INTERNATIONAL STANDARD

ISO 19133

First edition
2005-10-15

Geographic information — Location-based services — Tracking and navigation

Information géographique — Services basés sur la localisation — Suivi et navigation

Reference number
ISO 19133:2005(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Figures

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 19133 was prepared by Technical Committee ISO/TC 211, *Geographic information/Geomatics*.

Introduction

This International Standard is a description of the data and services needed to support tracking and navigation applications for mobile clients. The web services views of this International Standard are given in Annex C.

Geographic information — Location-based services — Tracking and navigation

1 Scope

This International Standard describes the data types, and operations associated with those types, for the implementation of tracking and navigation services. This International Standard is designed to specify web services that can be made available to wireless devices through web-resident proxy applications, but is not restricted to that environment.

2 Conformance

Conformance to this International Standard takes on two meanings dependent on the type of entity declaring conformance.

Mechanisms for the transfer of data are conformant to this International Standard if they can be considered to consist of transfer record or type definitions that implement or extend a consistent subset of the object types described within this International Standard.

Web servers for tracking and navigation are conformant to this International Standard if their interfaces implement one or more of the subtypes of service defined in this International Standard and their communications and messaging are accomplished using a conformant transfer mechanism.

Clauses 6 and 7 of this International Standard use the Unified Modeling Language (UML) to present conceptual schemas for describing the information and services for tracking and navigation. Clause 8 further describes a general schema for addresses to be used as location equivalents in three types of services. Clause 9 describes network data appropriate for these services. This International Standard concerns only externally visible interfaces and places no restriction on the underlying implementations other than what is needed to satisfy the interface specifications in the actual situation, such as

— interfaces to software services using techniques such as COM or CORBA;

— interfaces to databases using techniques such as SQL;

— data interchange using encoding as defined in ISO 19118.

Few applications will require the full range of capabilities described by this conceptual schema. This clause, therefore, defines a set of conformance classes that will support applications whose requirements range from the minimum necessary to define data structures to full object implementation. This flexibility is controlled by a set of UML types that can be implemented in a variety of manners. Implementations that define full object functionality shall implement all operations defined by the types of the chosen conformance class, as is common for UML designed object implementations. Implementations that choose to depend on external “free functions” for some or all operations, or forgo them altogether, need not support all operations, but shall always support a data type sufficient to record the state of each of the chosen UML types as defined by its member variables. Common names for “metaphorically identical” but technically different entities are acceptable. The UML model in this International Standard defines abstract types, application schemas define conceptual classes, various software systems define implementation classes or data structures, and the XML from the encoding standard (ISO 19118) defines entity tags. All of these reference the same information content. There is no difficulty in allowing the use of the same name to represent the same information content

even though at a deeper level there are significant technical differences in the digital entities being implemented.

Details of the conformance classes are given in the abstract test suite in Annex A.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 3166-1, *Codes for the representation of names of countries and their subdivisions — Part 1: Country codes*

ISO 19107, *Geographic information — Spatial schema*

ISO 19108, *Geographic information — Temporal schema*

ISO 19109, *Geographic information — Rules for application schema*

ISO 19111, *Geographic information — Spatial referencing by coordinates*

ISO 19112, *Geographic information —Spatial referencing by geographic identifiers*

ISO 19118, *Geographic information — Encoding*

4 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

4.1
candidate route
any **route** that satisfies all constraints of the routing request with the possible exception of optimality of the **cost function**

NOTE Navigation is the process of finding the candidate route that optimizes a chosen cost function.

4.2
cost function
function that associates a measure (cost) to a **route**

NOTE The normal mechanism is to apply a cost to each part of a **route**, and to define the total **route** cost as the sum of the cost of the parts. This is necessary for the operation of the most common navigation algorithms. The units of cost functions are not limited to monetary costs and values only, but include such measures as time, distance, and possibly others. The only requirement is that the function be additive and at least non-negative. This last criteria can be softened as long as no zero or less cost is associated with any loop in the network, as this will prevent the existence of a "minimal cost" **route**.

4.3
Dijkstra graph
positively weighted directed graph appropriately configured to execute a shortest path search

NOTE The term comes from the most commonly known algorithm for finding a shortest path in a positively weighted graph, from E. Dijkstra's paper [7]. Although this algorithm is not the only one in use, the requirements for the graph are common to most. The most common relaxation of the requirement is the "positive weights", which are not needed in the Bellman–Ford algorithm [4], [8].

4.4
geocoding
translation of one form of **location** into another

NOTE Geocoding usually refers to the translation of “address” or “intersection” to “direct position”. Many service providers also include a “reverse geocoding” interface to their geocoder, thus extending the definition of the service as a general translator of **location**. Because routing services use internal **location** encodings not usually available to others, a geocoder is an integral part of the internals of such a service.

4.5
instantiate
to represent (an abstraction) by the creation of a concrete instance or to create the ability to create an instance

NOTE A class or data element definition instantiates a type if it creates the ability to create objects or data elements, respectively, that can represent the concepts (instance data and/or operations) defined by that type. A class is instantiated by an object if the class defines that object’s structure and function. A data schema is instantiated by a data element if the data schema defines that element’s structure.

4.6
junction
single topological node in a **network** with its associated collection of **turns**, incoming and outgoing **links**

NOTE Junction is an alias for node.

4.7
linear referencing system
linear positioning system [ISO 19116**]**
positioning system that measures distance from a reference point along a route (feature)

NOTE The system includes the complete set of procedures for determining and retaining a record of specific points along a linear feature such as the **location** reference method(s) together with the procedures for storing, maintaining, and retrieving **location** information about points and segments on the highways. [NCHRP Synthesis 21, 1974]

4.8
link
directed topological connection between two nodes (**junctions**), consisting of an edge and a direction

NOTE Link is an alias for directed edge.

4.9
link position
position within a network on a **link** defined by some strictly monotonic measure associated with that **link**

NOTE Link positions are often associated with a target feature that is not part of the network. The most common link measures used for this are the distance from start node or address. The most common use of a link position is to geolocate an “address”.

4.10
location
identifiable geographic place

[ISO 19112]

NOTE A location is represented by one of a set of data types that describe a **position**, along with metadata about that data, including coordinates (from a coordinate reference system), a measure (from a **linear referencing system**), or an address (from an address system).

4.11
location-based service
LBS
service whose return or other property is dependent on the **location** of the client requesting the service or of some other thing, object or person

4.12
location-dependent service
LDS
service whose availability is dependent upon the **location** of the client

4.13
main-road rule
set of criteria used at a **turn** in lieu of a **route instruction**; default instruction used at a node

NOTE This rule represents what is "most natural" to do at a node (intersection), given the entry link used. The most common version is "as straight as possible", or to exit a **turn** on the most obvious extension of the entry street, which is usually, but not always, the same named street that was the entry. Every node in a route is either associated with an instruction or can be navigated by the main-road rule.

4.14
maneuver
manœuvre
collection of related **links** and **turns** used in a route in combination

NOTE Maneuvers are used to cluster **turns** into convenient and legal combinations. They may be as simple as a single **turn**, a combination of quick **turns** ("jogs" in the American mid-west consisting of a **turn** followed immediately by a **turn** in the opposite direction) or very complex combinations consisting of entry, exit, and connecting roadways ("magic roundabouts" in the UK).

4.15
navigation
combination of **routing**, **route transversal** and **tracking**

NOTE This is essentially the common term navigation, but the definition decomposes the process in terms used in the packages defined in this International Standard.

4.16
navigation constraint
constraint
restriction on how a **link** or **turn** may be traversed by a **vehicle**, such as **vehicle** classification, physical or temporal constraint

4.17
network
abstract structure consisting of a set of 0-dimensional objects called **junctions**, and a set of 1-dimensional objects called **links** that connect the **junctions**, each **link** being associated with a start (origin, source) **junction** and end (destination, sink) **junction**

NOTE The **network** is essentially the universe of discourse for the **navigation** problem. **Networks** are a variety of 1-dimensional topological complex. In this light, **junction** and topological node are synonyms, as are **link** and directed edge.

4.18
position
data type that describes a point or geometry potentially occupied by an object or person

NOTE A direct position is a semantic subtype of position. Direct positions as described can only define a point and therefore not all positions can be represented by a direct position. That is consistent with the "is type of" relation. An ISO 19107 geometry is also a position, just not a direct position.

4.19
route
sequence of **links** and/or partial **links** that describe a path, usually between two **positions**, within a **network**

4.20
route instruction
information needed at a point along a **route** in a **network** that allows that **route** to be traversed

NOTE To minimize the number of **instructions** needed to complete a **route traversal**, a default instruction can be assumed at **junctions** without specifically associated **instructions**. This default is called the **main-road rule**.

4.21
route traversal
process of following a **route**

4.22
routing
finding of optimal (minimal **cost function**) **routes** between **locations** in a **network**

4.23
slope
rate of change of elevation with respect to curve length

4.24
tracking
monitoring and reporting the **location** of a **vehicle**

4.25
traveller
person subject to being navigated or tracked

cf. vehicle

NOTE Includes pedestrians. See ISO 14825. In this International Standard, traveller can be replaced by **vehicle** without any change of intent.

4.26
traversable
condition of a **link** or **turn** that allows or restricts all traffic's traversal, as opposed to a more detailed **navigation constraint**

NOTE Traversability is usually a function of physical, cultural, or legal conditions. If traversable is false, then the object cannot be navigated. This effectively removes a **link** from the usable network. In the case of a node, it effectively removes the node and all associated **links** from the useable network. In the case of a **turn**, it simply removes it from any viable **route**. Non-traversable entities are not included in **maneuvers** or **routes**.

4.27
turn
part of a **route** or **network** consisting of a **junction** location and an entry and exit **link** for that **junction**

4.28
vehicle
object subject to being navigated or tracked

cf. traveller

NOTE Includes pedestrians. See ISO 14825. In this International Standard, vehicle can be replaced by **traveller** without any change of intent.

4.29
vehicle classification
type of **vehicle**, based on the nature of its construction or intended purpose

NOTE Classifications based on construction include automobile, truck, bus, bicycle, etc. Classifications based on purpose include taxi, emergency vehicle, etc. Vehicle classification can be used to determine the application of **navigation** constraints.

4.30
waypoint
location on the **network** that plays a role in choosing **candidate routes** potentially satisfying a routing request

5 Abbreviated terms and UML notation

5.1 Abbreviated terms

CRS Coordinate Reference System

CSL Conceptual Schema Language

ECCMA Electronic Commerce Code Management Association

GDF Geographic Data Files

GML Geography Markup Language

GPS Global Positioning System

IAEC International Address Element Code

LBS Location Based Service

LDS Location Dependent Service

LRM Linear Referencing Method

LRS Linear Reference System

OCL Object Constraint Language

PDA Personal Digital Assistant

UML Unified Modeling language

XML eXtensible Markup Language

5.2 UML notation

The UML notation used in this International Standard is described in ISO 19107, and differs from standard UML only in the existence and interpretation of some special stereotypes, in particular “CodeList” and “Union”.

The term “context diagram” used extensively in the naming of figures in this International Standard means a diagram that illustrates the context of a specified central type, meaning the types of it attributes, operations and association targets. This is the information most useful to the implementer of this central class.

6 Tracking

6.1 Semantics

The package “Tracking” contains other packages that are used in tracking services and related functions (see Figure 1).

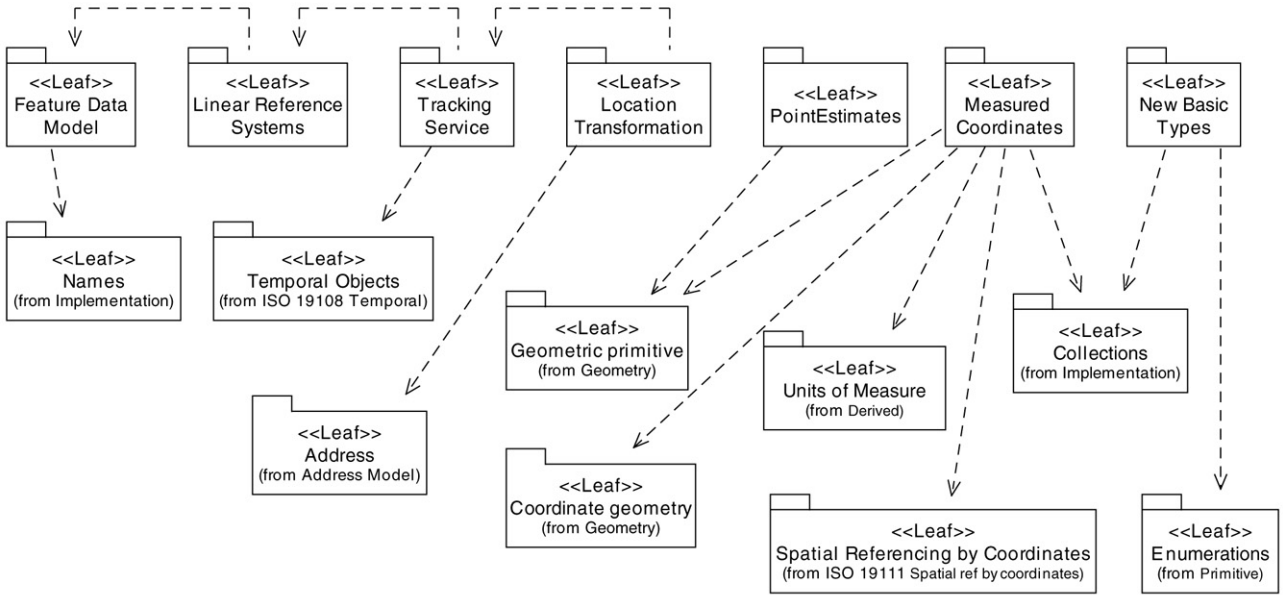

Figure 1 — Tracking packages

6.2 Package: Tracking Service

6.2.1 Semantics

The package “Tracking Service” contains types and classes useful in creating a tracking service. Since this is the core of many of the navigation functions described elsewhere in this International Standard, this package contains some important types universal to most if not all location-based services.

6.2.2 TK_Position

6.2.2.1 Semantics

The union class “TK_Position” is used to represent positions in tracking and associated applications. An instance of this class locates a position or place within the network. It should always be interpretable as a direct position (coordinate in a reference system), an address or as a network position. The discriminators for the union, and their associated types are as follows:

directPosition:	DirectPosition
placeName:	SI_LocationInstance
featureID:	FD_FeatureName
linearReference:	LR_PositionExpression
networkPosition:	NT_NetworkPosition
address:	AD_AbstractAddress
phone:	CharacterString

The operations on this type are all cast operators that allow the programmers to determine the position in the form most useful to them. The UML for TK_Position is given in Figure 2.

6.2.2.2 Operation: asPosition

The operator “asPosition” derives the coordinates in the CRS of the data of this location.

```
TK_Position :: asPosition() : DirectPosition
```

6.2.2.3 Operation: asNetworkPosition

The operator “asNetworkPosition” derives the network position of this location.

```
TK_Position :: asNetworkPosition() : NT_NetworkPosition
```

6.2.2.4 Operation: asAddress

The operator “asAddress” derives the address of this location.

```
TK_Position :: asAddress() : AD_Address
```

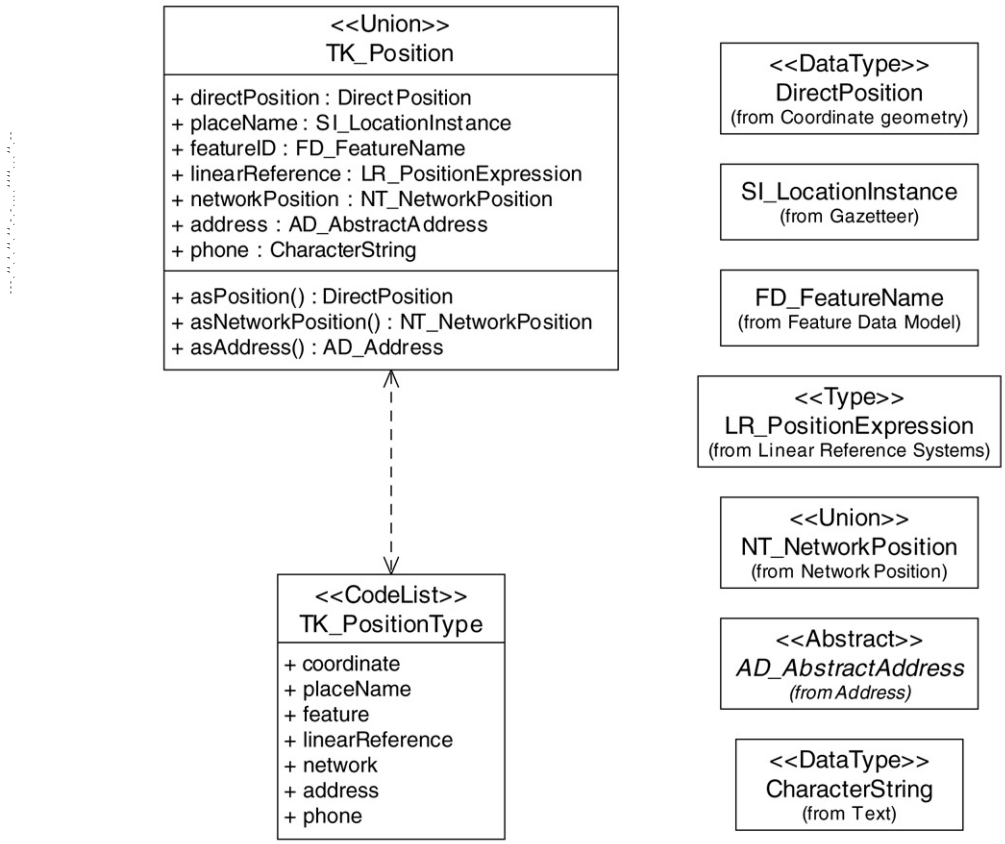

Figure 2 — Context Diagram: TK_Position

6.2.3 TK_MobileSubscriber

6.2.3.1 Semantics

The type “TK_MobileSubscriber” is used to model the items being tracked within the tracking service. The most common usage is the subscriber to a mobile service such as cell phone, where the service has the ability to determine the location of the device. The interfaces for the tracking services are also applicable to services that query the device for its location and would thus be applicable to GPS equipped devices as well. The service is a UML <<Type>> as opposed to a UML <<Interface>> because of the need to associate subscribers with their service.

There are deep issues of authentications between the service and the client, especially if the client is different from the mobile subscriber (in which case there are privacy issues). This International Standard assumes that those issues have been solved outside of the tracking interfaces (possible through a “verification” service, not dependent on geography, and therefore outside the scope of this International Standard). The UML for TK_MobileSubscriber is given in Figure 3.

