Universität Stuttgart

Sonderforschungsbereich 627

Umgebungsmodelle für
mobile kontextbezogene Systeme

SFB 627 Bericht Nr. 2006/10

Navigation Component Report

Datum: April 2006

Autor(en): Tobias Drosdol, Frank Dürr, Nicola Hönle, Steffen Volz

CR Klassifikation: E.2, H.2.1, H.2.8

Content

1 Introduction

Navigation was one of the first applications in the context of location-based services. Today, there is a wealth of navigation systems available, especially for cars, and the rise of mobile communication will enable more sophisticated systems which can even take the current traffic situation into account. Small and more powerful devices for mobile communication like modern mobile phones and personal digital assistants (PDAs) will further extend the usage of navigation applications beyond the scope of car navigation.

It is one of our goals to provide a value added service to NEXUS applications to support them in navigational tasks. To realize such a service, we need to store topological information in the Augmented World Schema (AWS). (For an introduction into the ideas, concepts, and terms of the Nexus project and the Nexus platform, please see [Nexus], [InfEx04].) The navigation service will use this information as a basis for the navigation process. Therefore, the Standard Class Schema of the AWS is enriched by several navigational objects and also by various mobile object types. To reduce the modeling effort, we want to be able to integrate existing topological information, e.g. GDF and ATKIS data, into our model, and therefore we need a mapping from these data formats to our topological description in the Augmented World Schema. Furthermore, we are aiming at combining navigation approaches for outdoor and indoor areas as well as for different types of locomotion (intermodal navigation). In order to optimize navigation procedures we further intend to develop techniques allowing for hierarchical algorithms that can be run on different levels of details of the underlying data.

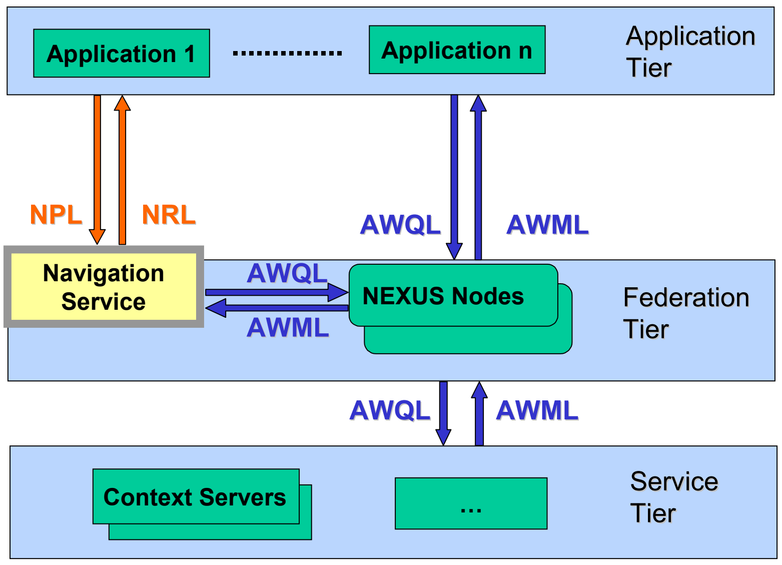


Figure 1: NPL/NRL Interfaces in the Nexus Platform

Figure 1 shows an overview of the Nexus architecture. Basically, different types of context-aware applications can access the federation tier via standardized interfaces. In the case of the navigation service, interfaces to query the service and to receive the result of the navigation are provided. Two languages, namely the Navigation Parameter Language NPL and the Navigation Result Language NRL, will be designed for this purpose in this report. If the navigation service receives a query, it demands the data needed to process this query from a nexus node, the actual federation component, which itself gathers the required data from the different distributed data sources or context servers, respectively, and returns an integrated and consistent data set to the navigation service. On this data basis, the navigation service performs its operations.

In the following report, first the interfaces of the navigation service, namely NPL and NRL, are explained in detail in section 2. In section 3, mobile objects which have to be considered by the navigation service are presented. Section 4 explains the two different types of objects that are represented within the Augmented World Schema for navigation purposes: real world entities which make up navigation networks like roads or railway tracks and topological or navigational objects (nodes and edges) which are used to represent graphs on which shortest path algorithms can run. Section 5 describes how topological information is managed within the Nexus Platform. Finally section 6 provides a summary and discusses open issues and future tasks.

2 Interfaces of the Navigation Service

2.1 Navigation Parameter Language (NPL)

In order to query the navigation service, the NPL (Navigation Parameter Language) has to be introduced. It allows specifying which kind of graph-related query has to be carried out and also includes the parameters that can be used for navigation tasks. Basically, there are two tasks that have to be dealt with by the NEXUS prototype:

- Find shortest route (following a predefined sequence of positions)
- Find best order (traveling salesman problem)

For the development of the specification, it was intended to cover as many possible navigation tasks as possible, so that further changes of the DTD can be reduced to a minimum. Some issues, though, remain to be solved in the future, since we are not able to find the appropriate solution at this stage of the process. They are addressed in the "open issues" section.

