
  <wsdlsoap:operation soapAction="delfi5eng/Locations"/>
  <wsdl:input name="LocationsInput">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="LocationsOutput">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="Multi">
  <wsdlsoap:operation soapAction="delfi5eng/Multi"/>
  <wsdl:input name="MultiInput">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="MultiOutput">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="PartialConnections">
  <wsdlsoap:operation soapAction="delfi5eng/PartialConnections"/>
  <wsdl:input name="PartialConnectionsInput">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="PartialConnectionsOutput">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="RefineConnections">
```

```
      <wsdlsoap:operation soapAction="delfi5eng/RefineConnections"/>
      <wsdl:input name="RefineConnectionsInput">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="RefineConnectionsOutput">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Transitions">
      <wsdlsoap:operation soapAction="delfi5eng/Transitions"/>
      <wsdl:input name="TransitionsInput">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="TransitionsOutput">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Usage">
      <wsdlsoap:operation soapAction="delfi5eng/Usage"/>
      <wsdl:input name="UsageInput">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="UsageOutput">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="delfi5Service">
    <wsdl:port name="delfi5Port" binding="tns:delfi5Binding">
<!--      <wsdlsoap:address location=""/> -->
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```