6.2.3.2 Attribute: id : CharacterString

The attribute “id” is the identifier by which the associated tracking service knows this subscriber.

```
TK_MobileSubscriber :: id : CharacterString
```

6.2.3.3 Attribute: location : TK_TrackingLocation

The derived attribute “location” is the location of the subscriber at the time the attribute is accessed. This is essentially another access to the tracking service in many cases. In the case where the subscriber is a GPS equipped device, accessing this attribute would not necessarily activate the tracking service.

```
TK_MobileSubscriber :: location : TK_TrackingLocation
```

6.2.3.4 Role: trackingService[1..*] : TK_TrackingService

The association role “trackingService” links the subscribers to the tracking service that supports tracking them.

```
TK_MobileSubscriber :: trackingService[1..*]  : TK_TrackingService
```

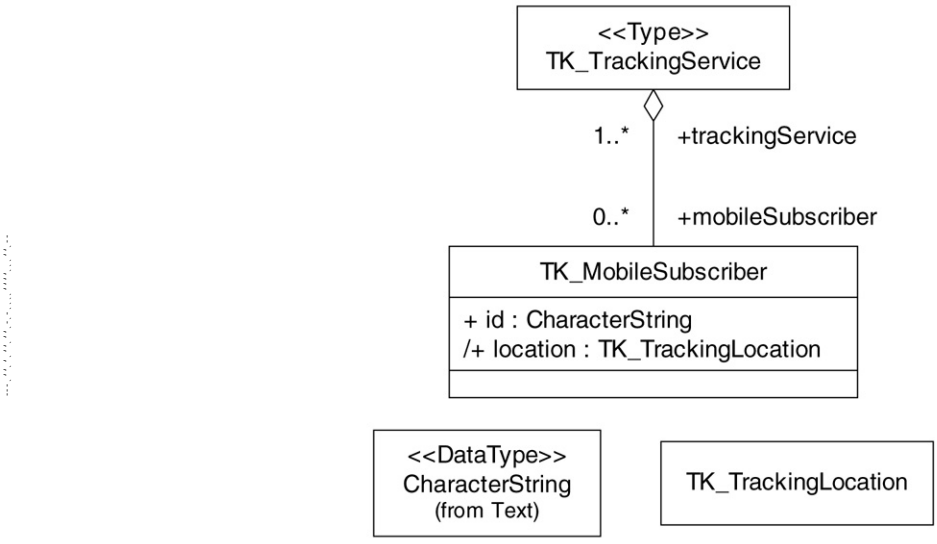

Figure 3 — Context Diagram: TK_MobileSubscriber

6.2.4 TK_TrackingLocation

6.2.4.1 Semantics

The type “TK_TrackingLocation” is used to represent objects describing a location and its associated metadata, such as time of measure, or direction of travel. The UML for TK_TrackingLocation is given in Figure 4.

6.2.4.2 Attribute: position : TK_Position

The attribute “position” describes the measurements that represent the position of this object.

```
TK_TrackingLocation :: position : TK_Position
```

6.2.4.3 Attribute: time[0..1] : TM_Primitive

The optional attribute “time” describes the time of the measurements that represent the position of this object.

```
TK_TrackingLocation :: time[0..1] : TM_Primitive
```

6.2.4.4 Attribute: direction[0..1] : Bearing

The attribute “direction” describes the direction of travel of the subscriber being tracked.

```
TK_TrackingLocation :: direction[0..1] : Bearing
```

6.2.4.5 Attribute: speed[0..1] : Velocity

The attribute “speed” describes the velocity of travel of the subscriber being tracked.

```
TK_TrackingLocation :: speed[0..1] : Velocity
```

6.2.4.6 Role: metadata[0..*] : TK_TrackingLocationMetadata

The association role “metadata” aggregates optional metadata elements associated with this location.

```
TK_TrackingLocation :: metadata[0..*] : TK_TrackingLocationMetadata
```

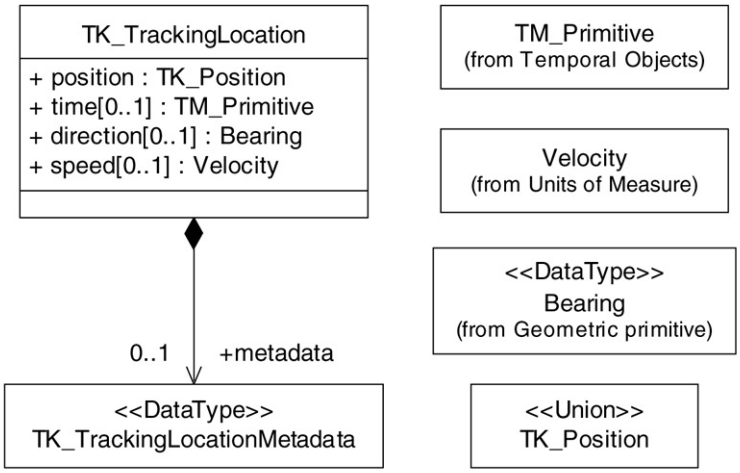

Figure 4 — Context Diagram: TK_TrackingLocation

6.2.5 TK_TrackingService

6.2.5.1 Semantics

The type “TK_TrackingService” defines interfaces and associations for tracking services. The UML for TK_TrackingService is given in Figure 5.

6.2.5.2 Role mobileSubscriber[0..*] : TK_MobileSubscriber

The association role “mobileSubscriber” aggregates items (vehicle or traveller) that may be tracked by using this service.

```
TK_TrackingService :: mobileSubscriber[0..*] : TK_MobileSubscriber
```

6.2.5.3 Operation: locate

The operation “locate” returns a single location for a tracked item.

```
TK_TrackingService ::
      locate(ms : TK_MobileSubscriber, type : TK_PositionType) :
      TK_TrackingLocation
```

6.2.5.4 Operation: track

The operation “track” returns a sequence of locations for a tracked item. The trigger types that cause a new location to be placed in the sequence are specified in the operation protocol.

```
TK_TrackingService ::
      track(ms : TK_MobileSubscriber, triggerType[1..*] : TK_Trigger) :
      TK_TrackingLocationSequence
```

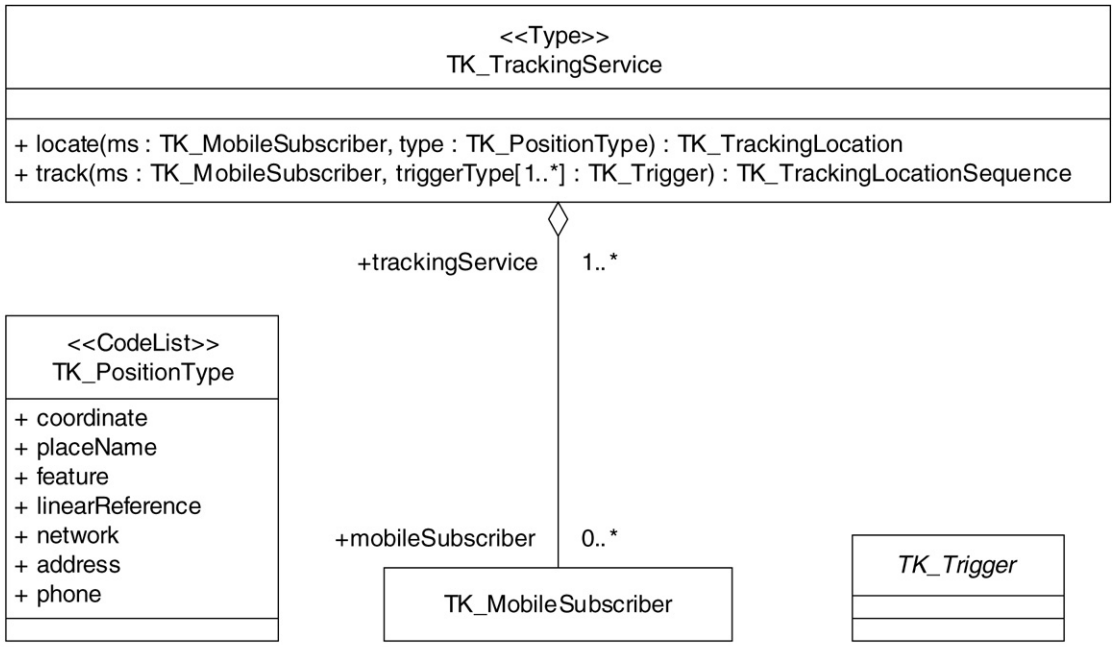

Figure 5 — Context Diagram: TK_TrackingService

6.2.6 TK_PositionType

The code list “TK_PositionType” lists the types of position available to the application. The initial members of this list include: “coordinate”, “placeName”, “feature”, “linearReference”, “network”, “address”, and “phone”. The UML for TK_PositionType is given in Figure 6.

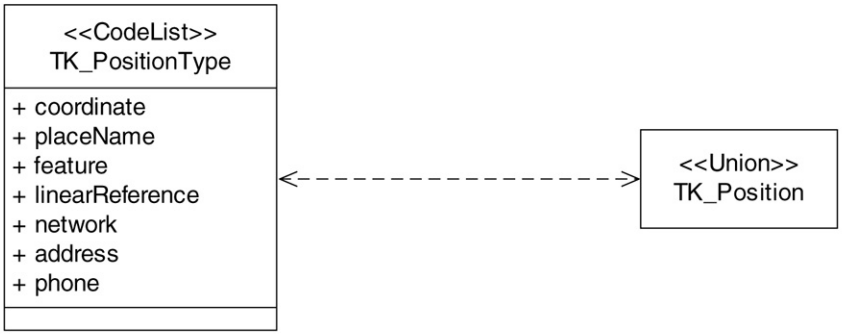

Figure 6 — Context Diagram: TK_PositionType

6.2.7 TK_TrackingLocationSequence

6.2.7.1 Semantics

The type “TK_TrackingLocationSequence” is an access mechanism for the return of a continuous tracking of an item. The UML for TK_TrackingLocationSequence is given in Figure 7.

6.2.7.2 Attribute: ms : TK_MobileSubscriber

The attribute “ms” gives the identity of the item being tracked.

```
TK_TrackingLocationSequence :: ms : TK_MobileSubscriber
```

6.2.7.3 Attribute: sequenceID : CharacterString

The attribute “sequenceID” gives the identity of the location sequence so that a client can repeatedly access the location sequence.

```
TK_TrackingLocationSequence :: sequenceID : CharacterString
```

6.2.7.4 Operation: next

The operation “next” returns the next location in the sequence.

```
TK_TrackingLocationSequence :: next() : TK_TrackingLocation
```

6.2.7.5 Operation: suspend

The operation “suspend” stops temporarily the stream of locations in the sequence.

```
TK_TrackingLocationSequence :: suspend() : Boolean
```

6.2.7.6 Operation: restart

The operation “restart” restarts the stream of locations in the sequence that has been stopped by “suspend”.

```
TK_TrackingLocationSequence :: restart() : TK_TrackingLocation
```

6.2.7.7 Operation: terminate

The operation “terminate” stops permanently the stream of locations in the sequence.

```
TK_TrackingLocationSequence :: terminate() : Boolean
```

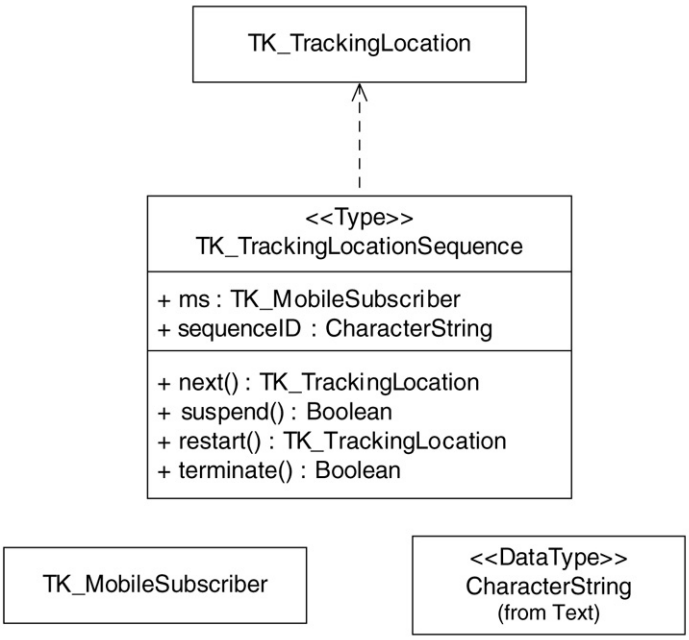

Figure 7 — Context Diagram: TK_TrackingLocationSequence

6.2.8 TK_Trigger

The abstract class “TK_Trigger” acts as a root class for trigger types to be used in controlling a location sequence. There are generally two types: triggered by an event, or triggered by the passage of time. The UML for TK_Trigger is given in Figure 8.

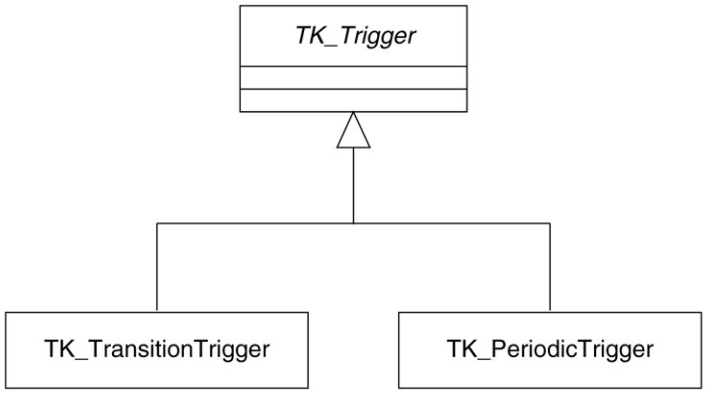

Figure 8 — Context Diagram: TK_Trigger

6.2.9 TK_PeriodicTrigger

6.2.9.1 Semantics

The class “TK_PeriodicTrigger” is used to control location sequences by setting up temporal limits on how far apart in time tracking samples are taken. Either minTime or maxTime shall be present for a valid TK_PeriodicTrigger. The UML for TK_PeriodicTrigger is given in Figure 9.

6.2.9.2 Attribute: minTime[0..1] : TM_Primitive

The attribute “minTime” is the minimum amount of time between tracking samples. In the absence of other criteria, the tracking service will begin the process to take a sample when the “minTime” has passed. Assuming no lag time is involved, the samples will be “minTime” apart.

```
TK_PeriodicTrigger :: minTime[0..1] : TM_Primitive
```

6.2.9.3 Attribute: maxTime[0..1] : TM_Primitive

The attribute “maxTime” is the maximum amount of time between tracking samples. In the absence of other criteria, the tracking service will attempt to assure that no two samples in the sequence are further apart than the “maxTime” parameter.

```
TK_PeriodicTrigger :: maxTime[0..1] : TM_Primitive
```

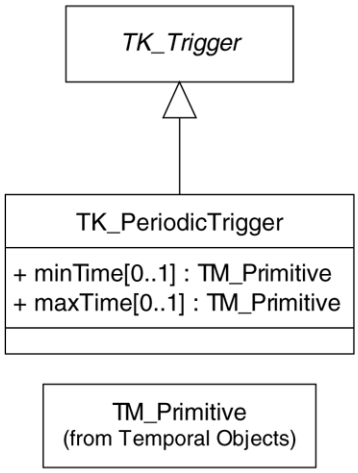

Figure 9 — Context Diagram: TK_PeriodicTrigger

6.2.10 TK_TransitionTrigger

6.2.10.1 Semantics

The class “TK_TransitionTrigger” models transitions relative to the item being tracked that will trigger the taking of a new location sample. In a TK_TransitionTrigger, at least one of the attributes shall be non-null. The UML for TK_TransitionTrigger is given in Figure10.

6.2.10.2 Attribute: type[0..*] : TK_TransitionType

The attribute “type” describes the type or types of transitions being watched. The various types of transitions are listed in the code list associated with the service.

```
TK_TransitionTrigger :: type[0..*] : TK_TransitionType
```

6.2.10.3 Attribute: deltaDirection[0..1] : Angle

The attribute “deltaDirection” describes the magnitude of the change of direction that triggers a new location sample.

```
TK_TransitionTrigger :: deltaDirection[0..1] : Angle
```

6.2.10.4 Attribute: deltaPosition[0..1] : Distance

The attribute “deltaPosition” describes the magnitude of the change of location (distance) that triggers a new location sample.

```
TK_TransitionTrigger :: deltaPosition[0..1] : Distance
```

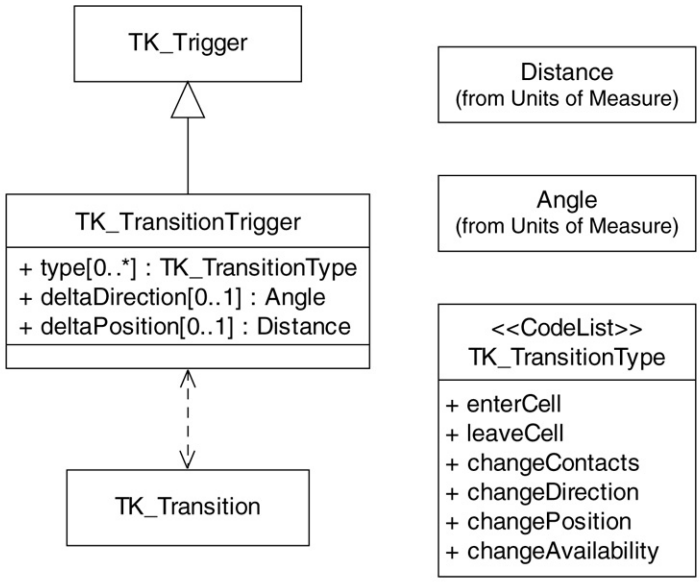

Figure 10 — Context Diagram: TK_TransitionTrigger

6.2.11 TK_TransitionType

The code list "TK_TransitionType" enumerates the types of transition tracked by this service. The initial values of the list include the following:

"enterCell"	mobile subscriber enters a cell associated with the network, e.g. cell phone network
"leaveCell"	mobile subscriber leaves a cell associated with the network
"changeContacts"	mobile subscriber changes the receivers with which it has contact
"changeDirection"	mobile subscriber changes the direction of its motion greater than the deltaDirection parameter specifies
"changePosition"	mobile subscriber changes its position greater than the deltaPosition parameter allows
"changeAvailability"	contact with the mobile subscriber is either lost or gained. If lost, the position just prior to loss is given. If gained, the first position calculated for the mobile subscriber after the new contact is given

6.2.12 TK_TrackingLocationMetadata

6.2.12.1 Semantics

The data type "TK_TrackingLocationMetadata" contains the metadata that is used to describe the position returned by the tracking service. As such, this data type shall always be contained in a TK_TrackingLocation, or other location type. The UML for TK_TrackingLocationMetadata is given in Figure 11.