2.1.1 Suggestion for DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT npl (query, result)>
<!ELEMENT query (shortest_route | best_order)>
<!ELEMENT shortest_route ((start | loc_start), (intermediate | loc_intermediate)*, end, locomotion+, time?, crite-
rion?)>
<!ELEMENT best_order (start, intermediate*, end?, locomotion+, time?, criterion?)>
<!ELEMENT start (position)>
<!ELEMENT loc_start (start, locomotion+)>
<!ELEMENT intermediate (position)>
<!ELEMENT loc_intermediate (intermediate, locomotion+)>
<!ELEMENT end (position)>
<!ELEMENT position (nol | address | point | name)>
<!ELEMENT nol (#PCDATA)>
<!ELEMENT address EMPTY>
<!ATTLIST address
    country NMTOKENS #REQUIRED
    city NMTOKENS #REQUIRED
    postal_code NMTOKEN #REQUIRED
    street NMTOKENS #REQUIRED
    number NMTOKEN #REQUIRED>
<!ELEMENT point (gml | wkt)>
<!ELEMENT gml (#PCDATA)>
<!ELEMENT wkt (#PCDATA)>
<!ATTLIST gml
    sr_id NMTOKEN #REQUIRED>
<!ATTLIST wkt
    sr_id NMTOKEN #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT locomotion (loc_type, loc_property*, loc_use)>
<!ELEMENT loc_type (#PCDATA)>
<!ELEMENT loc_property EMPTY>
<!ATTLIST loc_property
    name CDATA #REQUIRED
    value CDATA #REQUIRED>
<!ELEMENT loc_use EMPTY>
<!ATTLIST loc_use
    boolean (true | false) "true">
<!ELEMENT time (#PCDATA)>
<!ATTLIST time
    start_end (start | end) #REQUIRED>
<!ELEMENT criterion (#PCDATA)>
<!ELEMENT result EMPTY>
<!ATTLIST result
    format CDATA #REQUIRED>
```

2.1.2 The Language Elements of NPL

Basic Navigation Options

shortest_route

This option means that the result route has to follow a predefined sequence of positions: A fixed order of positions is given in the navigation task. These positions have to be visited in the specified order. Therefore individual locomotion parameters for every part of the route make sense. (Part of the route means the part between two consecutive positions.)

best_order

The optimal order of the positions specified in the navigation task has to be calculated (Traveling Salesman Problem). Only the first and the last position (same as first position if round trip) can be explicitly specified. Individual parts of the trip are not specified at the time you define the navigation task, so parameters for parts of the route make no sense here.

Short Description of the Language Elements

npl

npl is the root element of the navigation parameter language. An npl statement contains two elements: query and result. In the query part the actual navigation query is specified. In the result part the format of the navigation result is specified.

query

The query part of an npl statement contains the actual navigation query. Either shortest_route or best_order queries are allowed, specified by the following parameters.

result

The result part of an npl statement contains a description of the result format. See Section 2.2 for result formats.

start

The start element specifies the first point of the trip, given by a position.

loc_start

The loc_start element also specifies the first point of a trip. Like loc_intermediate it can be used to define individual locomotion parameters for the first part of the route in a shortest_route query.

intermediate

The intermediate elements specify the points which you want to visit in the current trip. An intermediate element is given by a position.

loc_intermediate

The loc_intermediate elements also specify the points which you want to visit in the actual trip. In contrast to the intermediate element you have to specify a position and some locomotion values. loc_intermediate makes sense only in a shortest_route query. It can be used to define individual parameters for every part of the route (part of the route = the part between two consecutive positions). The defined locomotion values in a loc_intermediate element have to be considered in the part of the route succeeding the position given in this loc_intermediate element.
If there are intermediate values instead of loc_intermediate values in a shortest_route query then the global locomotion values defined later in the query are valid.

end

The end element specifies the last point of the trip, given by a position. In best_order queries the end element is optional. If then no end element is given, the optimal end point out of all intermediate points will be calculated.

locomotion

In the locomotion element you can specify your favorite locomotion types or locomotion types you don't want to use. The order of the given locomotion types specifies the preference list of locomotion types for the actual navigation task (or the actual part of the trip, respectively). To describe a locomotion type you have to specify values for loc_type, loc_properties and loc_use.

loc_type
loc_type references the means of transportation given in the Augmented World Model classes (Mobile Objects, see Section 3). Also the value others is possible; then there are no preferred means of transportations. If others is given in a list of locomotion types then all locomotion types which are ahead of others on this list have a higher preference value, but it is possible to use means of transportation not explicitly given in the list. If there is no others value in a list of locomotion types then only the explicitly defined means of transportation have to be used.

loc_properties
loc_properties helps to give more details about the means of transportation (e.g. average speed, maximum speed, costs per km). The properties have to be given in name-value-pairs because it is not possible to predefine a list of all characteristics for all means of transportation.

loc_use
With loc_use it is possible to explicitly exclude locomotion types you don't want to use. To exclude locomotion types makes sense only if the others value is given in the locomotion list.

position
The position is given by a nol, an address, a point, or a name.
A nol references an object in the Augmented World. Every object in the Augmented World contains position information, for further information about the Augmented World and NOLs have a look at [InfEx04].
A point specifies geographical coordinates of a position, given in the Geography Markup Language (gml) format or in the Well-Known Text (wkt) format. You also have to specify the Spatial Reference System (Attribute sr_id) the coordinates refer to.
The last possibility to specify a position is to use a symbolic name (e.g. "main station of Stuttgart").
A address is given by street names, house numbers, postal codes, and city names.
Addresses and names have to be mappable to corresponding geographic coordinates or NOLs.

time
The time element specifies the point in time when you want to start or to end your trip.

criterion
This element helps to specify the criteria to choose the best route for a navigational query. The criteria are not only the preference list of locomotion types, you can also think of shortest time, shortest path, minimal number of changes between means of transportation, minimal costs, and so on. Format and values have to be defined.

result
In the result element the result format can be defined. For more information about result formats see Section 2.2.

2.1.3 Open Issues

There are some requirements we don't know how to deal with yet. Maybe some requirements make no sense at all.

Defining positions in 3D (2D+Time)
In a best_order query this means a 2D+Time Traveling Salesman Problem.

Defining Positions in 3D (2D+Time) you can think of the following features: Is absolute compliance of the time points required? Values for maximal tolerance? Visit positions for time periods, not for time points.

Defining criteria for parts of the route
This only makes sense in shortest_route queries. Then its usage is analog to locomotion values for parts of the route.

Defining the level of detail of navigational graphs
It is possible that there are navigational graphs in different levels of detail. Maybe it makes sense to let the application choose the needed level of detail. (First, concepts for level of detail have to be introduced in the nexus platform at all.)

Mapping Addresses and Names to Geographic Coordinates or NOLs
Positions can be given by nols and geographic coordinates or by addresses and names. Addresses and names have to be mapped to corresponding geographic coordinates or NOLs using lookup services. How (and where) this works is not clear yet.

Defining locomotion types and other values
It is clear that it has to be specified somewhere (with every navigational component) which locomotion types, locomotion properties, and criterion values can be used in a NPL query to a navigational component. How this works is not clear yet.

2.2 *Navigation Result Language (NRL)*

The result of a query formulated in NPL has to be returned in another format, the so-called Navigation Result Language (NRL). Up to now, the only alternative for specifying the result is a sequence of points. But the language can easily be extended according to the needs of applications.

2.2.1 Suggestion for DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT nrl (pointlist)>
<!ELEMENT pointlist (point, point+)>
<!ELEMENT point (gml | wkt)>
<!ELEMENT gml (#PCDATA)>
<!ELEMENT wkt (#PCDATA)>
<!ATTLIST wkt
    sr_id NMTOKEN #REQUIRED>
<!ATTLIST gml
    sr_id NMTOKEN #REQUIRED>
```

2.2.2 The Language Elements of NRL

nrl
nrl is the root element of the Navigation Result Language.

pointlist
Up to now, a pointlist (list of points) is the only alternative for specifying the result. A pointlist contains at least two points.

point
A point specifies geographical coordinates of a position, given in the Geography Markup Language (gml) format or in the Well-Known Text (wkt) format. You also have to specify the Spatial Reference System (Attribute sr_id) the coordinates refer to.

2.2.3 Open Issues

We need more result formats: NavigationalObjects of the Augmented World, figures, and so on. We should support the Navigation Markup Language NVML [NVML].

3 Mobile Objects in the Augmented World Model

Mobile object classes are used in the process of navigation to define restrictions based on the mobile object type. For instance, some roads may be prohibited for heavy trucks, bikes, pedestrians, etc. Traffic rules were analyzed to define classes of mobile objects and an appropriate class hierarchy to be able to express such restrictions.
In the future, mobile object types will also be used for intermodal navigation to define sets of vehicles types which should (not) be used to get from location *A* to *B* ("I want to get from here to the city campus of the University of Stuttgart by public transport, and I am also willing to walk short distances if necessary.").
Although we think, we are able to express at least the common restrictions for road traffic with the currently modeled object types, there certainly will be further refinements of the class hierarchy in the future (especially for non-road traffic and in the context of intermodal navigation). Therefore, extensibility of the class hierarchy was one of the major design goals.

3.1 Class Hierarchy

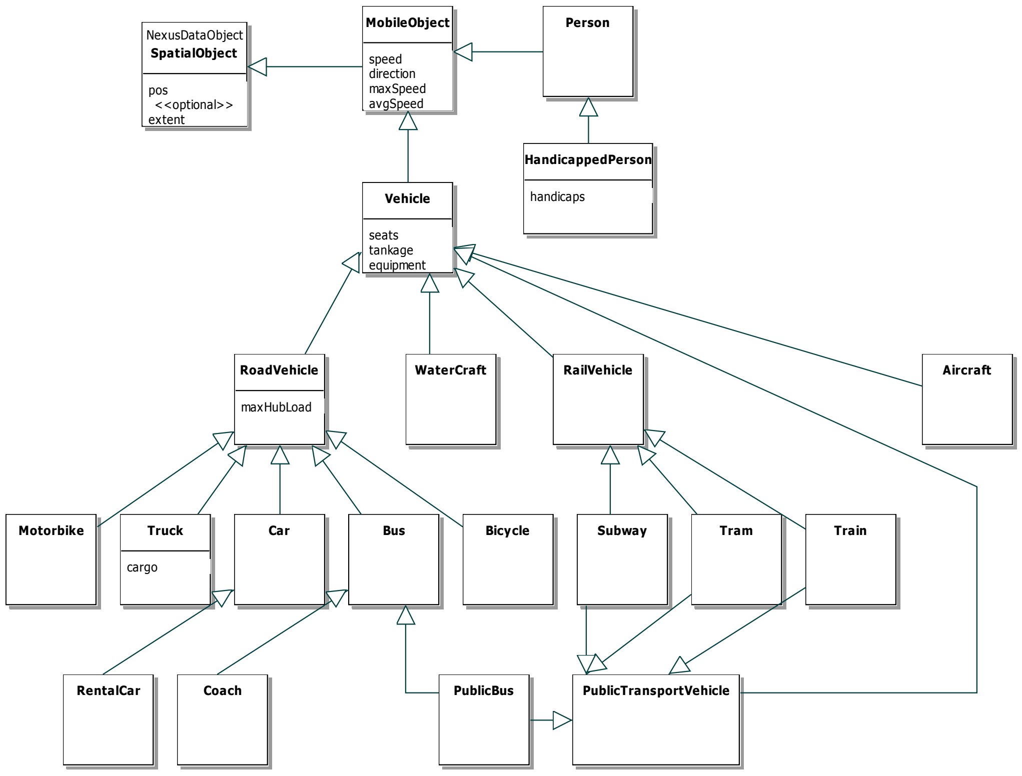


Figure 2: Class Hierarchy MobileObjects

3.2 Explanation

MobileObject
Base class for mobile objects.

- Speed **speed**: current speed.
- Direction **direction**: current direction of movement.
- Speed **maxSpeed**: maximum speed.
- Speed **avgSpeed**: average speed.

Person
A person (pedestrian).

Handicapped Person
A handicapped person.

- SetOfHandicaps handicaps: Handicaps.

Handicap
Different types of handicaps. Possible values are:

- **visually**: Person is visually handicapped.
- **walk**: Person cannot walk or can only hardly walk.

Vehicle
Base class for vehicles.

- int **seats**: number of seats.
- Volume **tankage**: tankage.
- SetOfEquipment **equipment**: the equipment of the vehicle.

Equipment
Different types of vehicle equipment. Possible values are:

- **snowchains**: vehicle has snow chains.

RailVehicle
Base class for rail vehicles.

Train
A train.

Tram
A tram.

Subway
A subway train.

RoadVehicle
Base class for road vehicles.

- Weight **maxHubload**: maximum hub load (kg).

Bus
A bus.

Car
A car.

Taxi
A taxi.

RentalCar
A rental car.

Truck
A truck.

- SetOfCargo **cargo**: loaded cargo.

Cargo
Different types of cargo. Possible values are:

- **dangerous**: dangerous cargo
- **hazardous_to_water**: cargo is hazardous to water

Bicycle
A bicycle.

Motorbike
A motorbike.

WaterCraft
Base class for water crafts.

Aircraft
Base class for aircrafts.

PublicTransportVehicle
Base “class” for public transport vehicles.

4 Geography and Topology in the AWS

It is the key idea of our approach to distinguish between geography and topology, i.e. between the geometric road run (e.g. curved roads, straight roads, etc.) on the one hand and the neighborhood relations between roads (road *A* is connected to road *B* at the junction *C*) on the other hand. The geographical representation can be used to visualize objects by drawing maps, and to get information about the length of a road section, which is needed for instance to find the shortest path, etc. The topological or navigational representation gives information of how to get from location *A* to *B*. Both types of representation shall be stored in different object classes, thus allowing us to provide exactly that representation which is most adequate for a certain task.
Figure 3 shows an overview of the geographical classes on the one side and the topological classes needed for navigation purposes on the other side. In this section, we will first have a closer look at the geographical classes. Then we will elaborate on the classes needed for navigation and on the algorithms used to perform shortest path analyses.

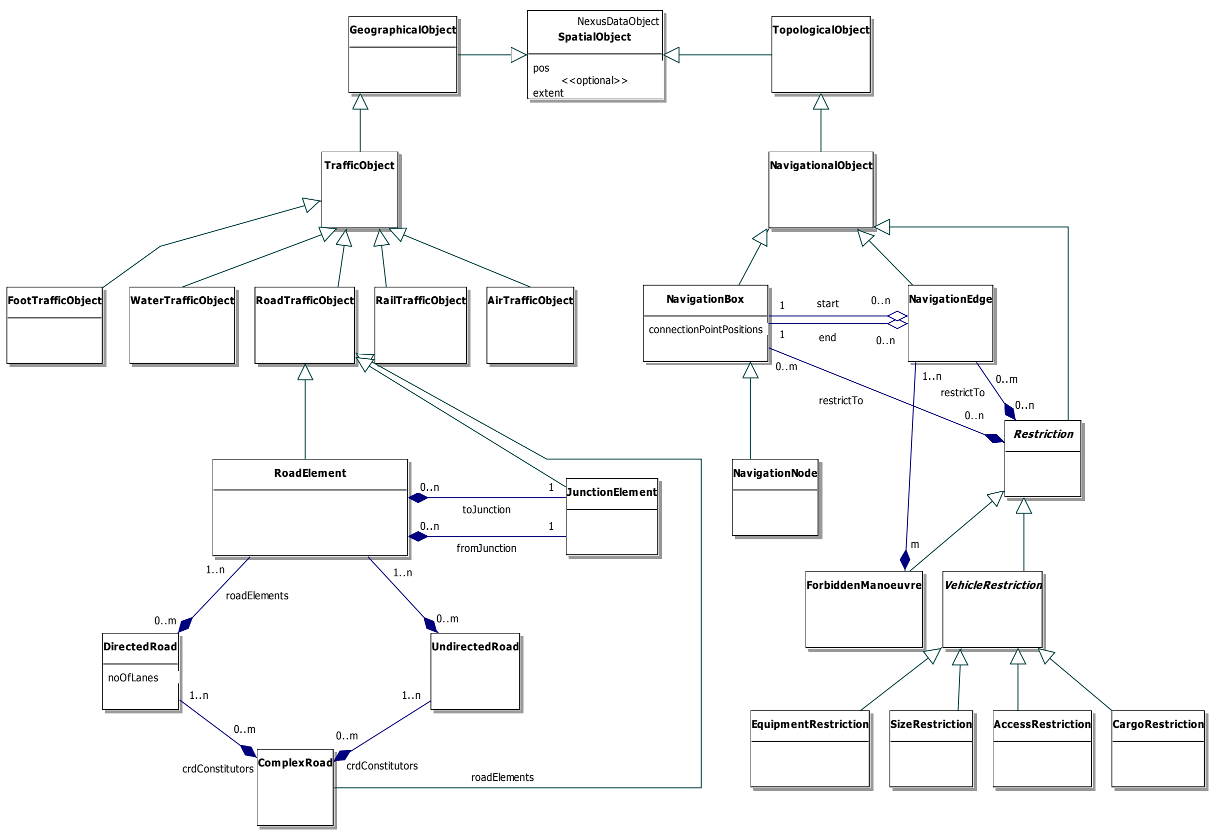


Figure 3: Topological and Geographical Classes in the AWM

4.1 Geographical Representations

Geographical objects used to represent real world navigation networks are called traffic objects according to our terminology. We distinguish between different types of traffic: road traffic, water traffic, air traffic, and rail traffic. In the following, we will describe the road traffic classes in detail. The other types of traffic are part of future work.

As we use test data for Germany, most example values refer to German categories. The work of modeling traffic objects is still in progress, so what is described here should not be considered as a final state.

4.1.1 Class Hierarchy

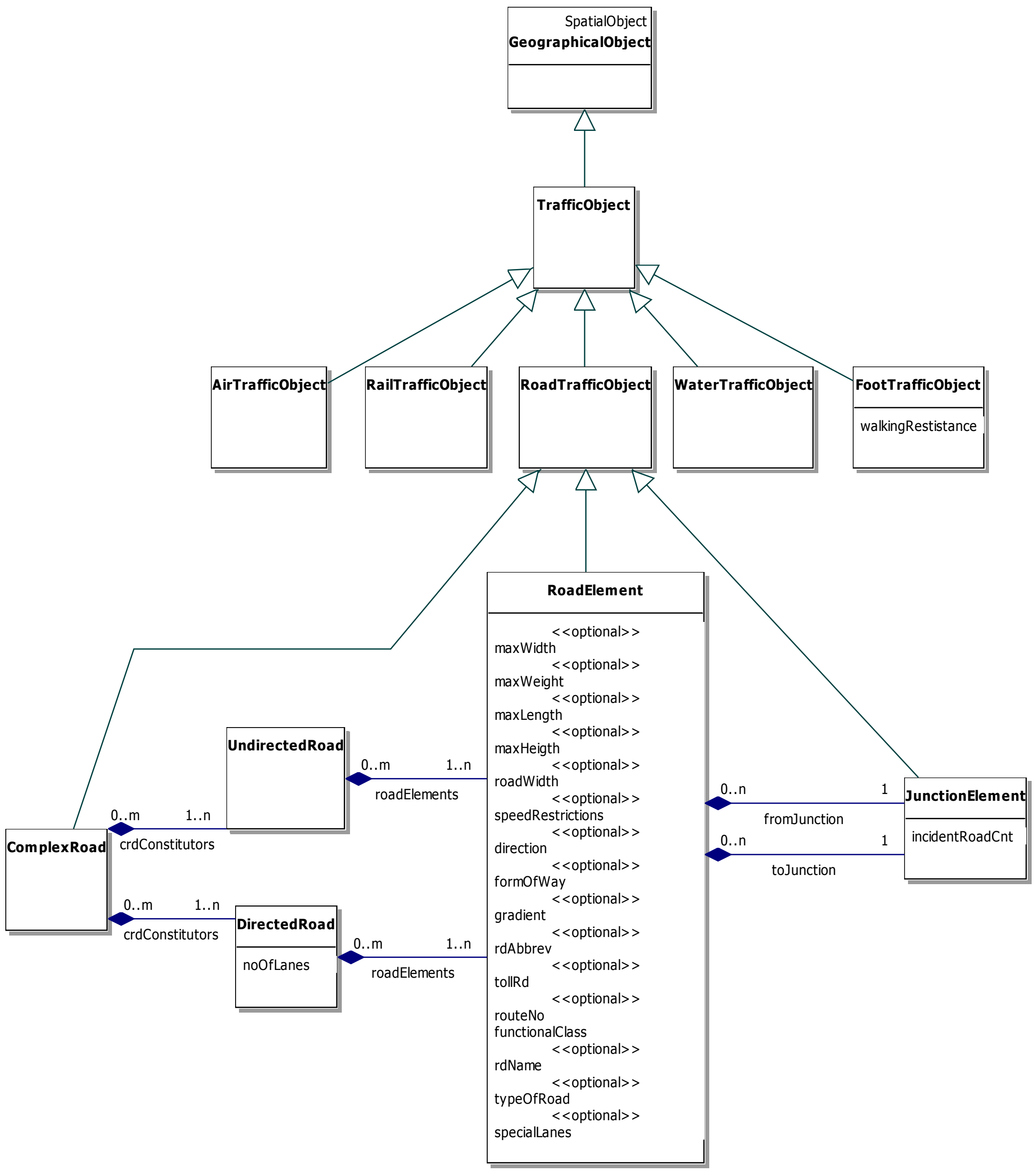


Figure 4: UML representation of road traffic objects in the AWS including cardinalities

4.1.2 Explanation

GeographicalObject
Base class for geographical objects.