6.2.12.2 Attribute: ms[0..1] : TK_MobileSubscriber

The attribute "ms" gives the identity of the item being tracked. If this attribute is not present, it should be derivable from context through the ownership of the containing TK_TrackingLocation.

```
TK_TrackingLocationSequence :: ms[0..1]: TK_MobileSubscriber
```

6.2.12.3 Attribute: quality : TK_QualityOfPosition

The attribute “quality” describes the quality of the position.

```
TK_TrackingLocationSequence :: quality : TK_QualityOfPosition
```

6.2.12.4 Attribute: time : TM_Primitive

The attribute “time” gives the time the position was measured.

```
TK_TrackingLocationSequence :: time : TM_Primitive
```

6.2.12.5 Attribute: clientID : CharacterString

The attribute “clientID” gives the identity of the client requesting the tracking service.

```
TK_TrackingLocationSequence :: clientID : CharacterString
```

6.2.12.6 Attribute: trigger[0..*] : TK_Transition

The attribute “trigger” gives the description of the transition that triggered this location measurement.

```
TK_TrackingLocationSequence :: trigger[0..*] : TK_Transition
```

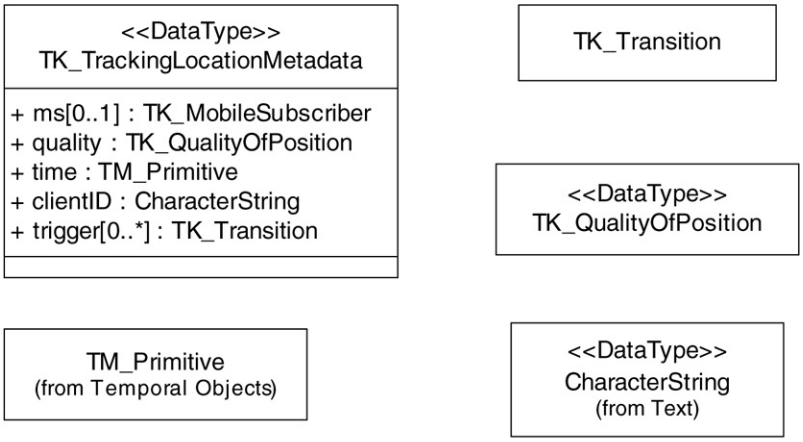

Figure 11 — Context Diagram: TK_TrackingLocationMetadata

6.2.13 TK_Transition

6.2.13.1 Semantics

The class “TK_Transition” describes a particular event or transition that triggered a location measure. The UML for TK_Transition is given in Figure 12.

6.2.13.2 Attribute: type[1..*] : TK_TransitionType

The attribute “type” describes the type or types of transitions.

```
TK_Transition :: type[1..*] : TK_TransitionType
```

6.2.13.3 Attribute: time : TM_Primitive

The attribute “time” describes the time that the transition occurred. Normally, this should be approximately the same time as in the location metadata.

```
TK_Transition :: time : TM_Primitive
```

6.2.13.4 Attribute: deltaDirection[0..1] : Angle

The attribute “deltaDirection” describes the magnitude of the change of direction that triggered this new location sample.

```
TK_Transition :: deltaDirection[0..1] : Angle
```

6.2.13.5 Attribute: deltaPosition[0..1] : Distance

The attribute “deltaPosition” describes the magnitude of the change of location (distance) that triggered this location sample.

```
TK_Transition :: deltaPosition[0..1] : Distance
```

6.2.13.6 Attribute: availability[0..1] : Boolean

The attribute “availability” describes whether the location of the item being tracked is currently available to be measured.

```
TK_Transition :: availability[0..1] : Boolean
```

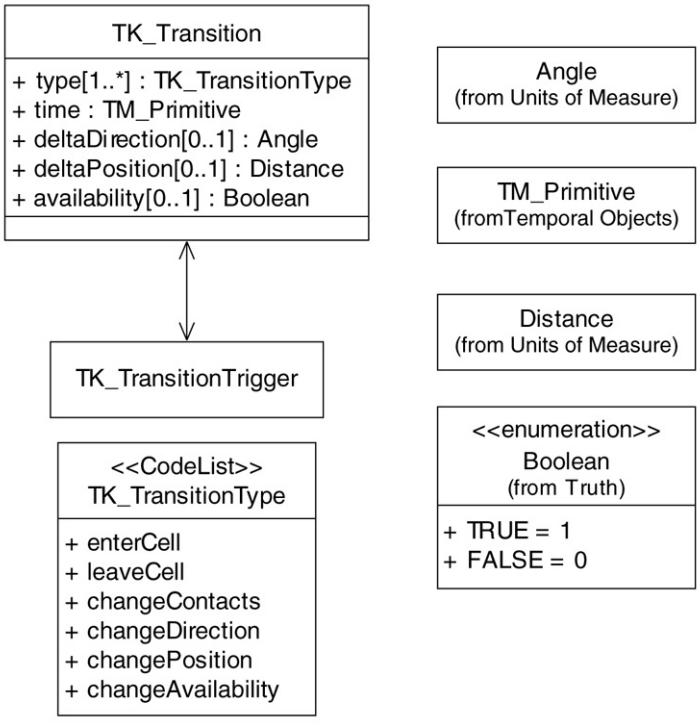

Figure 12 — Context Diagram: TK_Transition

6.2.14 TK_QualityOfPosition

6.2.14.1 Semantics

The data type “TK_QualityOfPosition” is used to describe the quality of position measurements. The UML for TK_QualityOfPosition is given in Figure 13.

6.2.14.2 Role: accuracyStatement[1..*] : TK_AccuracyStatement

The association role “accuracyStatement” aggregates statements about the accuracy of the position.

```
TK_QualityOfPosition :: accuracyStatement[1..*] : TK_AccuracyStatement
```

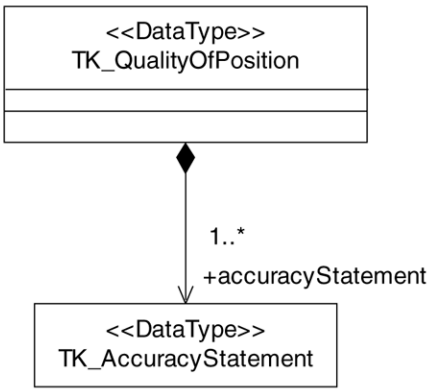

Figure 13 — Context Diagram: TK_QualityOfPosition

6.2.15 TK_Accuracy

6.2.15.1 Semantics

The class “TK_Accuracy” is used to describe accuracy of a measure. The numerical values are in the same units as the measure. The UML for TK_Accuracy is given in Figure 14.

6.2.15.2 Attribute: accuracyType : TK_AccuracyType

The attribute “accuracyType” gives the type of accuracy measure.

```
TK_Accuracy :: accuracyType : TK_AccuracyType
```

6.2.15.3 Attribute: sigmaLevel[0..1] : Number

The attribute “sigmaLevel” describes the accuracy of the measure based on a multiple of the standard deviation.

```
TK_Accuracy :: sigmaLevel[0..1] : Number
```

6.2.15.4 Attribute: sigma[0..1] : Number

The attribute “sigma” describes the standard deviation.

```
TK_Accuracy :: sigma[0..1] : Number
```

6.2.15.5 Attribute: units[0..1]: UnitOfMeasure

The attribute “units” gives the unit of measure for the accuracy for the measurement as needed.

```
TK_AccuracyStatement :: unit[0..1] : UnitOfMeasure
```

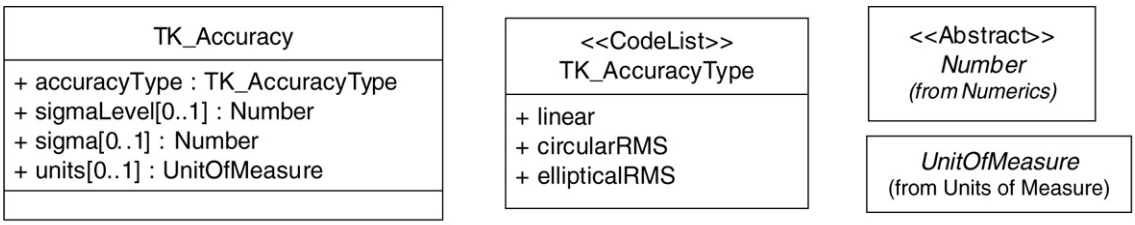

Figure 14 — Context Diagram: TK_Accuracy

6.2.16 TK_AccuracyType

The code list “TK_AccuracyType” lists the type of accuracy measure used by this service. The initial values in the value domain are “linear”, “circularRMS”, and “ellipticalRMS”.

6.2.17 TK_AccuracyStatement

6.2.17.1 Semantics

The data type “TK_AccuracyStatement” is used to make statements about metric accuracy of measurements. The UML for TK_AccuracyStatement is given in Figure 15.

6.2.17.2 Attribute: axis[1..*] : CharacterString

The attribute “axis” gives the axis or axes about which the statement applies. This name shall correspond to axes in a given coordinate reference system for the geometry of the given position.

```
TK_AccuracyStatement :: axis[1..*] : CharacterString
```

6.2.17.3 Attribute: accuracy : TK_Accuracy

The attribute “accuracy” gives accuracy for a measurement.

```
TK_AccuracyStatement :: accuracy : TK_Accuracy
```

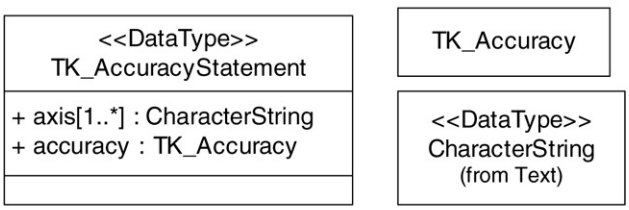

Figure 15 — Context Diagram: TK_AccuracyStatement

6.3 Package: Point Estimates

6.3.1 Semantics

The package “Point Estimates” contains geometry object types, all subclasses of “GM_Point", used in describing direct position estimates seen in tracking services. The UML for the inheritance hierarchy for the point estimate classes is given in Figure 16. A graphic of their form, showing the estimate with its error geometry, is given in Figure 17.

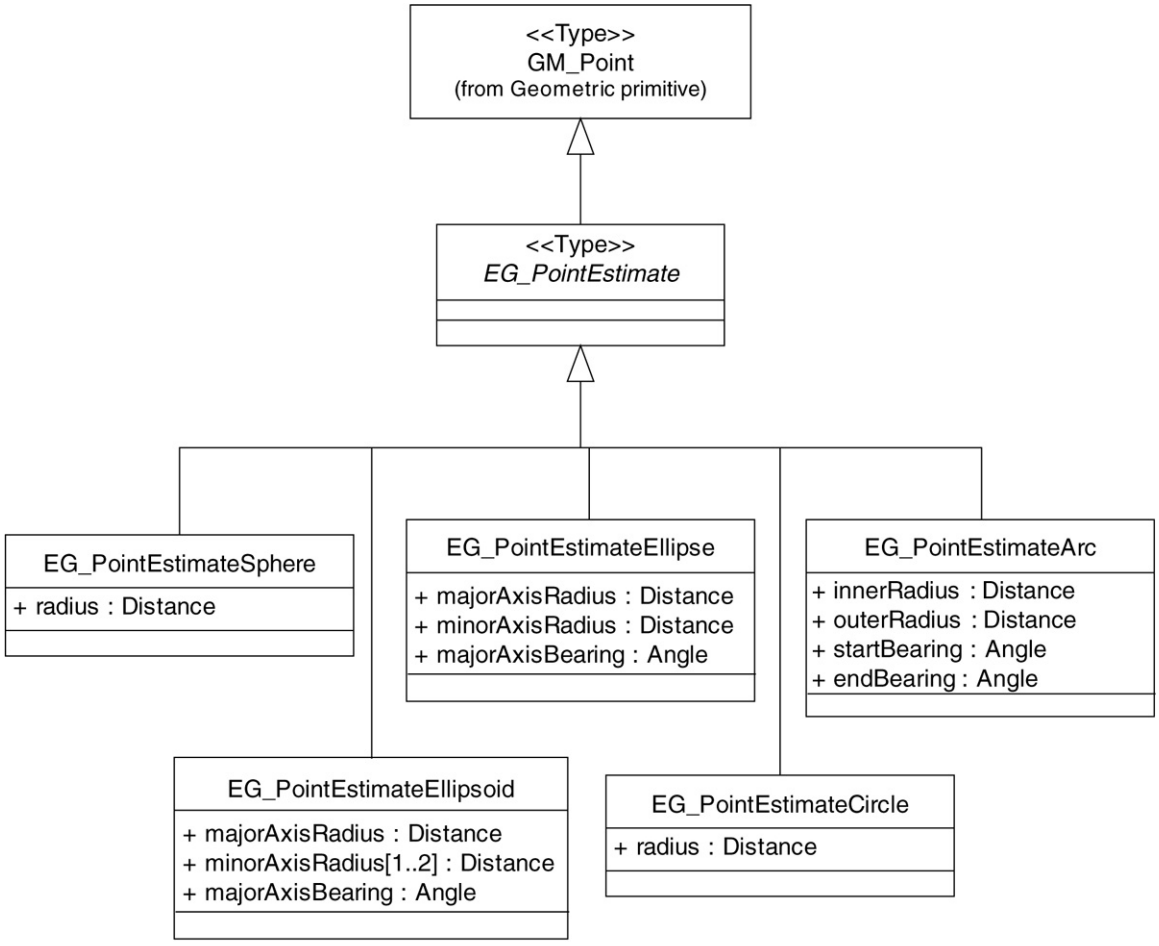

Figure 16 — Point Estimate classes

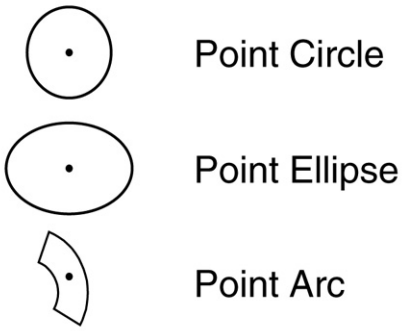

Figure 17 — Geometric interpretations of point estimate types

6.3.2 EG_PointEstimate

The type “EG_PointEstimate” is the abstract root of all the error estimate geometry objects. It is a subtype of point. The best estimate of the location is the centre of the given geometry. The coordinates of the point specify the primary constructive location of the geometry, and will be the best estimate of the location if and only if it is also the centre of the geometry specified. The UML for EG_PointEstimate is given in Figure 16.

6.3.3 EG_PointEstimateCircle

6.3.3.1 Semantics

The class “EG_PointEstimateCircle” is used to describe a point with an error estimate that is a metric circle of a given radius. It should be noted that this circle is not a circle in the coordinate reference system of the point unless the reference system is equivalent a local engineering one. The point coordinates (inherited from GM_Point) specify the centre of the circle, and the best estimate of position. The UML for EG_PointEstimateCircle is given in Figure 18.

6.3.3.2 Attribute: radius : Distance

The attribute “radius” is the radius of the error circle used in the point estimate. The usual semantics is to return a fixed “sigma” level that is a fixed multiple of the standard deviation of the measurement procedure. Unless otherwise specified by a profile, a 1,0-sigma level is given.

```
EG_PointEstimateCircle :: radius : Distance
```

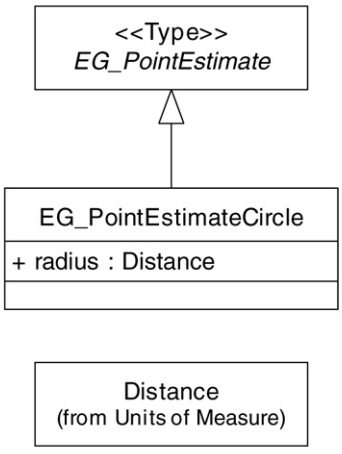

Figure 18 — Context Diagram: EG_PointEstimateCircle

6.3.4 EG_PointEstimateEllipse

6.3.4.1 Semantics

The point ellipse is the same as a point circle except that an ellipse defined by the parameters given specifies the error. The point coordinates (inherited from GM_Point) specify the centre of the ellipse, and the best estimate of position. The UML for EG_PointEstimateEllipse is given in Figure 19.

6.3.4.2 Attribute: majorAxisRadius : Distance

The attribute “majorAxisRadius” is half the length of the major axis of the ellipse.

```
EG_PointEstimateEllipse :: majorAxisRadius : Distance
```

6.3.4.3 Attribute: minorAxisRadius : Distance

The attribute “minorAxisRadius” is half the length of the minor axis of the ellipse.

```
EG_PointEstimateEllipse :: minorAxisRadius: Distance
```

6.3.4.4 Attribute: majorAxisBearing : Angle

The attribute “majorAxisBearing” is bearing of the major axis of the ellipse, given as an angle from true north. The minor axis is perpendicular to the major axis.

```
EG_PointEstimateEllipse :: majorAxisBearing : Angle
```

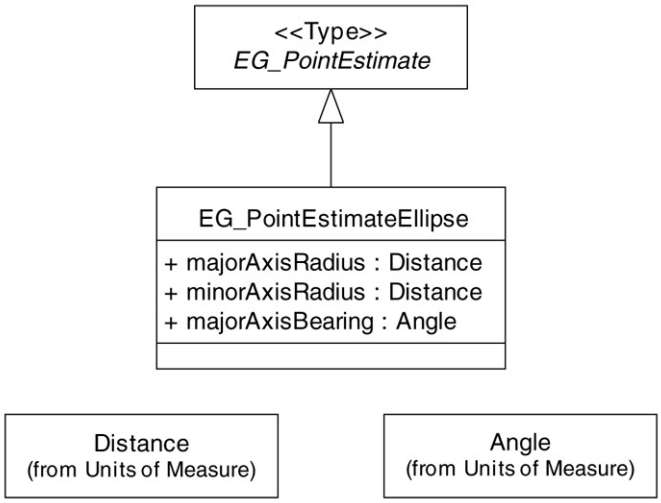

Figure 19 — Context Diagram: EG_PointEstimateEllipse

6.3.5 EG_PointEstimateArc

6.3.5.1 Semantics

The point arc is the same as a point circle except that a pair of arcs between two given radii and two given bearing angles gives the area estimate error. The point coordinates specify the centre of the two arcs. The best estimate of position is the along the centreline of the two arcs and halfway between the two radii. The UML for EG_PointEstimateArc is given in Figure 20.

6.3.5.2 Attribute: innerRadius : Distance

The attribute “innerRadius” is the radius of the smaller of the two arcs.

```
EG_PointEstimateArc :: innerRadius : Distance
```

6.3.5.3 Attribute: outerRadius : Distance

The attribute “outerRadius” is the radius of the larger of the two arcs.

```
EG_PointEstimateArc :: outerRadius: Distance
```

6.3.5.4 Attribute: startBearing : Angle

The attribute “startBearing” is the bearing of one side of the arcs (left-hand end when viewed from the arc centre).

```
EG_PointEstimateArc :: startBearing : Angle
```

6.3.5.5 Attribute: endBearing : Angle

The attribute “endBearing” is the bearing of the other side of the arcs (right-hand end when viewed from the arc centre).

```
EG_PointEstimateArc :: endBearing: Angle
```

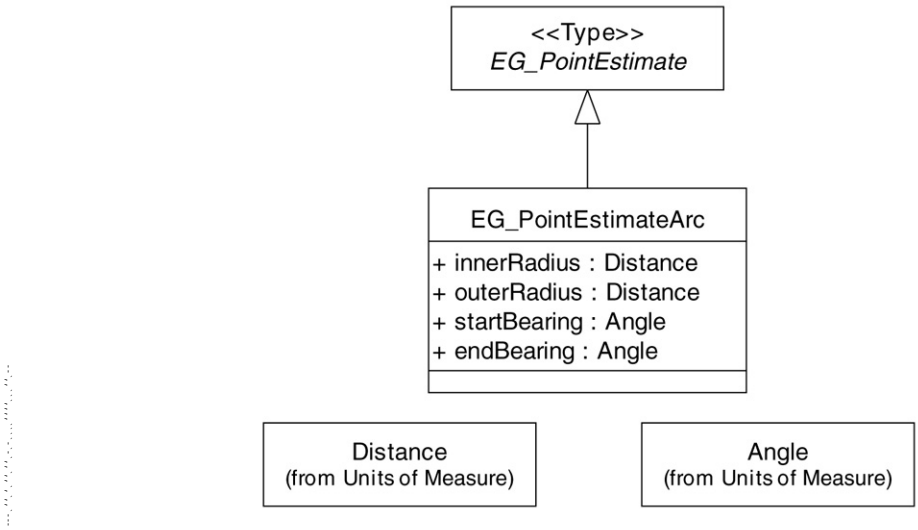

Figure 20 — Context Diagram: EG_PointEstimateArc

6.3.6 EG_PointEstimateSphere

6.3.6.1 Semantics

The class “EG_PointEstimateSphere” is used to describe a point with an error estimate that is a metric sphere of a given radius. It should be noted that this sphere is not a sphere in the coordinate reference system of the point unless the reference system is equivalent to a local engineering one. The point coordinates (inherited from GM_Point) specify the centre of the sphere, and the best estimate of position. The UML for EG_PointEstimateSphere is given in Figure 21.