- Boolean **walkable**: is this object walkable
- FaceType faceType: type of face (see below)

FaceType
Different types of faces. Possible values are:

- **fixed**: fixed ground
- **grass**
- **wood**
- **water**

TrafficObject
Base class for all geographical objects related to any kind of traffic.

RoadTrafficObject
Base class for all geographical objects related to road traffic.

RailTrafficObject
Base class for all geographical objects related to rail traffic.

AirTrafficObject
Base class for all geographical objects related to air traffic.

WaterTrafficObject
Base class for all geographical objects related to water traffic.

FootTrafficObject
Base class for all geographic objects related to foot traffic.

RoadElement
A road element.

- NOL **fromJunction**: references the From-Junction of the road element.
- NOL **toJunction**: references the To-Junction of the road element.
- Length **max_width** (optional): maximum allowed width of vehicles using this road.
- Weight **max_weight** (optional): maximum allowed weight of vehicles using this road.
- Length **max_length** (optional): maximum allowed length of vehicles using this road.
- Length **max_height** (optional): maximum allowed height of vehicles using this road.
- Length **road_width** (optional): width of road.
- SetOfSpeedRestrictions **vehicle_speed_restrictions** (optional): list of vehicles and their individual speed restrictions (e.g. PKW: 80 km/h, LKW: 60 km/h, etc.)
- Speed **speed_restriction** (optional): maximum allowed speed on this road.
- SetOfSpecialLanes **special_lanes** (optional): describes the special lanes of this road (e.g. lanes for bikes; see below)
- Direction **direction** (optional): describes in which direction this road can be traversed (see below)
- FormOfWay **form_of_way** (optional): describes the properties of a road element (see below)
- Gradient **gradient** (optional): the gradient of the road in degrees (the sign shows if the road is going up or down).
- String **rdAbbrev** (optional): abbreviation.
- Boolean **tollRd** (optional): describes if a toll is required to use this road.
- String **routeNo** (optional): number of route
- FunctionalClass **funct_class** (optional): functional class of this road (see below).
- String **rdName** (optional): name of the road
- TypeOfRoad **rdType** (optional): type of the road (see below)

TypeOfSpecialLane
Different types of special road lanes. Possible values are:

- **none:** no special lane
- **stop_lane:** special lanes, e.g. along an Autobahn, to stop in case of breakdowns, etc.
- **bus_lane:** special lane for buses and taxis
- **agricultural_vehicle_lane:** lane for farm vehicle traffic
- **pedestrian**: special lane for pedestrians
- **bike**: special lane for bikes
- **both**: special lanes for bikes and pedestrians

TypeOfDirection
Describes in which direction this road can be traversed. Possible values are:
- **open**: road can be traversed in forward direction.
- **rev_open**: road can be traversed in reverse direction
- **both_open**: road can be traversed in both directions
- **both_closed**: road is closed in both directions

FormOfWay
Describes the properties of a road element. Possible values are:
- **part_of_roundabout:** road is part of a roundabout
- **part_of_motorway**: road is part of a motorway.

FunctionalClass
This class describes different functional classes of roads, depending on their importance. Possible values are: Main roads, first class roads, second class roads, etc.

TypeOfRoad
This class describes different types of roads. Possible values are: **gemeindestrasse**, **kreisstrasse, bundesstrasse, europastrasse, autobahn, etc.**

JunctionElement
A junction element.
- int **incidentRoadCnt**: number of road elements which are incident

UndirectedRoad
A road composed of simple objects.
- ListOfRoadElements **road_elements**: a list of NOLs identifying all road elements which make up the undirected road.

DirectedRoad
A road composed of simple objects.
- ListOfRoadElements **road_elements**: a list of NOLs identifying all road elements which make up the directed road.
- int **no_lanes** (optional): number of lanes

ComplexRoad
A road composed of complex objects.
- ComplexRoadConstitutors crdConstitutors: a list of NOLs identifying all undirected and directed roads which make up the complex road.

4.1.3 Integrating Road Data from different sources into the AWS

In order to represent existing road databases according to the Augmented World Schema these data formats have to be mapped onto the geographical objects of the AWS. This process will be described in the following sections. The mapping allows on the one hand a further use of already available data. On the other hand, data from multiple sources can be federated since they are available in a common data format.

Existing Data Models for Road Data

The growing popularity of car navigation applications in recent years has lead to a multiple acquisition of road data in different formats. In the following sections the two most common data models for road data in Germany will be introduced. The Geographic Data File (GDF) is used by the major digital road data suppliers, while ATKIS is a topographic and cartographic database. Both formats are acquired in an approximate scale of 1:25 000.

The GDF Data Model

GDF 3.0 (CEN TC 278, 1995) is a European standard that is used to describe and transfer road networks and road related data. It is much more than a GIS data format, because GDF regulates how data should be captured and how features and their relationships have to be defined. Features correspond to "real world objects" like streets and railways or administrative areas. The feature category defines the type of representation of a feature. Objects represented by a point, line or area are called *simple features*, while *complex features* are composed of a group of simple features. Features contain geometrical as well as topological information. Every feature belongs to a specific feature class. The following feature classes are important for road traffic:

- Junction (point): feature bounding a Road Element
- Road Element (line): smallest independent unit of the road network having a Junction at each end
- Road (complex): contains different Road Elements
- Intersection (complex): composed by Junctions and Road Elements

Figure 5 illustrates the relations between simple and complex road traffic features.

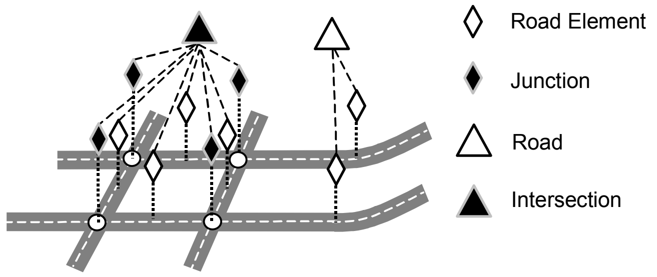


Figure 5: Simple GDF features can be aggregated to complex ones (after Walter, 1997)

Features can also share relationships. Relationships are used to express associations between two or more features like, for example, a prohibited manoeuvre from one road to another.

The ATKIS Data Model

The data model of ATKIS (Authoritative Topographic Cartographic Information System) has been developed by survey offices of the federal states of Germany (AdV, 1998). It is structured into seven functional classes like e.g. settlement, vegetation and traffic. Each functional class is again subdivided into object groups. Concerning traffic, its object groups are road traffic, railway traffic, air traffic, etc. An object group includes several object types. Object types specify the geometrical representation and the attributes of real world objects. In case of road traffic, defined object types are for example roads, squares or ways. Instances of the object types are the ATKIS objects which are built up by object parts. Object parts contain the geometry of ATKIS objects as vector elements. Therefore, each ATKIS object comprises at least one object part. In case of roads, further object parts are established, if the topology (in case of crossings) or the attributes (e.g. the road width) of an object change. ATKIS demands to build complex objects in case that a road has two or more physically separated lanes.

Mapping existing Road Data Models onto the Augmented World Schema
Concerning the mapping of existing road data models onto the object classes of the Augmented World Schema, it is important to preserve the original properties of features as far as possible. However, as it has been outlined in the previous section, both ATKIS and GDF contain complex objects. The rules to build these complex objects considerably differ in the two databases, i.e. there is no unique way of representing the different kinds of complex objects in the Augmented World Schema. For this reason we decided to develop mapping rules from the source databases to the AWS for the simple object types only. Since we need complex structures as well, we built our own rules within Nexus to form these objects.
First, we established object classes within the AWS to represent simple features, namely junctions and road segments. Afterwards, the XML schema definition of the Augmented World Modeling Language (AWML) could be extended to road traffic objects and thus the simple objects of the original GDF and ATKIS formats can be converted into AWML.

Within GDF, there are two types of simple objects: Junctions and Road Elements. In ATKIS, junctions are not explicitly modeled but can be easily derived from the underlying data (as topological connections or endpoints of object parts). Object parts within ATKIS represent road segments similar to the Road Elements of GDF. Thus we introduced two simple object types within the AWS, namely JunctionElement and RoadElement into which all simple objects of the source data sets can be converted in a 1:1 fashion. These two classes are derived from the abstract class *StaticObject* which is the superclass of all static spatial objects (like buildings, streets, etc.) within the AWS. To both object classes of the AWS, necessary attributes have been assigned. Only some most basic of them were declared mandatory.
However, if we only introduced the simple objects, information available in the source data, namely the formation of complex objects would be lost. In order to avoid this, we also introduced a relation object storing references to all simple objects which have together made up a complex object in the source data set. This relation is called BelongsToFormerComplexType, and has as its attributes the references to the simple objects that constituted a complex object in the source data and the name of the complex object type in the source data.
Within the AWS, we established 3 types of complex objects:

- UndirectedRoad: contains 1 to n RoadElements that have the same street name and that can be traversed in both directions
- Directed Road: contains 1 to n RoadElements that have the same street name and that can only be traversed in the same direction
- ComplexRoad: contains at least 1 DirectedRoad and 0 to n UndirectedRoads, with all constituting objects having the same street name

4.2 Topological (Navigational) Representations

Navigational objects are used to model topological information needed to do navigation. Navigation is based on a definition of possible paths between two locations, namely the start and the destination, respectively. Together with suitable metrics for these paths, the navigation service can calculate the fastest path, the shortest path, etc.

4.2.1 Requirements

A topological model representing connections between neighboring locations is required in order to define paths between the start and the destination of the navigation request. Since we aim at a navigation service that supports indoor navigation as well as outdoor navigation, the topological model must be applicable to indoor and outdoor locations as well. The same is true for different types of locomotion to be able to perform intermodal navigation where the user can switch between different locomotion types (walk, bike ride, subway, etc.) during the navigation.

Furthermore, our concept should allow for the definition of starting points and destinations on different levels of granularity. For instance, the target could be a city, a building, a room, or some location within a room. Moreover, a modular and hierarchical concept is highly desirable since this allows for the easy distribution of parts of the model on different servers and at the same time facilitates efficient routing algorithms.

4.2.2 Topology Model

The basic building block of the topology model is the "Navigation Box" (NB). A NB represents a location together with the location's connection points to neighboring locations. For each NB, a number of functions can be defined that represent the distance, time, or any other metric to get from one connection point to another connection point via the NB's location. Furthermore, a NB can be refined by a set of other NBs. A NB can be connected to a neighboring location at corresponding connection points by a Navigation Edge (NE). That means, a NE defines the possibility to get from one location to a neighboring location at a certain connection point. A NE connects exactly two NBs. NEs or transitions from one NB to another, respectively, can be legally or physically restricted. Such legal and physical restrictions are modeled explicitly by various classes of restrictions within the AWS. We distinguish between two basic classes of restrictions:
ForbiddenManoeuvres, e.g. prohibited U-/left/right turns
Vehicle restrictions, referring to certain vehicle properties, e.g. weight/width/height/length restrictions
Specifications of NBs are the Navigation Nodes (NN), which must only contain either one or two connection points. With the concept of Navigation Boxes also intermodal scenarios can be realized, since connection points can also link NBs that have been created for different locomotion types (e.g., car and pedestrian navigation, see example 1)

Example 1 (outdoor)

Consider for instance NB1 in the scenario depicted in Figure 6. NB1 represents a building with two entrances. Each entrance is a connection point (CP1.1, CP1.2) of NB1. NB2 defines a road element with three connection points: at CP2.1 and at CP2.2, the road element is connected to neighboring road elements; CP2.1 connects the road element to the building represented by NB1. The connection between the building and the road element at CP1.1 and CP2.1 is defined by NE1. Function f2.1 models the physical distance to cross the road element from one connection point to another. For instance, to get from one side of the road element (CP2.1) to the other (CP2.2) it takes 200 m.

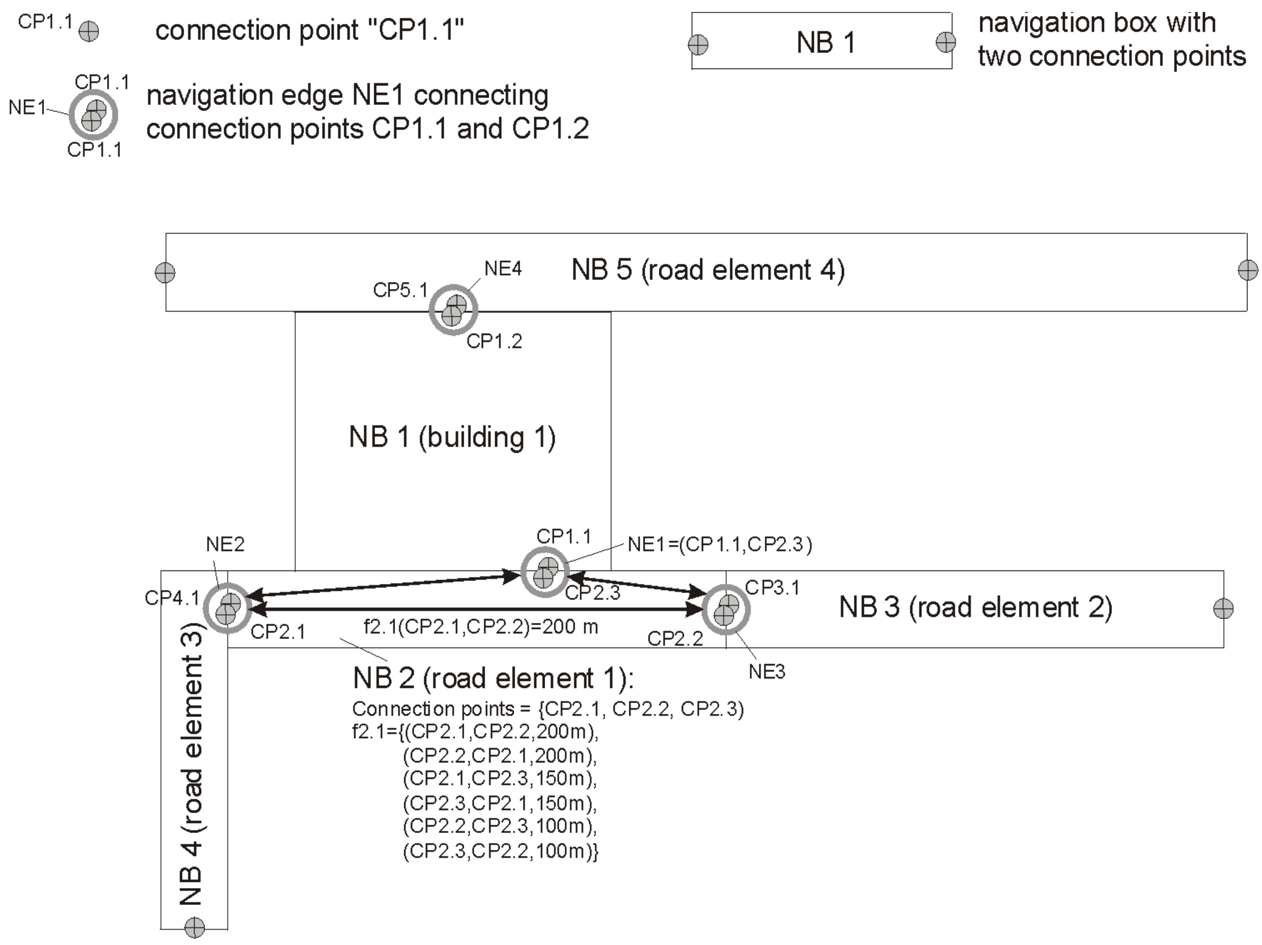


Figure 6: Navigation scenario using Navigation Boxes

Example 2 (refinement)

Figure 7 shows the refinement of the building represented by NB1. In this example, NB1 consists of two floors (NB1.1, NB1.2). The first floor NB1.1 has four connection points: two connection points represent the building's entrances on the first floor; CP1.1.3 and CP1.1.4 define connection points to floor 2 via stairs and an elevator.