6.3.6.2 Attribute: radius : Distance

The attribute “radius” is the radius of the error sphere used in the point estimate. The usual semantics is to return a fixed “sigma” level that is a fixed multiple of the standard deviation of the measurement procedure. Unless otherwise specified by a profile, a 1,0-sigma level is given.

```
EG_PointEstimateSphere :: radius : Distance
```

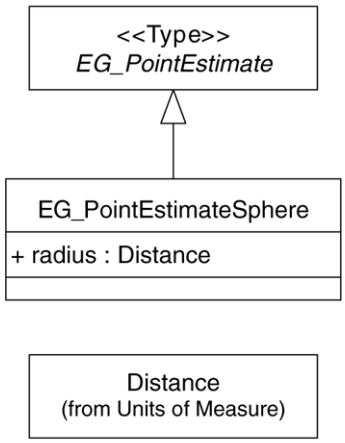

Figure 21 — Context Diagram: EG_PointEstimateSphere

6.3.7 EG_PointEstimateEllipsoid

6.3.7.1 Semantics

The point ellipsoid is the same as a point sphere except that an ellipsoid defined by the parameters given specifies the error. The point coordinates (inherited from GM_Point) specify the centre of the ellipsoid, and the best estimate of position. The UML for EG_PointEstimateEllipsoid is given in Figure 22.

6.3.7.2 Attribute: majorAxisRadius : Distance

The attribute “majorAxisRadius” is half the length of the major axis of the ellipsoid. In this case, major means “primary” or “central”. It may not necessarily be the longest of the three axes. It is the axis of rotation in the case where the remaining axes are equal.

```
EG_PointEstimateEllipsoid :: majorAxisRadius : Distance
```

6.3.7.3 Attribute: minorAxisRadius : Distance

The attribute “minorAxisRadius” is half the length of the minor axis of the ellipsoid. If one axis radius is given, the ellipsoid is a figure of rotation about the major axis, with this radius at the central “equator”. If two axes are given, then the figure is a fully general tri-axis ellipsoid.

```
EG_PointEstimateEllipsoid :: minorAxisRadius: Distance
```

6.3.7.4 Attribute: majorAxisBearing : Angle

The attribute “majorAxisBearing” is bearing of the major axis of the ellipsoid, given as an angle from true north. The minor axis is perpendicular to the major axis. If a second angle is given, this is an elevation angle from the local horizontal.

```
EG_PointEstimateEllipsoid :: majorAxisBearing[1..2] : Angle
```

6.3.7.5 Attribute: minorAxisBearing : Angle

The attribute “minorAxisBearing” is bearing of the minor axis of the ellipsoid, given as an angle from true north. The minor axis is perpendicular to the major axis. If no angle is given, then the minor axis is in the horizontal plane, perpendicular to the major axis. If a second angle is given, this is an elevation angle from the local horizontal. The third axis (second minor axis, if needed) is perpendicular to both other axes and therefore its direction is fixed by the values already given. Its unit vector direction in 3D is the cross product of the unit vector directions of the other two axes.

```
EG_PointEstimateEllipsoid :: minorAxisBearing[0..2] : Angle
```

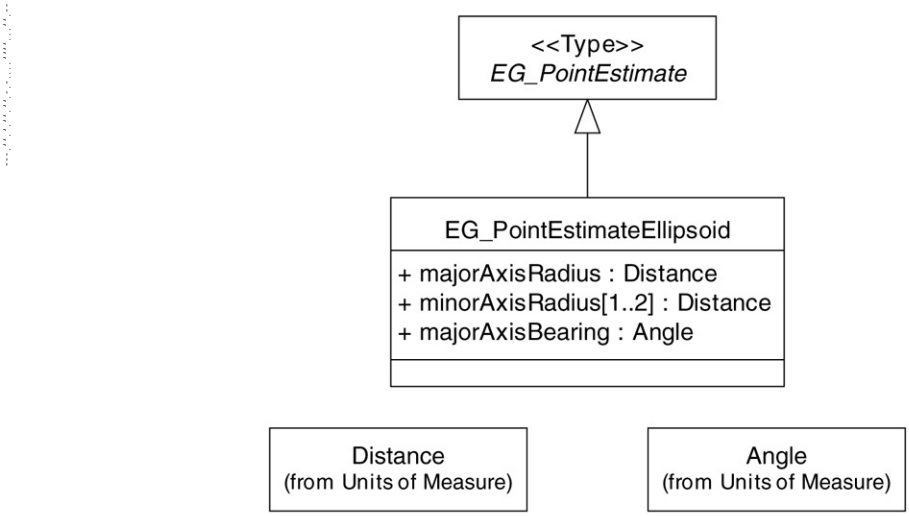

Figure 22 — Context Diagram: EG_PointEstimateEllipsoid

6.4 Package: Location Transformation

6.4.1 Semantics

The package “Location Transformation” contains classes to support the transformation of the various forms of location into one another.

6.4.2 LT_LocationTransformationService

6.4.2.1 Semantics

The abstract type “LT_LocationTransformationService” provides interfaces for services based upon the transformation of one form of location into another. The most common example of such a service is a geocoder, which maps addresses to geographic locations and vice versa. The UML for LT_LocationTransformationService is given in Figure 23.

6.4.2.2 Operation: cast

The operation “cast” takes one form of location and creates another form of the same location.

```
LT_LocationTransformationService ::
      cast(p : TK_TrackingLocation, asType : TK_PositionType) :
      TK_TrackingLocation
```

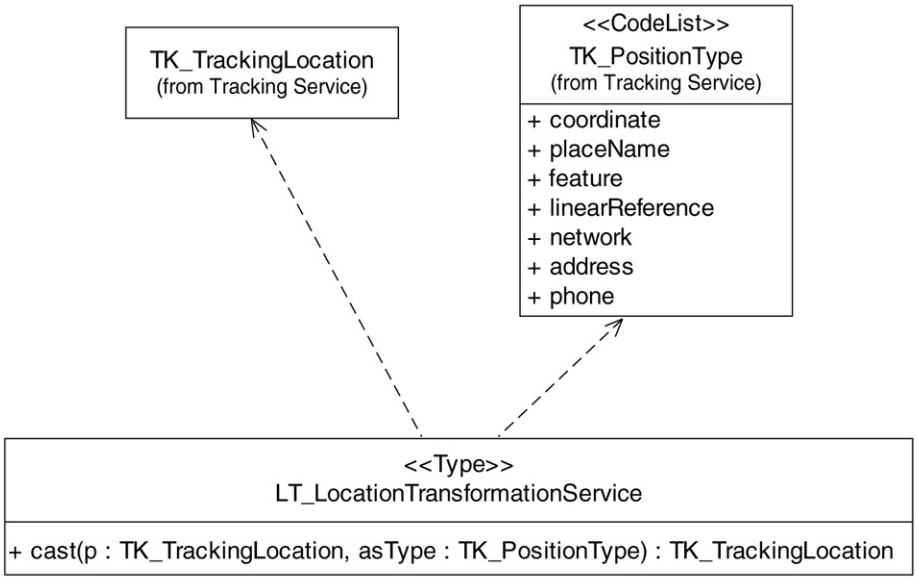

Figure 23 — Context Diagram: LT_LocationTransformationService

6.5 Package: Measured Coordinates

6.5.1 Semantics

The package “Measured Coordinates” contains classes for associating measures to coordinate positions. This is most commonly done in the creation of linear reference systems. This is done by modifying the coordinate reference systems defined in ISO 19111, by adding extra coordinate axes to the systems that allow for the storage of dependent coordinate measures, such as length measures.

This concept is illustrated in Figure 24. This illustrates how direct positions, the foundation of coordinate geometries in ISO 19107, are modified to carry measure information. Figure 25 shows the mechanism for altering a coordinate reference system to allow for the additional axes to be added allowing for the measure. Normally, only one measure is added (usually a nominal arc distance), and other measures are given as functions of this basic one.

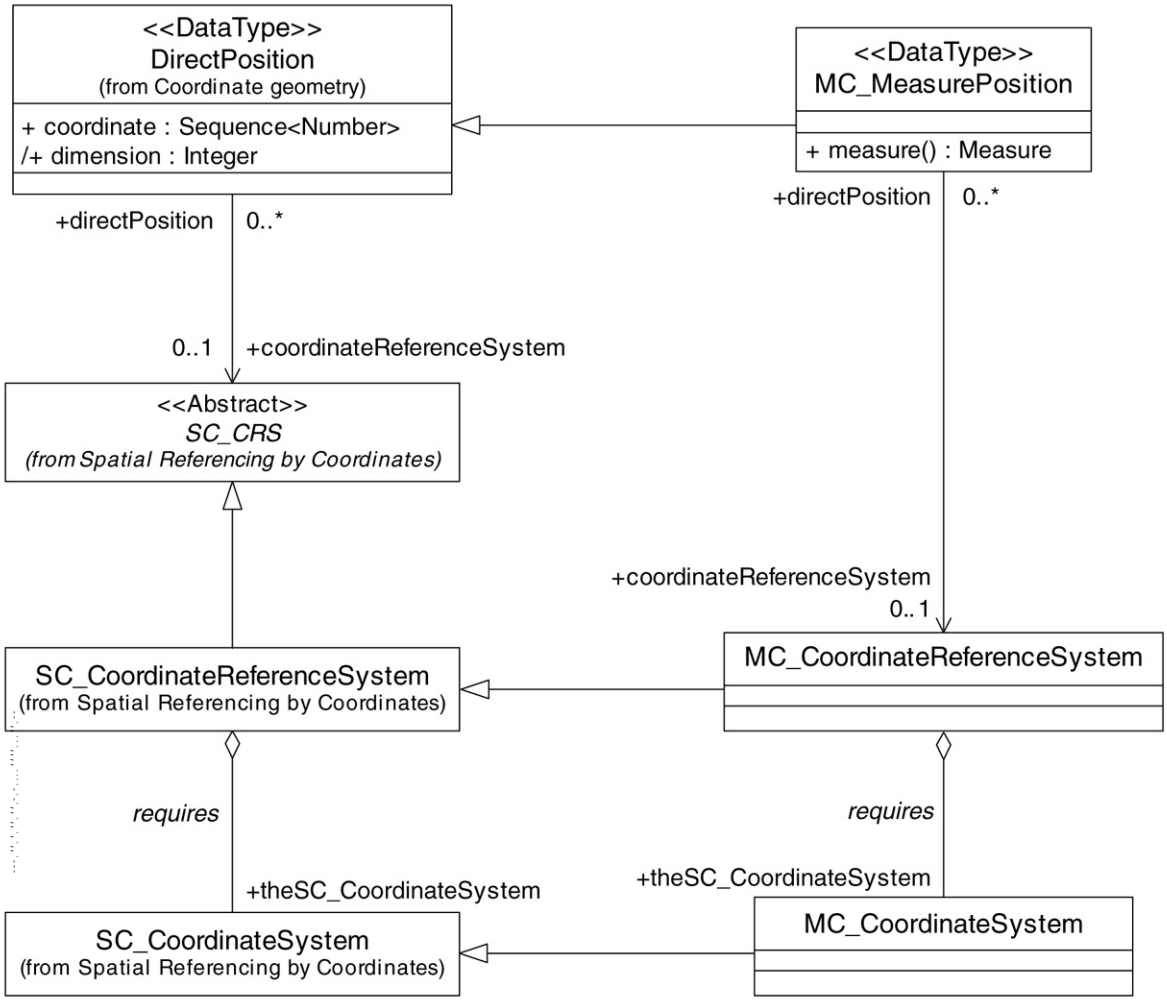

Figure 24 — Measure Position

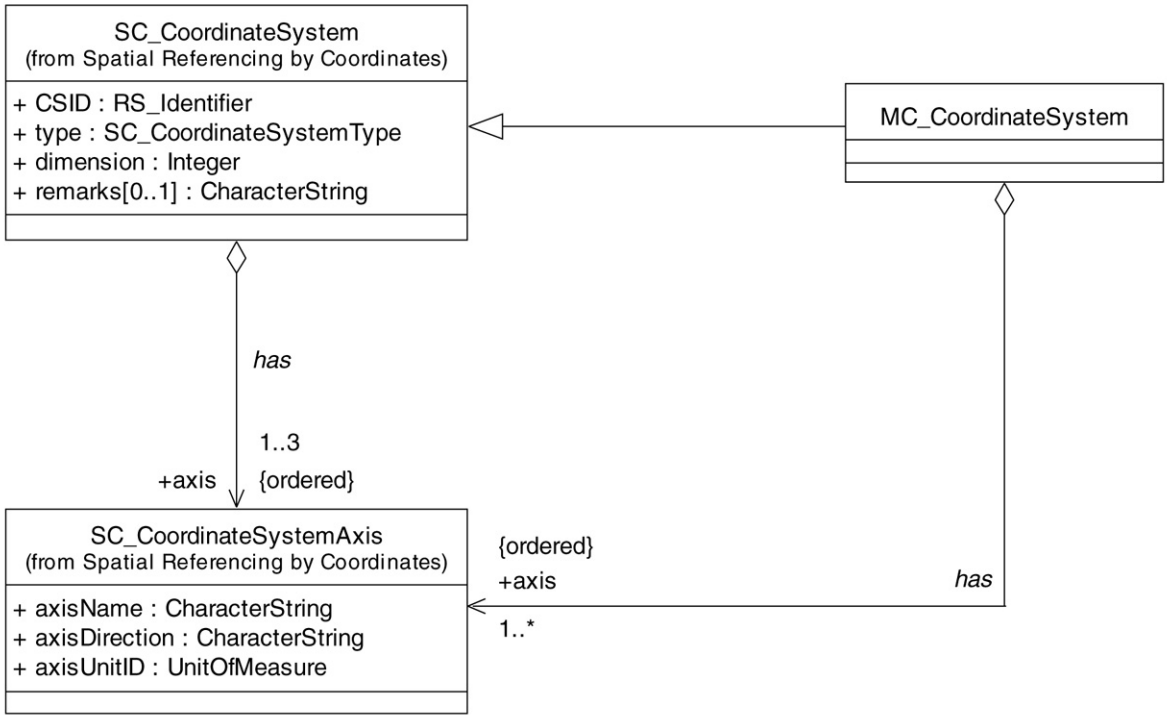

Figure 25 — Measured Coordinate Systems

6.5.2 MC_MeasurePosition

6.5.2.1 Semantics

The data type “MC_MeasurePosition” extends the concept of a DirectPosition to include additional measures as ordinates. The UML for MC_MeasurePosition is given in Figure 26.

6.5.2.2 Role: coordinateReferenceSystem[0..1] : MC_CoordinateReferenceSystem

The optional association role “CoordinateReferenceSystem” inherited from DirectPosition, will now reference a measured coordinate system. If this role is empty, then, as in the case of direct position, there shall be a larger context that implies the coordinate reference system.

```
MC_MeasurePosition :: CoordinateReferenceSystem[0..1] :
      MC_CoordinateReferenceSystem
```

6.5.2.3 Operation: measure

The operation “measure” gives the measured position of a projection onto its measure component.

```
MC_MeasurePosition :: measure() : Measure
```

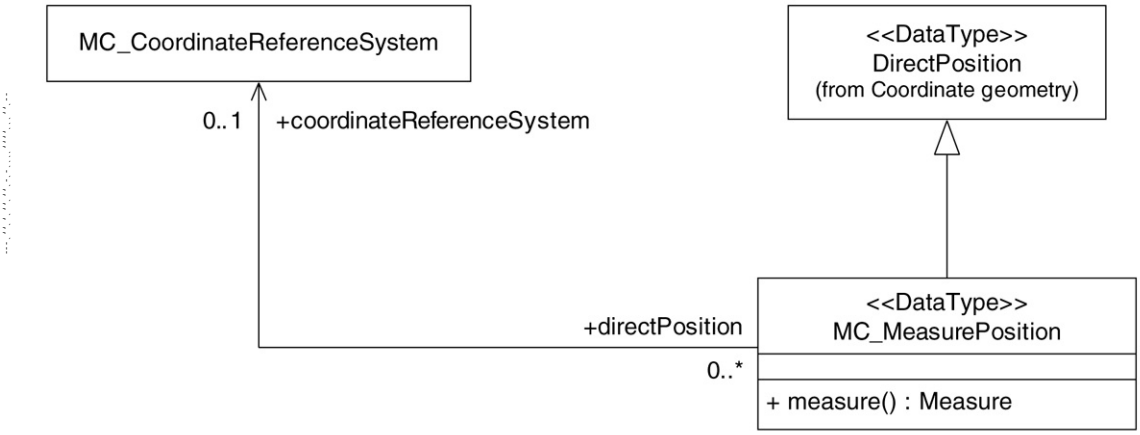

Figure 26 — Context Diagram: MC_MeasurePosition

6.5.3 MC_CoordinateSystem

6.5.3.1 Semantics

The class “MC_CoordinateSystem” describes coordinate systems that have been augmented with measures. The UML for MC_CoordinateSystem is given in Figure 27.