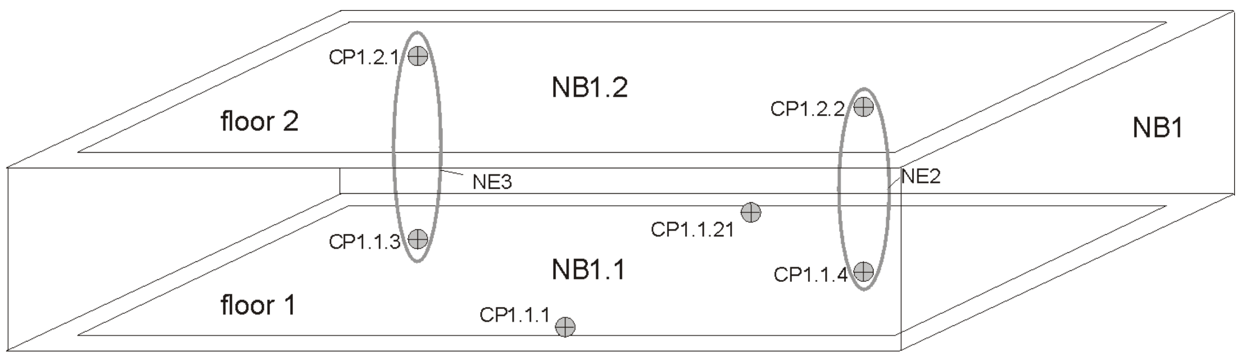


Figure 7: Navigation between different floors using the NB concept

Example 3 (indoor)

Figure 8 shows a typical indoor example. Typically, doors can be modeled as connection points. For each pair of a room's doors, the functions of the navigation box define the costs to pass through a room.

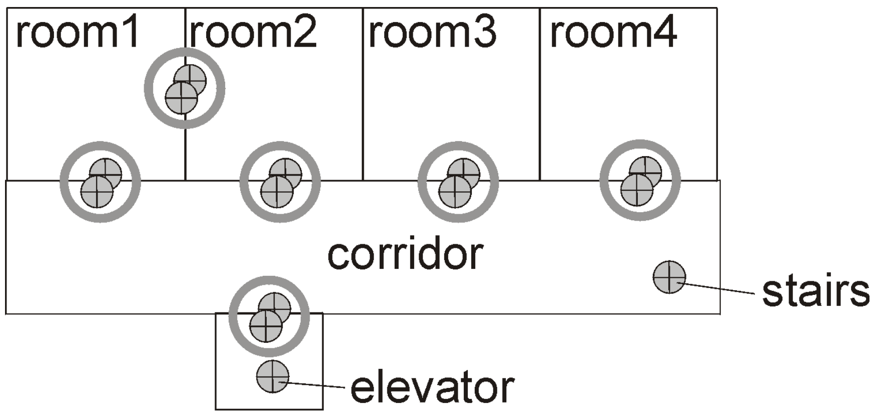


Figure 8: A typical example for indoor navigation

Example 4 (outdoor + restrictions)

The navigation concept of NEXUS does not follow the common approach of navigation systems, where NavigationNodes correspond to 0-dimensional features of the real world like street crossings and NavigationEdges represent linear features like roads or foot paths. Instead, the geographical entities of the AWS are mapped differently onto the NavigationalObjects of NEXUS (see also [VGH+02]). An example for outdoor navigation which also considers restrictions shall illustrate our approach. The example is depicted in Figure 9

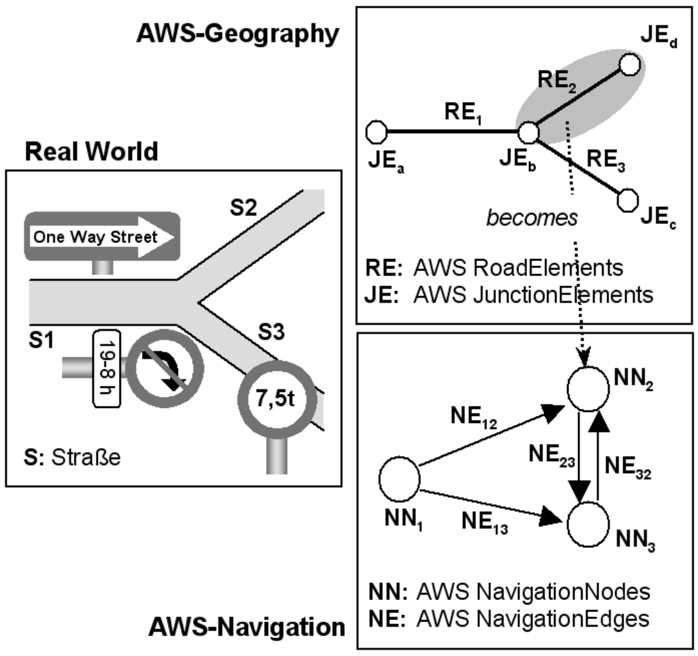


Figure 9: Geographical and topological representation of road data in NEXUS

A NavigationNode is made up of a From-JunctionElement, a RoadElement and a To-JunctionElement. The costs for traversing a line feature of the real world is thus an attribute of the NavigationNodes class according to the NEXUS topology concept. The connections of two NavigationNodes are referred to as NavigationEdges (NEs) as explained above.

In Figure 9, Street 1 (S1) is a one-way street. It is not allowed, to turn right from S1 into Street 3 (S3) from 7 p.m. until 8.a.m. Moreover, S3 can only be accessed by vehicles weighing up to 7.5 tons. The topological structure of the real world scene can directly be derived from the AWS geography. For example, NavigationNode NN_2 is composed of JunctionElement JE_b, RoadElement RE_2 and JunctionElement Je_d.

Basically, the Restrictions which apply to one single street are stored within the NavigationNodes, whereas Restrictions that affect a sequence of NavigationNodes are to be stored with NavigationEdge objects. In the example of Figure 9, the Restriction R_1, which tells us that Street S3 is closed for vehicles over 7.5 tons is stored with NavigationNode NN_3. The Restriction R_2 expressing that one is not allowed to turn right from S1 into S3 between 7 p.m. until 8 a.m., though, is stored with NavigationEdge NE_{13}. The topological situation of the example from Figure 9 can be displayed within a NavigationNode matrix (see Table 1).

*	NN_1	NN_2	NN_3, R_1
NN_1	-	NE_{12}	NE_{13}, R_2
NN_2	0	-	NE_{23}
NN_3, R_1	0	NE_{32}	-

Table 1: Matrix of NavigationNodes (NN), including Restrictions (R)

In the following example, the case of a forbidden sequence is illustrated. As it can be seen in the real world-sketch of Figure 10, it is not allowed to make a U-Turn from S1 into S2, but it is allowed to turn left from S1 into S3 and to turn left from S4 into S2. The real world situation is transferred in the AWS-Geography representation. On the basis of this representation, the NavigationalObjects are created.
Regarding the defined restriction, it is not allowed to go from NN_1 to NN_7 via NN_6. Thus, traversing the sequence of the NavigationEdges $\{NE_{16}, NE_{67}\}$ has to be explicitly forbidden. It is not possible to simply delete NE_{16} and NE_{67} since they are needed for the (allowed) sequences $\{NE_{16}, NE_{65}\}$ and $\{NE_{26}, NE_{67}\}$.

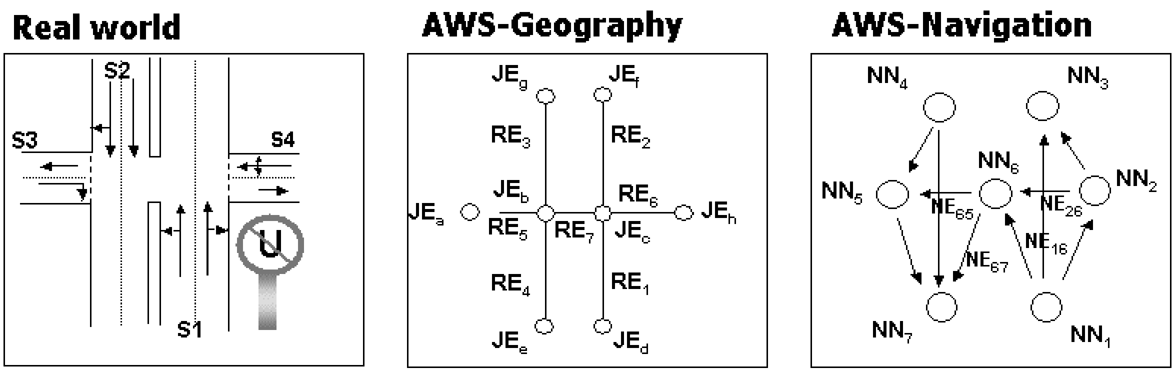


Figure 10: A forbidden sequence (U-turn example)

In order to create a navigation graph from road data available in GDF, first of all the geographical data stored within the GDF data have to be transferred into the NEXUS geography objects. Afterwards, the NavigationalObjects are built from the geography objects. In this step, GDF relations are transferred to Restrictions. On the basis of the NavigationalObjects a graph can be calculated applying the splitting node algorithm [Sc00]. On this graph the NEXUS Navigation will be performed (see 4.2.5).