6.5.3.2 Role: axis[1..*] : SC_CoordinateSystemAxis

The role “axis” aggregates the axes for the coordinate reference system. This association is inherited form ISO 19111, but is modified here to accommodate additional axes for the measures. Normally, only one strictly monotonic measure axis is needed, as other measures can be implemented as functions of this one. The most common measure for this is some version of arc length for curves.

```
MC_CoordinateSystem :: axis[1..*]: SC_CoordinateSystemAxis
```

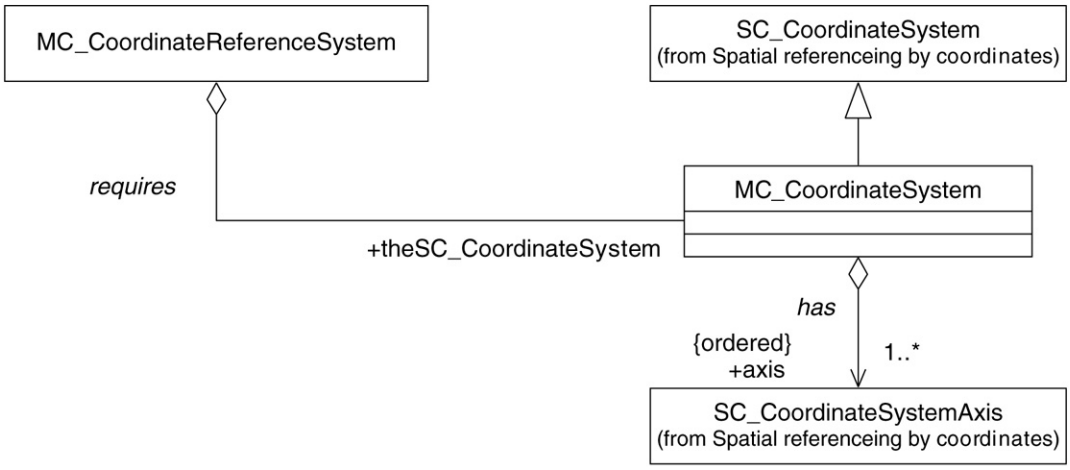

Figure 27 — Context Diagram: MC_CoordinateSystem

6.5.4 MC_CoordinateReferenceSystem

6.5.4.1 Semantics

The class “MC_CoordinateReferenceSystem” extends the class “SC_CoordinateReferenceSystem” defined in ISO 19111 to allow for additional coordinate axes to accommodate measures. The UML for MC_CoordinateReferenceSystem is given in Figure 28.

6.5.4.2 Role: theSC_CoordinateSystem : MC_CoordinateSystem

The association role “theSC_CoordinateSystem” inherited from ISO 19111 is extended here to allow for target class changes.

```
MC_CoordinateReferenceSystem ::
      theSC_CoordinateSystem : MC_CoordinateSystem
```

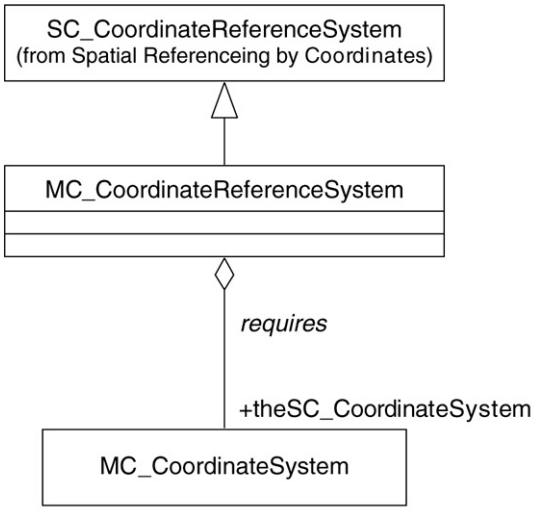

Figure 28 — Context Diagram: MC_CoordinateReferenceSystem

6.6 Package: Linear Reference Systems

6.6.1 Semantics

The package “Linear Reference Systems” supplies classes and types to the definition of linear reference systems. Linear reference systems are in wide use in transportation. They allow for the specification of positions along curvilinear features by using measured distances from known positions, usually represented by physical markers along the right-of-way of the transportation feature. The classes for this system and their relationships are depicted in Figure 29.

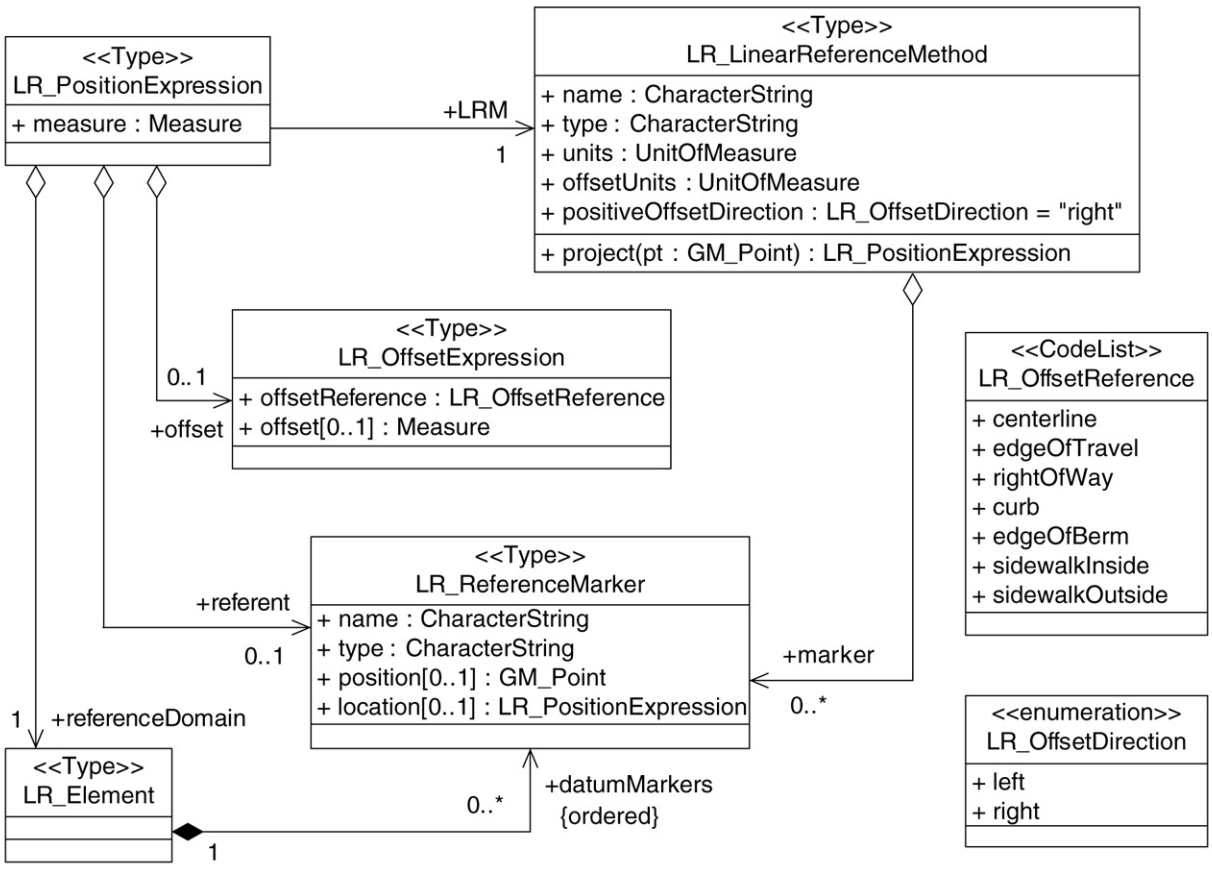

Figure 29 — LRS classes

6.6.2 LR_PositionExpression

6.6.2.1 Semantics

The class “LR_PositionExpression” is used to describe position given by a measure value, a curvilinear element being measured, and the method of measurement. The UML for LR_PositionExpression is given in Figure 30.

6.6.2.2 Attribute: measure : Measure

The attribute “measure” gives the measure (usually a distance) of this position expression.

```
LR_PositionExpression :: measure : Measure
```

6.6.2.3 Role: LRM : LR_LinearReferenceMethod

The role “LRM” gives the linear reference method used for this position expression.

```
LR_PositionExpression :: LRM : LR_LinearReferenceMethod
```

6.6.2.4 Role: referent [0..1] : LR_ReferenceMarker

The optional association role “referent” gives the marker or known position from which the measure is taken for the linear reference method used for this position expression. If the referent is absent, the measurement is made from the start of the LR_element.

```
LR_PositionExpression :: referent [0..1]: LR_ReferenceMarker
```

6.6.2.5 Role: referenceDomain : LR_Element

The role “referenceDomain” gives the linear object upon which the measure is taken for the linear reference method used for this position expression.

```
LR_PositionExpression :: referenceDomain : LR_Element
```

6.6.2.6 Role: offset[0..1] : LR_OffsetExpression

The optional association role “offset” gives perpendicular distance offset of this position expression. If the offset is absent, then the position is on the LR_element.

```
LR_PositionExpression :: offset[0..1] : LR_OffsetExpression
```

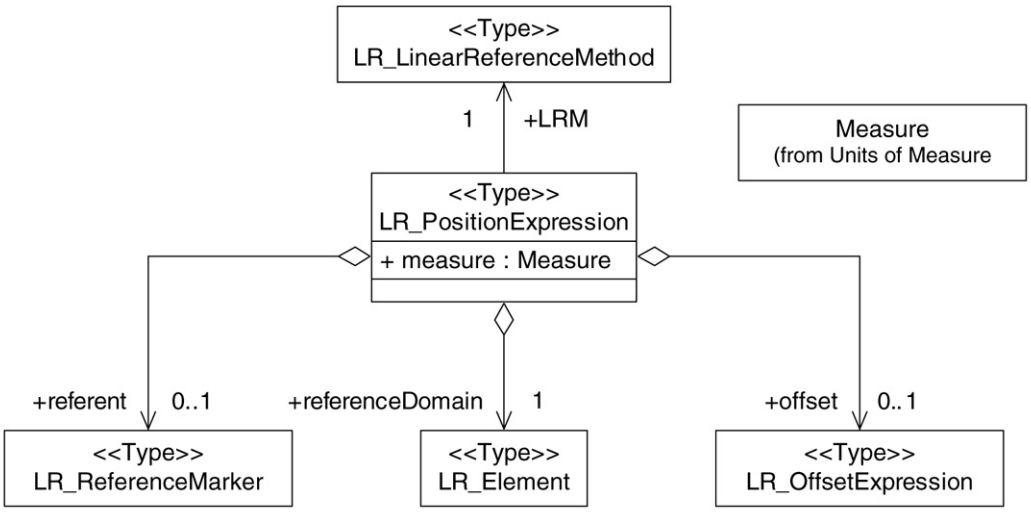

Figure 30 — Context Diagram: LR_PositionExpression

6.6.3 LR_LinearReferenceMethod

6.6.3.1 Semantics

The type “LR_LinearReferenceMethod” describes the manner in which measurements are made along (and optionally laterally offset from) a curvilinear element. The UML for LR_LinearReferenceMethod is given in Figure 31.

6.6.3.2 Attribute: name : CharacterString

The attribute: “name” gives the name of this linear reference method.

```
LR_LinearReferenceMethod :: name : CharacterString
```

6.6.3.3 Attribute: type : CharacterString

The attribute: “type” gives the type of this linear reference method.

```
LR_LinearReferenceMethod :: type : CharacterString
```

6.6.3.4 Attribute: units : UnitOfMeasure

The attribute: “units” gives the units of measure used for this linear reference method for measures along the base elements.

```
LR_LinearReferenceMethod :: units : UnitOfMeasure
```

6.6.3.5 Attribute: offsetUnits : UnitOfMeasure

The attribute: “offsetUnits” gives the units of measure used for this linear reference method for measures perpendicular to the base elements.

```
LR_LinearReferenceMethod :: offsetUnits : UnitOfMeasure
```

6.6.3.6 Attribute: positiveOffsetDirection : LR_OffsetDirection = "right"

The attribute: “positiveOffsetDirection” gives the direction used as positive for this linear reference method for measures perpendicular to the base elements. The default value is right for positive, left for negative.

```
LR_LinearReferenceMethod ::
      positiveOffsetDirection : LR_OffsetDirection = "right”
```

6.6.3.7 Role: marker[1..*] : LR_ReferenceMarker

The association role “marker” aggregates all reference markers used by the linear reference methods. Normally, this will be grouped by linear element.

```
LR_LinearReferenceMethod :: marker[0..*] : LR_ReferenceMarker
```

6.6.3.8 Role: referenceElement[1..*] : LR_Element

The role: “referenceElement” aggregates all the linear elements along which this method is supported.

```
LR_LinearReferenceMethod :: referenceElement[1..*] : LR_Element
```

6.6.3.9 Operation: project

The operation “project” will find the measure of the point on a base element closest to the given point, and then express the point as a position expression for the linear reference method. If the point is precisely on one of the linear elements, then the offset will be zero there is no offset expression.

```
LR_LinearReferenceMethod ::
      project(pt : GM_Point) : LR_PositionExpression
```

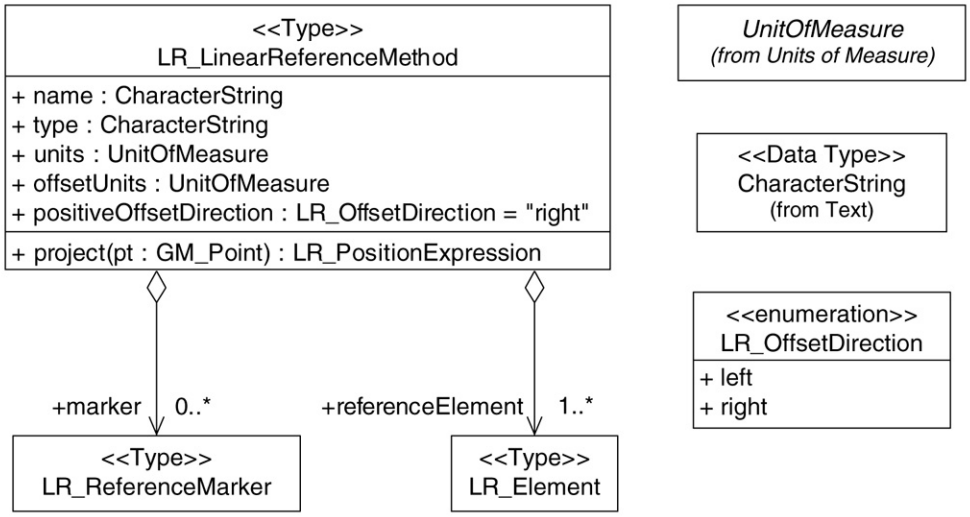

Figure 31 — Context Diagram: LR_LinearReferenceMethod

6.6.4 LR_OffsetDirection

The enumeration “LR_OffsetDirection” gives the two options for offset measure. The values are left and right. This offset direction is as viewed from above the linear element facing in the direction of increasing measure. The UML for LR_OffsetDirection is given in Figure 32.

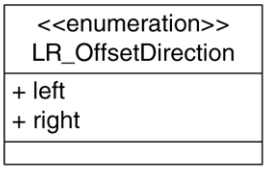

Figure 32 — Context Diagram: LR_OffsetDirection

6.6.5 LR_ReferenceMarker

6.6.5.1 Semantics

The type “LR_ReferenceMarker” is used to describe reference markers used in linear reference systems. At least one of the attributes “position” or “location” shall be given. If both are given they shall refer to the same physical location. The UML for LR_ReferenceMarker is given in Figure 33.

6.6.5.2 Attribute: name : CharacterString

The attribute “name” is the identifier used for this marker.

```
LR_ReferenceMarker :: name : CharacterString
```

6.6.5.3 Attribute: type : CharacterString

The attribute “type” is the type of this marker.

```
LR_ReferenceMarker :: type : CharacterString
```

6.6.5.4 Attribute: position[0..1] : GM_Point

The optional attribute “position” is the position of this for this marker, given in some coordinate system. If this attribute is not given, then the “location” shall be given.

```
LR_ReferenceMarker :: position[0..1] : GM_Point
```

6.6.5.5 Attribute: location[0..1] : LR_PositionExpression

The optional attribute “location” is the location of this marker given as a linear reference measure along and from the start of the underlying linear element.

```
LR_ReferenceMarker :: location[0..1] : LR_PositionExpression
```

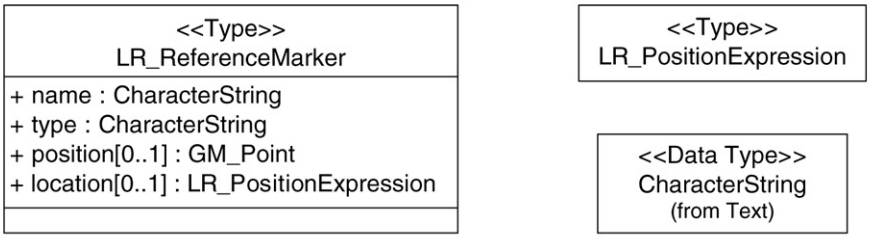

Figure 33 — Context Diagram: LR_ReferenceMarker

6.6.6 LR_OffsetReference

The code list “LR_OffsetReference” enumerates the offset reference types used for this linear reference method (see Figure 36). The initial value domain included:

1) “centerline” centre of the structure of the highway, or reference line for the highway
2) “edgeOfTravel” outside edge of all travel lanes
3) “edgeOfPavement” outside edge of travel-lane quality paved surface
4) “rightOfWay” edge of the legal right of way
5) “curbFace” curb (the roadway must be curbed for this to be used)
6) “curbBack” curb (the roadway must be curbed for this to be used)
7) “edgeOfShoulder” outside edge of all hardened surface (paved or gravel) surface
8) “edgeOfBerm” outside edge of levelled land for the road structure
9) “walkwayInside” sidewalk edge closest to travel lanes
10) “walkwayOutside” sidewalk edge furthest from travel lanes

6.6.7 LR_Feature

The type “LR_Feature” is a behavioural description of features used as base elements in a linear reference method. This is the most common approach used for LRS’s. The UML for LR_Feature is given in Figure 34.

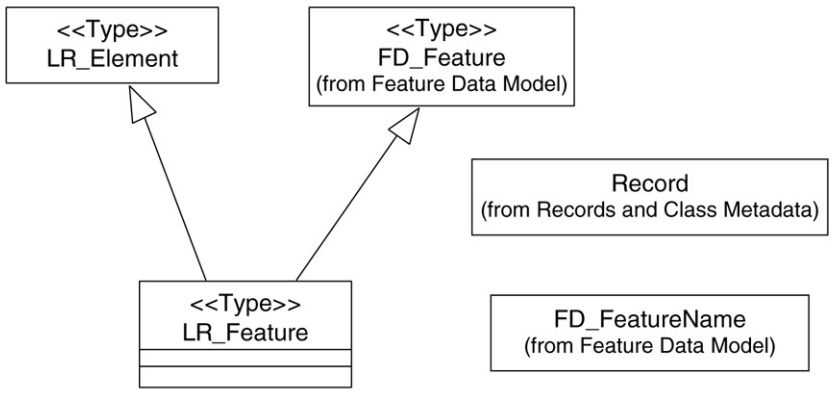

Figure 34 — Context Diagram: LR_Feature

6.6.8 LR_Element

6.6.8.1 Semantics

The type “LR_Element” describes the underlying curvilinear elements upon which the measures in the linear reference system are taken. The UML for LR_Element is given in Figure 35.