4.2.3 Class Hierarchy

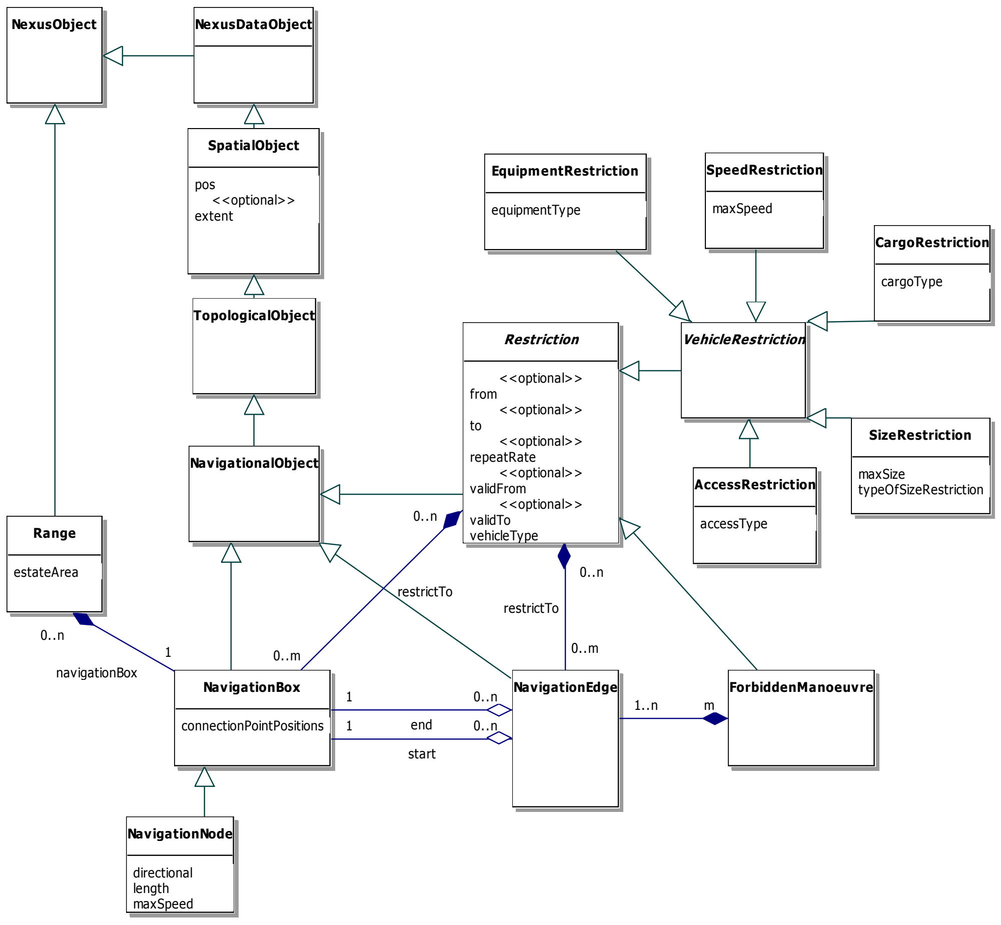


Figure 11: Class Hierarchy for NavigationalObjects

4.2.4 Explanation

NexusObject
Base class for all Nexus objects.

Range
A range represents a road section and the adjacent lots. Ranges can be used to navigate to and to search for addresses.

- NOL **roadElement**: references a road (element) object. The house number of this road element is given by the attribute *estateArea*.
- EstateArea **estateArea**: the estate area.

NexusDataObject
A data object.

- NOL **nol**: Nexus Object Locator (unique identifier of object)
- String **name**: name of the object.
- DataObjectType **kind**: kind of data object (**virtual** or **real** object).

SpatialObject
A spatial object.

- Position **pos**: geographic position of object.
- Extent **extent**: spatial extension of object.

TopologicalObject
Base class for topological objects.

NavigationalObject
Base class for navigational objects.

NavigationBox
- Position **pos**: geographic position of the center of the Navigation Box
- SetOfPositions **connectionPointPositions**: geographic positions of the connections points (entrances/exits) of the Navigation Box.
- SetOfBelongsTo **connectionPointBelongsTo**: reference to each geographical object the connection points belong to (e.g., entrances/exits of a building, road junctions, usw.).
- BelongsTo **endBelongsTo**: reference to the geographical object the second terminating object belongs to.
- BelongsTo **belongsTo**: reference to the geographical object the Navigation Box belongs to (e.g., a building, a city, a road element, …).

NavigationNode
A navigation node represents a road section including the two terminating objects (e.g. junctions, (bus) stops, etc.).
A navigation node is linked to its associated geographic objects by the attributes “startBelongsTo”, “endBelongsTo”, and “belongsTo”, so it is easily possible to retrieve geographic information for this topological object.
- A navigation node is also associated with geographic positions by the attributes “startPos”, “endPos”, and “pos”. This geographic information can be used for the retrieval of navigation nodes for a certain area (see Section5).
- Directional **directional**: shows, if this is a one-directional or a bi-directional road section.
- Length **length**: length of road section (m).
- Speed **maxSpeed**: maximum speed (m/s). Together with the attribute *length*, the cost of this node in the navigational graph can be calculated. Additional information for calculating the cost of the node can be given with the member *attributes* or by an extended class schema.
- SetOfRestrictions **restrictTo**: Restrictions applying to this navigation node.

Note: Additional standard class schema attributes for calculating the cost of a node can be defined in the future, if necessary.
Note: The *partOf* attribute can be used to refine this navigation node. For instance, a fine-grained city plan can be associated with a highway plan.

NavigationEdge
A navigation edge represents a directed connection of two Navigation Boxes, e.g., two road sections at the respective terminations of the sections (e.g., at a junction, or at a (bus) stop, etc.). It can be used to get from one section to the other.
- NOL **start**: reference to first navigation box
- NOL **end**: reference to second navigation box. Together with the attribute *start*, we get a directed connection (directed from “start” to “end”) of these two navigation boxes.
- Position **pos**: geographic position of the connection of the two navigation boxes (e.g., position of junction). This geographic information can be used for the retrieval of navigation edges for a certain area (see Section 5).
- SetOfRestrictions **restrictTo**: Restrictions applying to this navigation edge.

Restriction
Base class for restrictions. A restriction object represents a legal or physical restriction, e.g. a prohibited U-turn, or a road closed for heavy trucks. A concrete restriction is modeled by a sub-class (see below).

- Time **from** (optional): a restriction may only be valid for a certain period each day (e.g. from 9:00 to 17:00 o'clock). This attribute defines the begin of this period.
- Time **to** (optional): end of the daily period of validity.
- RepeatRate **repeatRate**: a restriction may only be valid for certain weekdays, e.g. only on working days or only at the weekend (see below).
- Date **validFrom** (optional): additionally to the regular daily period of validity, a restriction may only be valid once for a certain period of time (e.g. from October, 18th until November, 1st). This attribute defines the begin of this period.
- Date **validTo** (optional): end of period of validity.
- MobileObjectType **vehicleType** (optional): restrictions may only be relevant for certain types of mobile objects (e.g. a road closed only for trucks). This is the name of the corresponding mobile object class the restriction applies to (see section 3).

RepeatRate
This class defines possible repeat rates for restrictions. Possible values are: **daily**, **workingdays**, **nonworkingdays, monday, tuesday, wednesday, thursday, friday, saturday, sunday**

ForbiddenManoeuvre
Forbidden manoeuvres (e.g. a forbidden U-/left/right turn) are represented by a sequence of navigation edges. It is forbidden to traverse this navigation edges in the given order. For instance, if it is prohibited to go from road section *X* to road section *Y*, and then from road section *Y* to section *Z*, we get the forbidden sequence *S* = (NavigationNode(*X*,*Y*), NavigationNode(*Y*,*Z*)).

- SequenceOfNOL **forbiddenSequence**: reference to a sequence of navigation edges that are not allowed to be traversed in the given order.

VehicleRestriction
Base class for restrictions referring to a certain vehicle property.

EquipmentRestriction
This restriction refers to vehicles with (or better without) certain equipment. For instance, some roads may require snow chains in winter.

- Equipment **equipmentType**: required equipment (see above for a list of possible equipment).

CargoRestriction
This restriction refers to vehicles (especially trucks) that transport a certain type of cargo.

- CargoType **cargoType**: forbidden cargo type (see above for a list of cargo types).

SizeRestriction
This restriction refers to vehicles, which exceed a certain dimension.