6.6.8.2 Role: datumMarkers[1..*] : LR_ReferenceMarker

The ordered association role “datumMarkers” aggregates the markers along this element. The ordering of the markers is consistent with the order in which the markers would be found in traversing the LR_Element from beginning to end (i.e. in increasing order of distance from the “zero marker” the beginning of the element).

```
LR_Element :: datumMarkers[1..*] : LR_ReferenceMarker
```

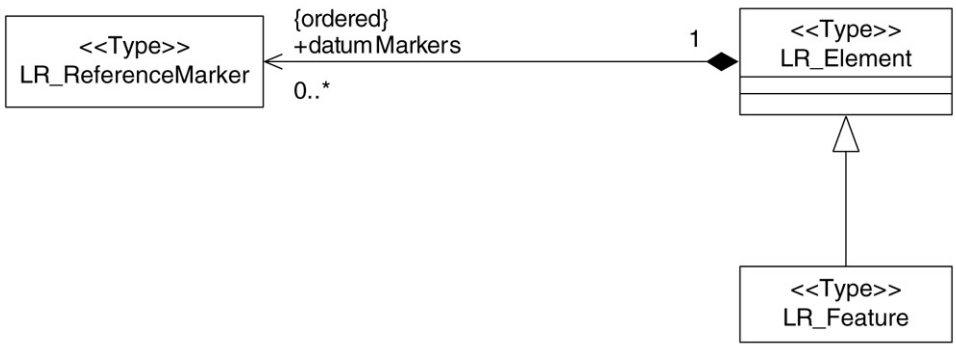

Figure 35 — Context Diagram: LR_Element

6.6.9 LR_OffsetExpression

6.6.9.1 Semantics

The type “LR_OffsetExpression” is used to describe the offset for a position described using a linear reference method. The UML for LR_OffsetExpression is given in Figure 36.

6.6.9.2 Attribute: offsetReference : LR_OffsetReference

The attribute “offsetReference” indicates the base line for the offset measure.

```
LR_OffsetExpression :: offsetReference : LR_OffsetReference
```

6.6.9.3 Attribute: offset[0..1] : Measure

The optional attribute “offset” is the measure of the offset of the position expression. A missing value is to be interpreted as being located at the offset reference.

```
LR_OffsetExpression :: offset[0..1] : Measure
```

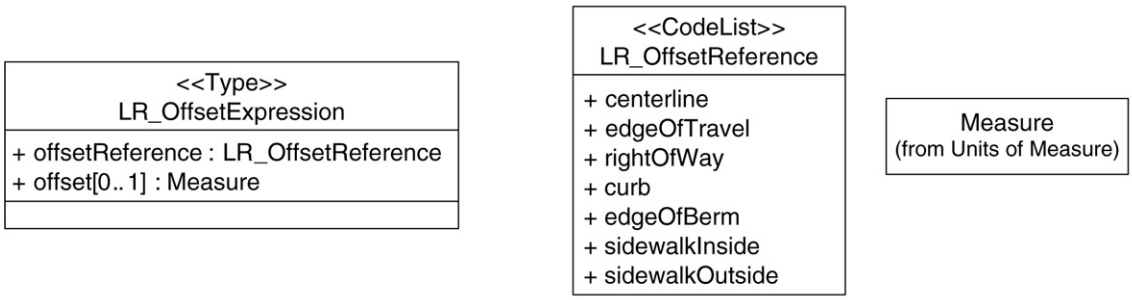

Figure 36— Context Diagram: LR_OffsetExpression

7 Navigation

7.1 Semantics

7.1.1 Package structure

The package “Navigation” supplies classes and types to describe navigation services and their supporting data.

Tracking is the process of following the position of a vehicle in a network, and associating it to steps in a route. Routing is the finding of optimal (minimal cost function) routes between positions in a network. Route traversal is the execution of a route, usually through the use of instructions at each node in the path, and a start and stop instruction, at the first and last position of the route. The combination of routing, route transversal and tracking is Navigation. The various subpackages for Navigation are shown in Figure 37.

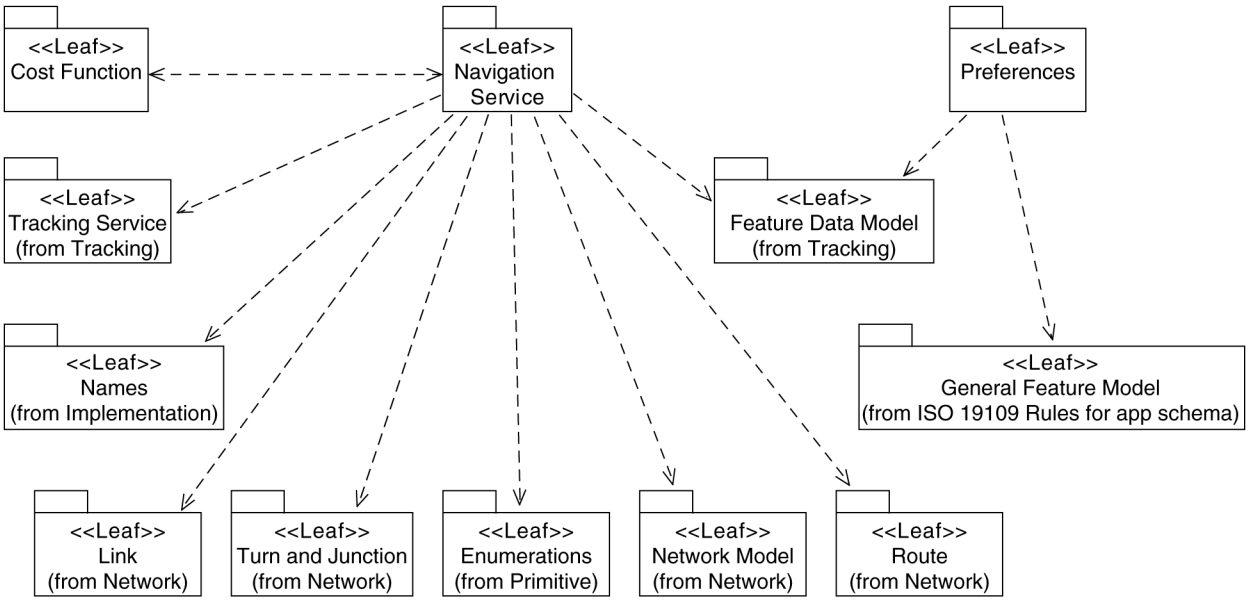

Figure 37 — Navigation Packages

7.1.2 Route

A “route” represents a possible path for a vehicle to navigate across some portion of a network. A generic example of the geometry of a route is given in Figure 38. A route consists of the following parts.

a) Nodes that represent important points in the network, such as intersections.

b) Links that represent uninterrupted paths between nodes with an orientation that indicates which direction the link is to be traversed.

c) Turns that associate a node with an entry link and exit link to a node.

d) Stops that consist of either nodes or positions on links within the network:

— the start of the route that is a type of stop;

— the end of the route that is a type of stop.

The route can be sorted into the following.

1 The start position.

2 A link consistent with this start position:

— if the start is a node, then the link is an exit for that node;

— if the start is a link position, then the link is the one on which that position is located.

2n+1 A turn which includes the previous link as its entry, and the following link as its exit.

2n A link that is the exit of the previous turn and an entry of the following turn.

last the stop position.

In Figure 38, there is a composite curve, which is the underlying geometry for a route. The route begins at a position on the first “link” element of the composite, and then traverses to the next node. That node is associated with a “turn” that transitions to the next link, and so forth, until the last link is reached. The route terminates at some position on the last link, possibly the end node.

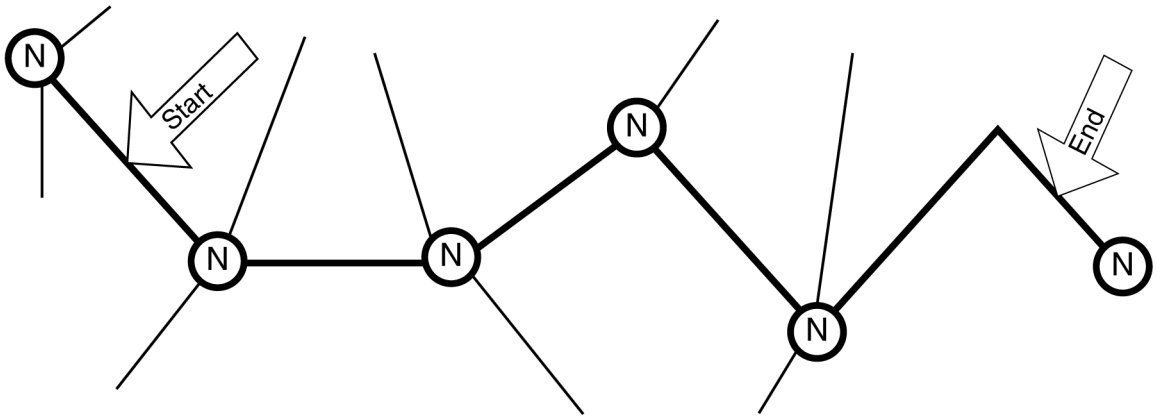

Figure 38 — Example of route from one link position to another

7.1.3 Turns

A turn is the description of a possible traversal of a node. It consist of three major parts:

— the incoming link,

— the junction (node),

— the outgoing link.

In addition, restrictions on how the turn may be executed are associated with the turn. This included such things as which lanes in the incoming link access the turn, and whatever controlling appliance may exit there.

7.1.4 Maneuver

A maneuver aggregates legal combinations of turns with their spanning links. Some turns are maneuvers in their own right (and can be treated as such). Some combinations of turns are either illegal or impossible, and are not traversable maneuvers. When a vehicle comes to a decision point in the network, a choice between the legal maneuvers at that point must be made for the vehicle to move onward.

7.1.5 Junctions

A junction is a collection of all the turns at a particular node.

7.1.6 Links

Links are directed edges between nodes, with additional information on navigation. Each edge will have at least two links, one for each directed edge. Multiple links for each directed edge are possible, but no use for such a generalization is seen at this time.

7.1.7 Network positions

Network positions are positions described not by coordinates from a coordinate reference system, but in relation to objects within the network. For positions related to nodes, the node is sufficient to describe the position. For an edge, the link determines the “side” of the road, and some measure is used to specify the specific position upon that link. The most common and natural measure is the length along the link from its start point that corresponds to the position.

7.2 Cost Functions and algorithms

Much of this International Standard has been affected by the assumption that implementations of the implied applications would use one of the well-known algorithms for finding the shortest path through a weighted graph. The two primary such algorithms are known as Dijkstra’s algorithm [7], and the Bellman–Ford algorithm [4], [8]. Other algorithms are possible. Some basic understanding of these types of algorithms is helpful in understanding some of the decisions made in formulating this International Standard (see Annex B).

The solution in this model involves the concept of a maneuver. A maneuver aggregates legal combinations of turns (with their spanning links) into a single edge in the Dijkstra graph.

For this International Standard, cost functions are divided into groups based upon attributes: simple or complex, static, predictive or dynamic. Simple functions are easy to calculate and approximate cost in terms of either time or distance. They tend to take a simplified view of the world, but may approximate reality to such a degree to assure an optimal or near-optimal route selection. Complex cost functions take a richer view of the world, taking many factors into consideration. They are more complex, more difficult to use, but may give more realistic results in unusual situations. Static cost functions take average values for cost and will return the same answer regardless of actual time constraints on targets. Predictive cost functions take time into account, but only in terms of a priori averages, and not actively retrieved current values. Predictive cost functions are usually based on long-term histories of the link, and are separated to some degree on time of day, and date (day of week, type of day). Dynamic cost functions take real-time variables into consideration and modify link and node measures based first upon predictive variables but eventually on the values of these variables based on the actual time of traversal. Dynamic functions use real-time data and can be used for dynamic routing that modifies routing based on current traffic and road conditions.

Navigation services may provide various levels of service depending on the type of measure supported and type of cost functions used.

A basic navigation service shall support at least cost functions based on distance and expected average time.

A predictive navigation service is a basic service that shall be able to take the chosen time of day and date into account for predicting time of traversal.

A real-time navigation service is a predictive service that shall be able to monitor traffic and road conditions and to reroute based on current information.

A multiple-stop service is a basic, predictive or real-time service that shall be able to handle multiple stops (uncosted) along the route (see model for a description of how multiple stops are defined).

A complex navigation service is a multiple-stop, real-time service that shall be able to include cost based on activities associated with traversal of the route, such as costing stops based on price of activities at those stops (see the description of cost functions below).

Each of these services may require other data beyond what is needed for the simpler service, or may require external services to supply extra data. This is described in Table 1.

Table 1 — Navigation service type summary

Navigation service level	Link and maneuver measures required	Node measure required	Cost function types	Time usage	Other data, services required
basic	length, average time, address	none	distance, time	static	yellow pages, geolocation
predictive	length, average time, average speed, address	time	distance, time	static, predictive	yellow pages, geolocation
real-time	length, average time, average speed, address	time	distance, time	static, predictive dynamic	yellow pages, geolocation
multiple-stop (basic, predictive, or real-time)	length, average time, average speed, address	time	distance, time	static, predictive dynamic	yellow pages, geolocation
complex	length, average time, average speed, address, slope, others	time, other	distance, time, cost, price	static, predictive dynamic	yellow pages, geolocation, price directory, etc.

7.3 Package: Navigation Service

7.3.1 Semantics

The package “Navigation Service” provides classes that describe the actual services themselves. The basic services in navigation are depicted in the UML of Figure 39.

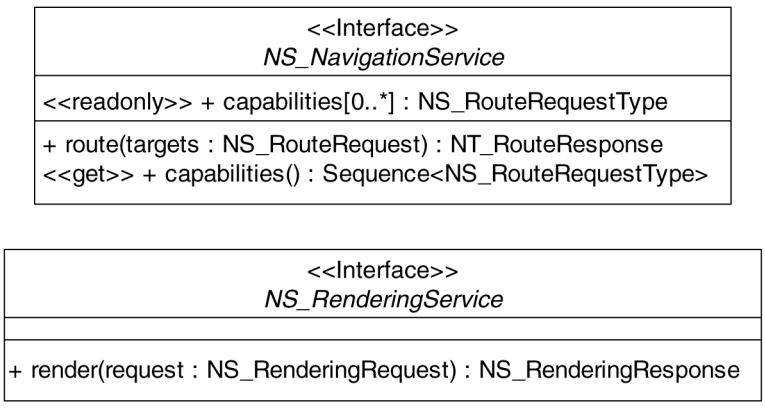

Figure 39 — Services

7.3.2 NS_NavigationService

7.3.2.1 Semantics

The interface “NS_NavigationService” supplies operations central to navigating and tracking a single target. The UML for NS_NavigationService is given in Figure 40.

7.3.2.2 Attribute: capabilities [0..*]: NS_RouteRequestType

The attribute “capabilities” list the types of request that this service can handle.

```
NS_ BasicNavigationService :: capabilities[0..*] : NS_RouteRequestType
```

7.3.2.3 Operation: route

This operation “route” returns a proposed route matching the criteria passed in through the request, which is optimal for these criteria.

```
NS_ BasicNavigationService ::
      route(targets : NS_RouteRequest) : NT_RouteResponse
```

7.3.2.4 Operation: capabilities

The operation “capabilities” returns a sequence of route request types that are properly handled by the service supporting this interface.

```
NS_NavigationService ::
      capabilities() : Sequence<NS_RouteRequestType>
```

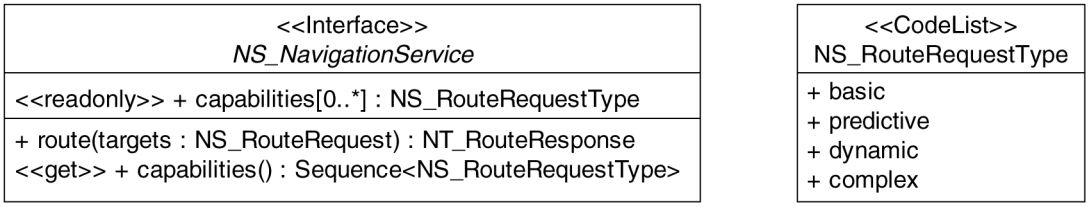

Figure 40 — Context Diagram: NS_NavigationService

7.3.3 NS_RouteRequest

7.3.3.1 Semantics

The data type “NS_RouteRequest” is the navigation request specifying the source (starting point), waypoints and destination (ending point) of the requested route. Cost functions used are based on general criteria. Depending on the capabilities of the routing services, other attributes are added. The UML for NS_RouteRequest is given in Figure 41.

7.3.3.2 Attribute: routeRequestType [1..*] : NS_RouteRequestType

The attribute “routeRequestType” indicates the type of request being made. The types “basic” and “complex” cannot be used together. The “complex” type requires that a cost function be specified. “Predictive” will require a time statement for either departure or arrival. “Dynamic” will require an update interval be specified.

```
NS_RouteRequest :: routeRequestType [1..*] : NS_RouteRequestType
```

7.3.3.3 Attribute: vehicle : NT_Vehicle

The attribute “vehicle” indicates the type of vehicle for which the route is being requested. It is necessary to know so that the routing service can check constraints against links, turns and maneuvers.

```
NS_RouteRequest :: vehicle : NT_Vehicle
```

7.3.3.4 Attribute: wayPointList[1..*] : NT_WayPointList

The attribute “wayPointList” enumerates the starting point and all stopping points for the required route. Multiple way point list may be given if the route required is essentially a sequence of subroutes, each with its own list.

```
NS_RouteRequest :: wayPointList[1..*] : NT_WayPointList
```

7.3.3.5 Attribute: avoidList : NS_AvoidList

The attribute “avoidList” describes the items to avoid in determining candidate routes for this request.

```
NS_ComplexRouteRequest :: avoidList : NS_AvoidList
```

7.3.3.6 Attribute: departureTime[0..1] : TM_Period

The attribute “departureTime” is a temporal period during which the traveller plans to begin navigation. The departure time or arrival time shall be specified if this request is used for predictive or dynamic routing.

```
NS_RouteRequest :: departureTime[0..1] : TM_Period
```

7.3.3.7 Attribute: arrivalTime[0..1] : TM_Period

The attribute “arrivalTime” is a temporal period during which the traveller plans to begin navigation. The departure time or arrival time shall be specified if this request is used for predictive or dynamic routing.

```
NS_RouteRequest :: arrivalTime[0..1] : TM_Period
```

7.3.3.8 Attribute: costFunction[0..1] : NS_CostFunctionCode = "distance"

The attribute “costFunction” specifies the cost function used to choose among the candidate routes.

```
NS_RouteRequest :: costFunction[0..1] : NS_CostFunctionCode = "distance"
```

7.3.3.9 Attribute: preferences[0..*] : NS_RoutePreferences

The attribute “preferences” enumerates the user preference for the required route.

```
NS_RouteRequest :: preferences[0..*] : NS_RoutePreferences
```

7.3.3.10 Attribute: advisories[0..*] : NT_AdvisoryCategory

The attribute “advisories” enumerates the types of advisories to be returned with the requested route.

```
NS_RouteRequest :: advisories[0..*] : NT_AdvisoryCategory
```

7.3.3.11 Attribute: isDynamic : Boolean

The Boolean attribute “isDynamic” specifies whether this request is slated for dynamic update.

```
NS_RouteRequest :: isDynamic: Boolean = “FALSE”
```

7.3.3.12 Attribute: refreshInterval : TM_Duration

The attribute “refreshInterval” is the maximum duration between dynamic recalculations of the route during execution.

```
NS_RouteRequest :: refreshInterval : TM_Duration
```

7.3.3.13 Attribute: returnRouteInstructions : Boolean

The Boolean attribute “returnRouteInstructions” specifies whether a full instruction set should be returned with the requested route.

```
NS_RouteRequest :: returnRouteInstructions : Boolean
```

7.3.3.14 Attribute: returnRouteMaps : Boolean

The Boolean attribute “returnRouteMaps” specifies whether a set of maps should be returned with the requested route.

```
NS_RouteRequest :: returnRouteMaps : Boolean
```

7.3.3.15 Attribute: returnRouteGeometry : Boolean

The Boolean attribute “returnRouteGeometry” specifies whether summary geometry should be returned with the requested route.

```
NS_RouteRequest :: returnRouteGeometry : Boolean
```

7.3.3.16 Role: costFunction : NS_CostFunction

The association role “costFunction” indicates the cost function to use for this request.

```
NS_ComplexRouteRequest :: costFunction : NS_CostFunction
```

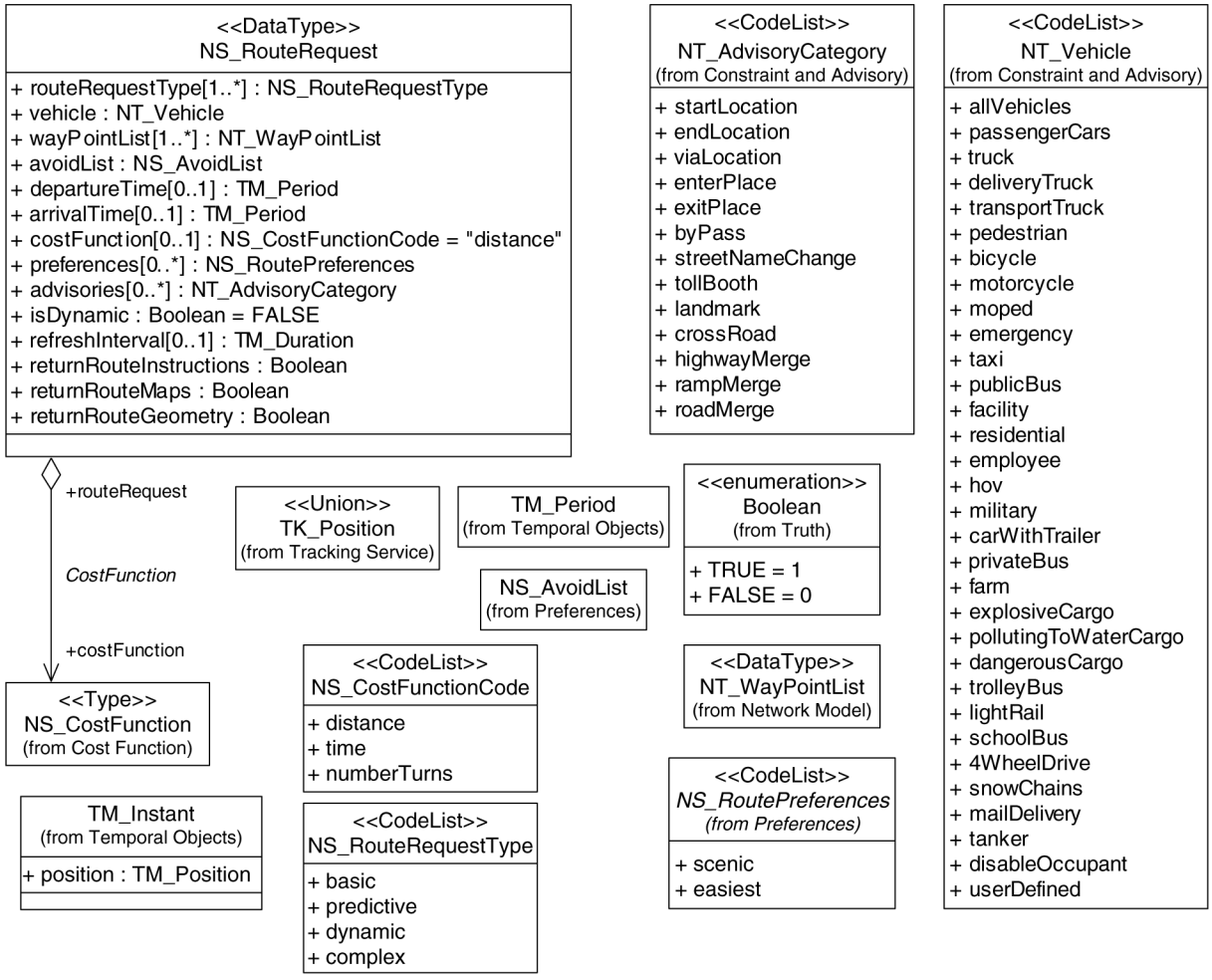

Figure 41 — Context Diagram: NS_RouteRequest

7.3.4 NS_Instruction

7.3.4.1 Semantics

The data type “NS_Instruction” describes a single route instruction or advisory. An instruction can cover multiple maneuvers as in “turn right and then immediately left”. Most such examples involve maneuvers that are not separated by “long” links. A route can be navigated by executing a sequence of instructions. Instructions are required for each maneuver that does not execute the main-road rule. The UML for NS_Instruction is given in Figure 42.

7.3.4.2 Role: maneuver : NS_Maneuver

The role “maneuver” is the maneuver being executed by this instruction.

```
NS_Instruction :: maneuver : NS_Maneuver
```

7.3.4.3 Attribute: cost : Measure

The attribute “cost” is the cost calculated for this maneuver, and the following link (which takes the route up to the next maneuver/instruction). For example, the instruction might say “turn right and go 40km”, in which case the value for a distance-based cost function is “40km”.

```
NS_Instruction :: cost : Measure
```

7.3.4.4 Attribute: action[0..1] : CharacterString

The attribute “action” describes the action to be taken to navigate through the associated maneuver.

```
NS_Instruction :: action[0..1] : CharacterString
```

7.3.4.5 Attribute: advisory[0..*] : NT_Advisory

The attribute “advisory” describes any advisory associated with the maneuvers being executed by the instruction.

```
NS_Instruction :: advisory[0..*] : NT_Advisory
```

7.3.4.6 Operation: renderAsMap

The operation “renderAsMap” renders the instruction as a map view of the associated maneuvers.

```
NS_Instruction :: renderAsMap(scale : Scale) : Map
```

7.3.4.7 Operation: renderAsVoice

The operation “renderAsVoice” renders the instruction as a voice data stream.

```
NS_Instruction :: renderAsVoice() : VoiceStream
```

7.3.4.8 Operation: renderAsText

The operation “renderAsText” renders the instruction as a text stream.

```
NS_Instruction :: renderAsText() : CharacterString
```

7.3.4.9 Operation: renderAsGroundLevelView

The operation “renderAsGroundLevelView” renders the instruction as a ground level view.

```
NS_Instruction :: renderAsGroundLevelView() : Image
```

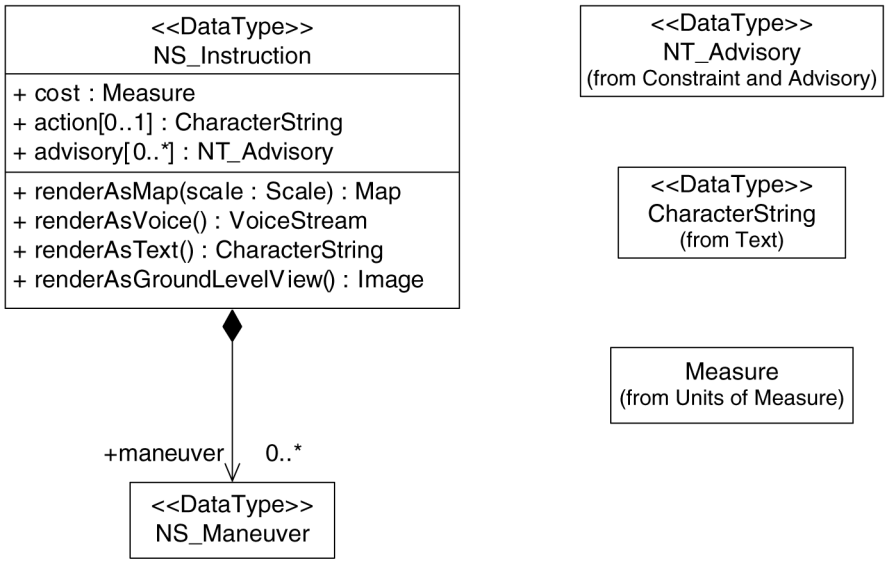

Figure 42 — Context Diagram: NS_Instruction

7.3.5 NS_InstructionList

7.3.5.1 Semantics

The data type “NS_InstructionList” is an ordered list of instructions. The UML for NS_InstructionList is given in Figure 43.

7.3.5.2 Role: instruction[1..*] : NS_Instruction

The association role “instructions” aggregates the instructions in the list.

```
NS_InstructionList :: instruction[1..*] : NS_Instruction
```

7.3.5.3 Attribute: route : NT_Route

The attribute “route” describes the route that execution of the instructions in the list will execute.

```
NS_InstructionList :: route : NT_Route
```

7.3.5.4 Operation: renderAsMap

The operation “renderAsMap” renders the instruction list as a sequence of map views of the associated maneuvers.

```
NS_Instruction :: renderAsMap(scale : Scale) : Sequence<Map>
```

7.3.5.5 Operation: renderAsVoice

The operation “renderAsVoice” renders the instruction list as a sequence of voice data streams.

```
NS_Instruction :: renderAsVoice() : Sequence<VoiceStream>
```

7.3.5.6 Operation: renderAsText

The operation “renderAsText” renders the instruction list as a text stream.

```
NS_Instruction :: renderAsText() : Sequence<CharacterString>
```

7.3.5.7 Operation: renderAsGroundLevelView

The operation “renderAsGroundLevelView” renders the instruction list as a sequence of ground level views.

```
NS_Instruction :: renderAsGroundLevelView() : Sequence<Image>
```

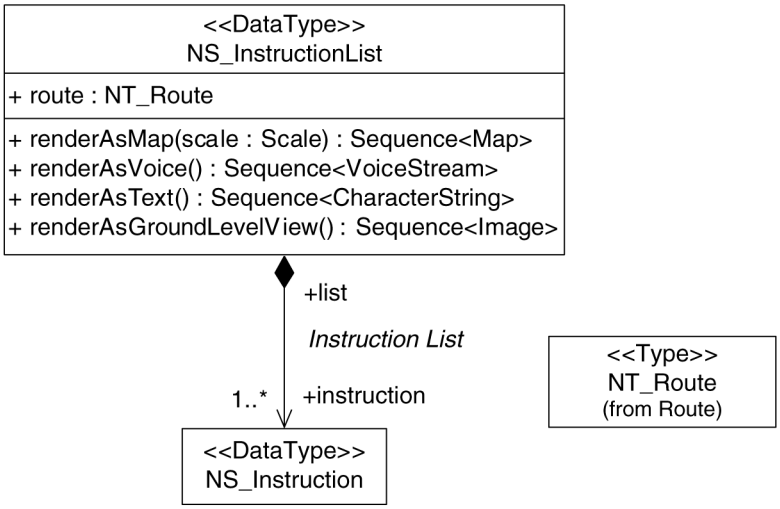

Figure 43 — Context Diagram: NS_InstructionList

7.3.6 NS_RouteResponse

7.3.6.1 Semantics

The data type “NS_RouteResponse” describes the route response for a navigation service. The UML for NS_RouteResponse is given in Figure 44.

7.3.6.2 Attribute: request[0..1] : NS_RouteRequest

The attribute “request” contains the request information that caused this route to be generated. It can be eliminated if there is no chance of confusion.

```
NS_RouteResponse :: request[0..1] : NS_RouteRequest
```

7.3.6.3 Attribute: route[0..*] : NT_Route

The attribute “route” contains the requested route (unless an error was found in the request that prevented a route from being calculated).

```
NS_RouteResponse :: route[0..1] : NS_Route
```

7.3.6.4 Role: rendering[0..*] : NS_RenderingResponse

The role “rendering” contains the requested route renderings (unless an error was found in the request that prevented a route from being calculated).

```
NS_RouteResponse :: rendering[0..*] : NS_RenderingResponse
```

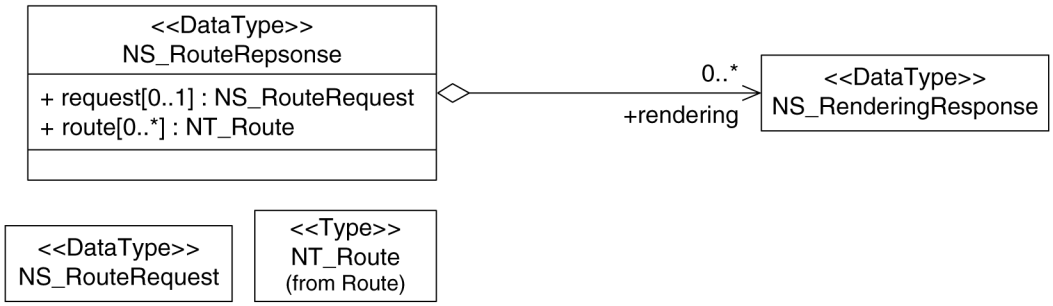

Figure 44 — Context Diagram: NS_RouteResponse

7.3.7 NS_CostedTurn

7.3.7.1 Semantics

The class “NS_CostedTurn” is used to represent a turn for which the cost has been calculated according to the indicated cost function. The UML for NS_CostedTurn is given in Figure 45.

7.3.7.2 Role: costFunction : NS_CostFunction

The association role “costFunction” indicates the cost function used to calculate the cost for this turn.

```
NS_CostedTurn :: costFunction : NS_CostFunction
```

7.3.7.3 Role: turn : NT_Turn

The association role “turn” indicates the turn that represents this costed turn with the cost stripped out (or not yet calculated).

```
NS_CostedTurn :: turn : NT_Turn
```

7.3.7.4 Attribute: cost : Measure

The attribute “cost” is the cost of this turn according to the cost function given by the association role “cost function".

```
NS_CostedTurn :: cost : Measure
```

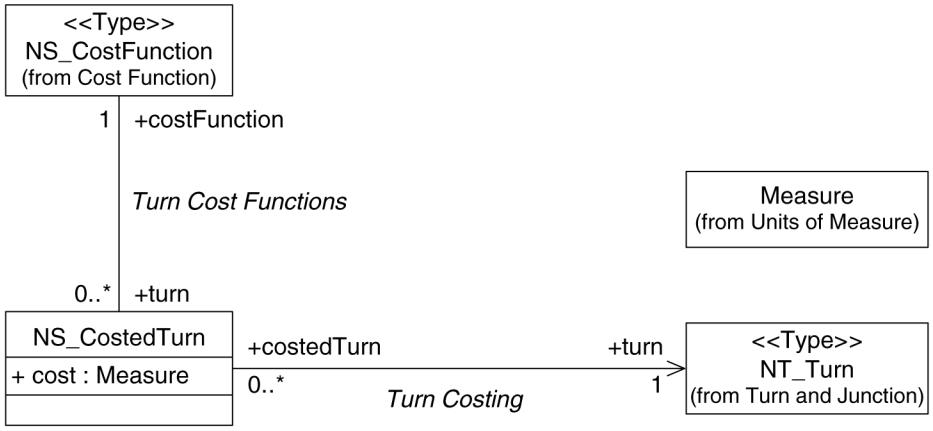

Figure 45 — Context Diagram: NS_CostedTurn

7.3.8 NS_RenderingService

7.3.8.1 Semantics

The interface “NS_RenderingService” specifies the interfaces required of a portrayal service that will render the calculated routes in a form suitable for the user interface requirements. The UML for NS_RenderingService is given in Figure 46.

7.3.8.2 Operation: render

The operation “render” translates a route into one or more appropriate portrayals.

```
NS_RenderingService ::
      render(request : NS_RenderingRequest) : NS_RenderingResponse
```

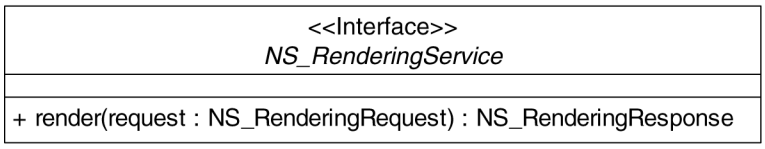

Figure 46 — Context Diagram: NS_RenderingService

7.3.9 NS_RenderingRequest

7.3.9.1 Semantics

The data type “NS_RenderingRequest” formats a request to translate a route into one or more forms usable in the navigation services. The UML for NS_RenderingRequest is given in Figure 47.

7.3.9.2 Attribute: route : NT_Route

The attribute “route” specifies the route to be rendered or portrayed.

```
NS_RenderingRequest :: route : NT_Route
```

7.3.9.3 Attribute: type[1..*] : NS_RenderingType

The attribute “type” specifies the type of portrayal required

```
NS_RenderingRequest :: type[1..*] : NS_RenderingType
```

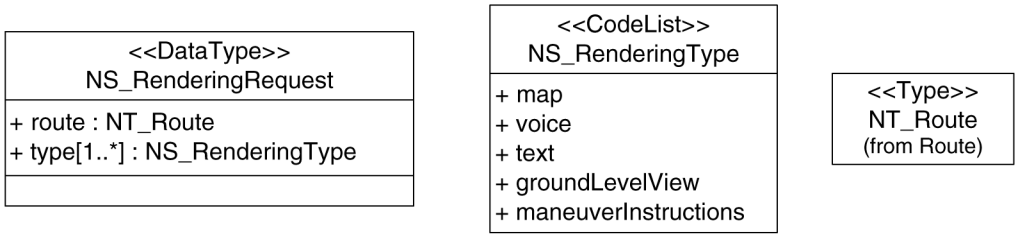

Figure 47 — Context Diagram: NS_RenderingRequest

7.3.10 NS_RenderingResponse

7.3.10.1 Semantics

The data type “NS_RenderingResponse” contains rendered instructions corresponding to a route. The UML for NS_RenderingResponse is given in Figure 48.