- float **maxValue**: maximum allowed value (e.g. maximum width).
- TypeOfSizeRestriction **typeOfSizeRestriction**: type of restriction (see below).

TypeOfSizeRestriction
This class describes the different size restrictions. Possible values are: **width**, **height**, **length**, **weight**, **hubload**

AccessRestriction
Restrictions referring to a certain kind of traffic, e.g. road allowed for adjacent owners or agricultural vehicles only.

- TypeOfAccessRestriction **accessType**: type of access restriction (see below)

TypeOfAccessRestriction
This class describes the different access restrictions. Possible values are:
adjacent_owners_only, **agricultural_traffic_only**

SpeedRestriction
Restriction defining a maximum allowed speed.

- int **maxSpeed**: maximum allowed speed

4.2.5 Building Navigational Graphs

After all relevant navigational objects have been extracted from the model we need another step before we can do the navigation. To use standard navigation algorithms, like for instance the A* algorithm for shortest path routing, we first have to build up a graph from the topological information of the model. We cannot use the original graph formed by the navigation nodes and edges directly, because we have to take the various restrictions into account. That means, we first have to convert the original graph to a so-called navigation graph that contains no restrictions anymore.

Vehicle restrictions can be handled easily. We just have to check every navigation node and edge to see whether a vehicle restriction applies to this object (therefore, we need additional context information like a list of preferred means of transport, or the weight of the truck I am driving; this information is given in the query in NPL (see Section 4)). If it applies, the respective navigation node or edge is removed.

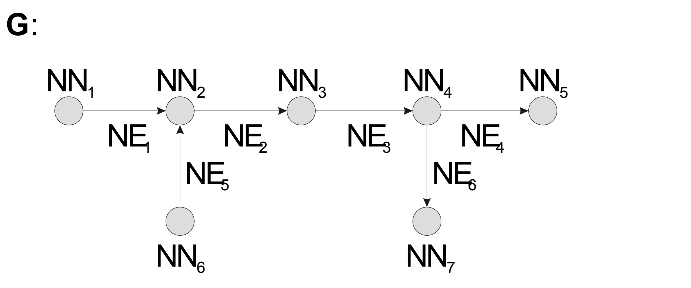
Forbidden sequence restrictions are more complex. We have to make sure, the navigation graph does not permit to traverse a forbidden sequence of navigation edges. A detailed description of how this can be done can be found in [Sc00]. In this paper, we just give a short overview of the principle.

The idea is to "break up" every forbidden sequence. Therefore, the first navigation edge of the forbidden sequence is removed. Hence, the sequence can not be traversed anymore, because the first edge is missing. But also every other path that used the removed navigation edge is now impossible, i.e. we have removed too much. To solve this problem, the forbidden path is duplicated without the last edge. Then, a "redirection" is built up that uses the duplicated nodes, and makes sure that all allowed sequences can be traversed again.

Figure 12 shows a simple example. Let G be the original graph with the forbidden sequence $S = (NE_1, NE_2, NE_3, NE_4)$. Now, we construct the navigation graph G' without the forbidden sequence.

First, we remove NE_1. Then, we duplicate the nodes NN_2, NN_3, and NN_4, and the edges NE_1, NE_2, and NE_3, i.e. the forbidden sequence without the last edge. The new nodes and edges are drawn in dark color in Figure 12 G'. Next, we have to connect the duplicated navigation nodes and the original graph. For instance, it must be possible to get from $NN_{4'}$ to NN_7. Therefore, we insert the edge $NE_{6'}$.

A formal description of the algorithm and a proof of correctness can be found in [Sc00].



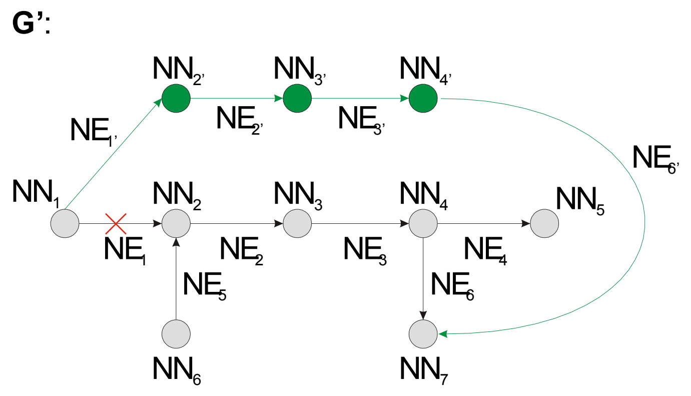


Figure 12: Construction of navigation graph
(G is the original graph with the forbidden sequence S = {NE_1, NE_2, NE_3, NE_4}; G' is the constructed navigation graph)

5 Retrieval and Management of Topological Information

As it has already been outlined we distinguish in our approach between geography and topology. Geography is the main index to access objects of our world model. Therefore, we first have to specify the area that is used in the query to retrieve all navigational objects needed for the navigation process. That means we need a heuristic to get the area the final route will go through. Then, this area can be used to query the federation for all relevant navigational objects (the determination of this area is an open issue and part of the future work).

We also have to associate geographic information with navigational objects (note: some Context Servers may store geographical objects *as well as* objects describing topological information; others may *exclusively* store geographic *or* topological information). To associate navigational objects with the respective geographical information, we can either use references to geographic objects or store geographic information at the navigational objects themselves. For the first alternative, we see two possible methods to retrieve all navigational objects for a certain region *R* by a request to the federation layer:

1. Retrieve all navigational objects from *all* Context Servers. The federation will use the references to geographic objects of these objects to retrieve the associated geographic objects from the Context Servers, and their geographic information is used in turn to sort out the navigation objects that are outside the region *R*. Note that this naïve approach may not be feasible because of the possibly huge set of navigational objects stored by all Context Servers.
2. Retrieve all geometrical objects for the region *R*. Search all navigation objects referring to these objects.

If we store geographic information directly at the navigational objects, we can use this information to retrieve all navigational objects for region *R*. We only need to query the Context Servers which are responsible for *R* and store navigational objects, because we can use the Augmented Area Service Register to identify all these [VB02] Context Servers. Surely, this procedure is the most efficient, so we decided to store geographic information at the navigational objects (navigation boxes, nodes and edges).

6 Summary, Open Issues and Future Work

6.1 Open Issues: Navigation Parameter/Result Language

6.1.1 Navigation Parameter Language

See section 2.1.
Defining positions in 3D (2D+Time).
Defining criteria for parts of the route.
Defining the level of detail of navigational graphs.
Defining locomotion types (and other values).
Mapping addresses and names to geographic coordinates or NOLs.

6.1.2 Navigation Result Language

See section 2.2.
Defining more result formats.

6.2 Future Work: Connecting Topological Graphs for Intermodal Navigation

Up to now, there is one topological graph for every locomotion type. In the future, we want to support intermodal navigation, so we have to connect these topological graphs. The result will be some layer architecture with one topological graph per locomotion type and per layer. The first open question is how to connect these graphs and the second open question is how the navigation application knows which locomotion type belongs to the actual topological graph.

7 References

[ATKIS98] Amtlich Topographisch-Kartographisches Informationssystem (ATKIS). Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland (AdV), Bonn, 1998.

[GDF95] Geographic Data Files Version 3.0, *GDF for Road Traffic and Transport Telematics*. CEN TC 287, 1995.

[InfEx04] Bauer et al.: *Information Management and Exchange in the Nexus Platform*. Bericht Nr. 2004/04, Fakultät Informatik, Universität Stuttgart, 2004.

[Nexus] The Nexus Project Homepage: http://www.nexus.uni-stuttgart.de/index.en.html

[NVML] Navigation Markup Language NVML; http://www.w3.org/TR/NVML.

[Sc00] Schmid, Wolfgang: *Berechnung kürzester Wege in Straßennetzen mit Wegeverboten.* Dissertation, Universität Stuttgart, 2000.

[VB02] Volz, S., Bofinger, J.-M.: *Integration of Spatial Data within a generic platform for location-based applications*. In: Proc. of the Joint International Symposium on Geospatial Theory, Processing and Applications, Ottawa 2002.

[VGH+02] Volz, S., Grossmann, M., Hönle, N., Nicklas, D., Schwarz, T.: *Integration mehrfach repräsentierter Straßenverkehrsdaten für eine föderierte Navigation*. In: it+ti, Informationstechnik und Technische Informatik, Oldenbourg, 5/2002, p. 260.

[Wa97] Walter, V.: *Zuordnung von raumbezogenen Daten – am Beispiel der Datenmodelle ATKIS und GDF*. Dissertation, Deutsche Geodätische Kommission (DGK), Reihe C, Heft Nr. 480, 1997.