7.3.10.2 Attribute: map[0..1] : Sequence<Map>

The attribute “map” contains the route rendered as a sequence of maps.

```
NS_RenderingResponse :: map[0..1] : Sequence<Map>
```

7.3.10.3 Attribute: voice[0..1] : Sequence<VoiceStream>

The attribute “voice” contains the route rendered as a sequence of voice data streams.

```
NS_RenderingResponse :: voice[0..1] : Sequence<VoiceStream>
```

7.3.10.4 Attribute: text[0..1] : Sequence<CharacterString>

The attribute “text” contains the route rendered as a sequence of character strings.

```
NS_RenderingResponse :: text[0..1] : Sequence<CharacterString>
```

7.3.10.5 Attribute: groundLevelView[0..1] : Sequence<Image>

The attribute “groundLevelView” contains the route rendered as a sequence of images depicting the ground level views of the route.

```
NS_RenderingResponse :: groundLevelView [0..1] : Sequence<Image>
```

7.3.10.6 Attribute: instructionList[0..1] : NS_InstructionList

The attribute “instructionList” contains the route rendered as a sequence of instructions, each associated with a maneuver in the returned route.

```
NS_RenderingResponse :: instructionList[0..1] : NS_InstructionList
```

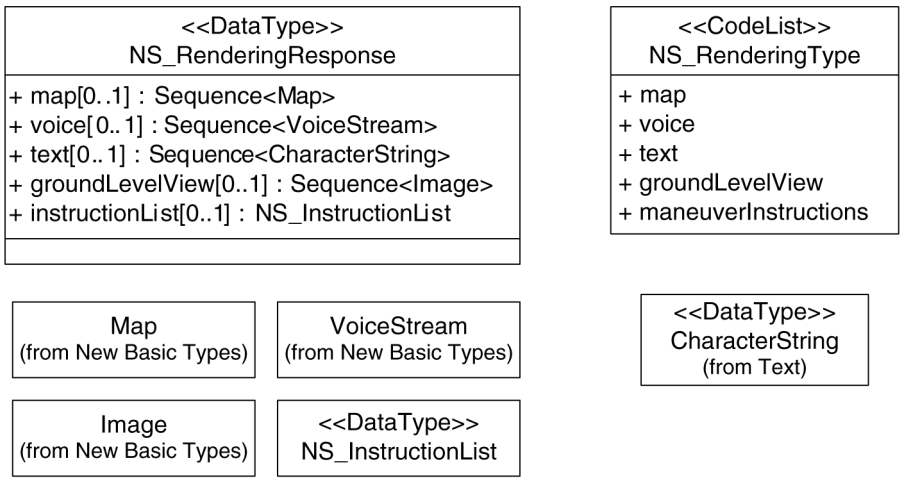

Figure 48 — Context Diagram: NS_RenderingResponse

7.3.11 NS_RenderingType

The code list “NS_RenderingType” supplies the types of portrayal available through the service using it. The initial values are “map”, “voice”, “text”, “groundLevelView” and “maneuverInstructions”. The UML for NS_RenderingType is given in Figure 49.

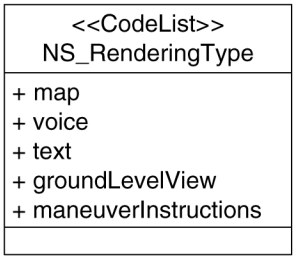

Figure 49 — Context Diagram: NS_RenderingType

7.3.12 NS_CostedLink

7.3.12.1 Semantics

The class “NS_CostedLink” is used to represent links that have been assigned a cost based upon a cost function. The creation of such links is one of the first steps is setting up a graph for calculating minimum cost paths. The UML for NS_CostedLink is given in Figure 50.

7.3.12.2 Role: costFunction : NS_CostFunction

The association role “costFunction” links this costed link to the function used to calculate its cost weight.

```
NS_CostedLink :: costFunction : NS_CostFunction
```

7.3.12.3 Role: link : NT_Link

The association role “link” links this costed link to the underlying costed link.

```
NS_CostedLink :: link : NT_Link
```

7.3.12.4 Attribute: totalCost : Measure

The attribute “totalCost” indicated the cost of traversing this entire link in a route.

```
NS_CostedLink :: totalCost : Measure
```

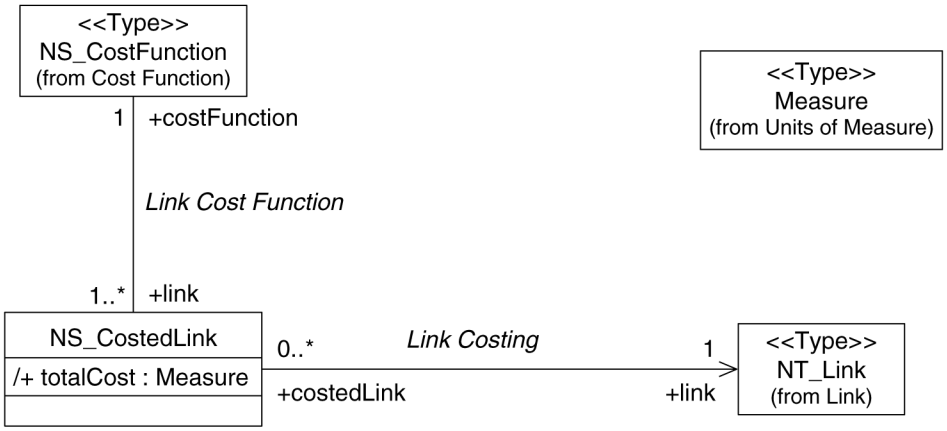

Figure 50 — Context Diagram: NS_CostedLink

7.3.13 NS_CostFunctionCode

The code list “NS_CostFunctionCode” enumerates the types of cost functions supported by the service with this version of the code list. The initial values are “distance”, “time”, and “numberTurns”. The UML for NS_CostFunctionCode is given in Figure 51.

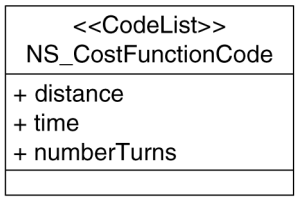

Figure 51 — Context Diagram: NS_CostFunctionCode

7.3.14 NS_RouteRequestType

The code list “NS_RouteRequestType” enumerates the types of route requests. The initial values are “basic”, “predictive”, “dynamic”, and “complex”. The UML for NS_RouteRequestType is given in Figure 52.

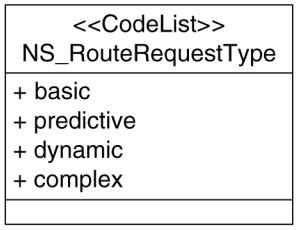

Figure 52 — Context Diagram: NS_RouteRequestType

7.4 Package: Cost Function

7.4.1 Semantics

7.4.1.1 Basic contents

The package “Cost Function” contains classes and types for the description of cost functions for use in route determination.

7.4.1.2 Standard variables

7.4.1.2.1 Semantics

The cost of executing a route (which determines the optimal choice) shall be calculable from the attributes and measures stored for the links (either on the link, or on the underlying edge) and the maneuvers. This section describes the most common variables that may appear within a user defined or standard cost function. The mechanism for storage or calculation of these variables is implementation dependent. These attributes may be stored directly as attributes on the associated objects, or derived from some other service, such as traffic information or travel condition services. Dynamic routing will be sensitive to real-time data, but static or predictive routing will by definition often use averages for time-dependent variables, such as travel or waiting time.

7.4.1.2.2 Distance

The distance travelled along the links is the most fundamental of all the variables, often being the basis for calculating other variables. Distance, or more precisely length, is defined on GM_Curve in ISO 19107. For each directed Curve (Link), it shall be possible to measure the length along this curve to any position on it.

The basic assumption is that distance is only associated with links. Maneuvers derive their lengths from the sum of the lengths of their included links. Turns, which are maneuvers in their own right and independent of the previous path of the vehicle up to that point, have zero (0,0) length. Nodes are considered to be 0-dimensional and hence zero length.

7.4.1.2.3 Time

Travel time to link position from the start node is the second most fundamental variable. The most common assumption is that the link has a constant average speed associated with it, making travel time proportional to length along the link. The simplifying assumption is often sufficient, but for long link, especially in urban areas, average speed may not be so constant. For this reason, time may vary non-linearly with distance. Such conditions may be modelled using linear reference methods defined in 6.6.

7.4.1.2.4 Stopping time

Maneuvers, turns and links may be associated with an average or current stopping time, such as the delay for a traffic control mechanism (e.g. a light).

7.4.1.2.5 Speed

Speed, time and distance are obviously related, and only two of them need to be stored, with the third being calculated from well-known formulae.

7.4.1.2.6 Speed limits

Constraints play a large part in navigation. One of the most obvious is the legal (or possibly physical) limit on speed over links or maneuvers.

7.4.1.2.7 Slope

Most roads, rails or other transportation links are designed within limits of slope and, for some vehicles, the slope of the link within this limit has little effect on any cost function. On the other hand, large commercial freight vehicles may be affected by slope in many ways. One of the most common vehicle-based constraints may be a limit on slope.

7.4.1.2.8 Link capacity

The link capacity is the limit of the link in terms of the traffic it can accommodate. It is dependent on the size of the link, the number of lanes, the sinks (outgoing turns) and sources (incoming turns) associated with the link. The normal assumption is that the closer a link is to its capacity, the slower the traffic moves along it.

7.4.1.2.9 Link volume

This temporal attribute represents the current amount of traffic on the link, given in the same measure as its total capacity.

7.4.1.2.10 Link average volume

This attribute represents the average amount of traffic on the link, given in the same measure as its total capacity.

7.4.1.2.11 Link peak volume

This attribute represents the normal peak amount of traffic on the link, given in the same measure as its total capacity.

7.4.1.2.12 Stopping time

An attribute of a turn that describes the expected (or current average) stopping time associated with the turn.

7.4.1.2.13 Conditions

Conditional attributes associated with a link or turn are attributes that may have a negative effect on the capacity of the link. These include, but are not limited to, the following:

— weather,

— special events or unusual occurrences,

— construction zones.

7.4.1.2.14 Tolls

The application of tolls to particular decision points (nodes in the Dijkstra graph) is a very culturally dependent issue. Toll values (usually as a marginal cost) should be incorporated in cost functions at the point where the decision to incur that toll is made irreversible. In the USA, the most common point is at exits, where the decision to progress to the next exit is made. At this point, the difference between the tolls paid at the two exits is added as a marginal cost to the trip (assuming the toll is counted towards the selected cost function). In the presence of non-toll alternatives, tolls alone cannot be used in the optimization algorithms. The existence of "zero-cost" cycles (closed curves, see ISO 19107) will potentially put such algorithms into infinite loops.

7.4.1.3 Simple cost functions

7.4.1.3.1 Basic

A "basic" cost function is one of the several static (non-varying with time) cost functions that are commonly used in routing. The most common are time and distance.

The simplest cost function is distance. It is static to a large extent, changing only when the network is physically modified. Routing based on distance is valid essentially for the life span of the current version of the network. It does not require contributions from the nodes. For node-to-node routes, each link shall have a total length. For position-to-position routes, where partial links are used, the distance from any point on a link to the end shall be calculable (as required in ISO 19107). Given this ability, the accuracy of distance functions is as good as the data accuracy will allow.

The next simplest function is time of travel. For short links, it is reasonable to assume that average speed along the link is probably close enough to maintain accurate time estimates, and assure optimality of the route finally chosen. For longer routes, it may be necessary to maintain a link measure that gives either average time to location or average speed at location (either can be derived from the other).

Travel time can also take into consideration the physical aspects of the route and the limitations of the vehicle.

Either of these two functions can approximate actual costs to some degree of assurance.

7.4.1.3.2 Predictive

A "predictive" cost function is a non-static (dynamic) cost function that does not use real-time data. General averages that are dependent on the time of the day, the type of day, the time of year and any other predictable factor, are used. Since real-time data is not required, the predictive cost function can be used at any time and will return the same optimal route (up to the capability of the algorithm to do so) regardless of when it is invoked. The most common such cost functions are time and distance.

This type of calculation of cost functions brings into account the time of traversal. For these cost functions, either the source or destination target position shall have a time constraint. Using this time constraint, the routing routine can predict at what time any particular point on the route would be traversed. Given time sensitive variables, the cost of traversal can calculated based on the time of traversal. Because of the nature of the process, the first pass at calculation of such a route will use variables based on past observations, experience or some form of predictive model. The most common time considerations are:

— time of day;

— type of day (weekday, weekend, holiday);

— particular date (day of the week, actual date, holiday);

— average traffic conditions on the route given the time traversal.

7.4.1.3.3 Dynamic

The dynamic cost functions take into account real-time data that may affect the total cost of a route, and hence the currently optimal route. In a dynamic service, the service provider is expected to recalculate optimal routes in a manner sufficient to ensure that the vehicle can be rerouted if the optimal route for the current vehicle position changes in some significant way.

Dynamic cost functions shall be linked to real-time or near real-time sources of information about conditions within the network that can affect travel time, such as traffic conditions, weather conditions, and special circumstances (police, fire, accident, special events). Beginning with a predictive route, and tracking the vehicle along the route, a dynamic service will recalculate the cost function for the remainder of the route, and re-optimize if this cost function value changes beyond a given limit (usually a percentage of remaining cost). Obviously, a dynamic cost function requires direct and reasonably constant contact with the vehicle.

7.4.2 NS_CostFunction

7.4.2.1 Semantics

The type “NS_CostFunction” is used to represent functions for the calculation of costs associated with the traversal of the various parts of a network. The UML for NS_CostFunction is given in Figure 53.

7.4.2.2 Role: turn : NS_CostedTurn

The association role “turn” aggregates the costed turn for this cost function.

```
NS_CostFunction :: turn : NS_CostedTurn
```

7.4.2.3 Role: link : NS_CostedLink

The association role “link” aggregates the costed link for this cost function.

```
NS_CostFunction :: link : NS_CostedLink
```

7.4.2.4 Attribute: description : CharacterString

The attribute “description” contains a general description of the cost function.

```
NS_CostFunction :: description : CharacterString
```

7.4.2.5 Attribute: formula[0..1] : CharacterString

The optional attribute “formula” contains algebraic formulae describing this cost function.

```
NS_CostFunction :: formula[0..1] : CharacterString
```

7.4.2.6 Operation: cost

The various operations called “cost” assign a cost to various types of components of routes or to entire routes. For dynamic cost functions, the variable “t” indicates the date and time for the evaluation of cost.

```
NS_CostFunction :: cost(
      turn : NT_Turn, t[0..1] : TM_DateAndTime = "now") : Measure
NS_CostFunction :: cost(
      link : NT_Link, t[0..1] : TM_DateAndTime = "now") : Measure
NS_CostFunction :: cost(
      route : NT_Route, t[0..1] : TM_DateAndTime = "now") : Measure
NS_CostFunction :: cost(
      maneuver : NT_Maneuver, t[0..1] : TM_DateAndTime = "now") : Measure
```

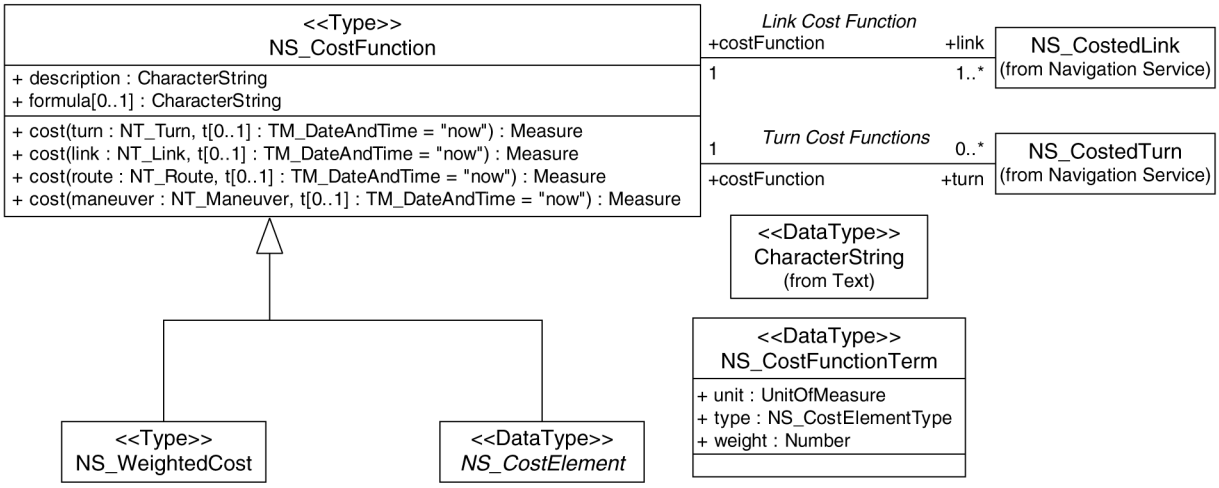

Figure 53 — Context Diagram: NS_CostFunction

7.4.3 NS_CostElements

The data type “NS_CostElement” is the root of the collection of data types used to define cost functions. The UML for NS_CostElements is given in Figure 54.

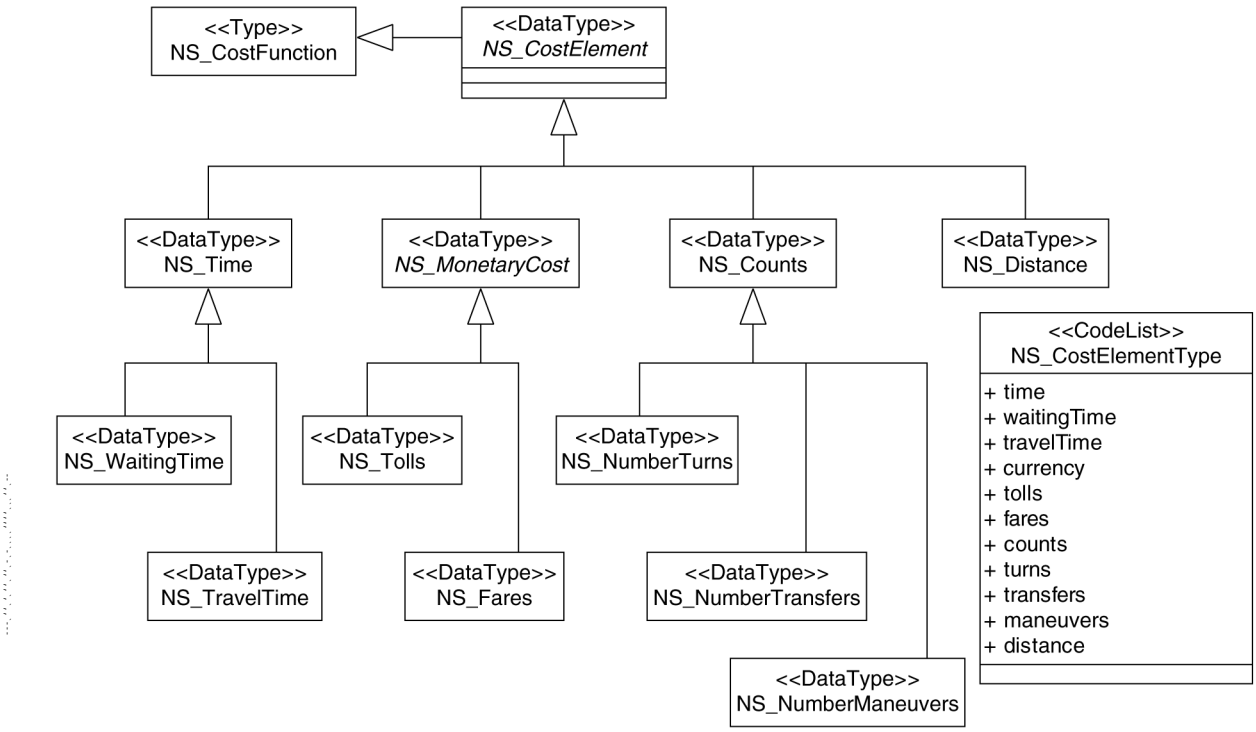

Figure 54 — Context Diagram: NS_CostElements

7.4.4 NS_MonetaryCost

7.4.4.1 Semantics

The data type “NS_MonetaryCost” is the root class for all types that have a monetary-valued cost. The UML for NS_MonetaryCost is given in Figure 55.

7.4.4.2 Operation: cost

The various operations called “cost” are overridden so that their return type is a type of currency.

```
NS_MonetaryCost :: cost(
      turn : NT_Turn, t[0..1] : TM_DateAndTime = "now") : Currency
NS_MonetaryCost :: cost(
      link : NT_Link, t[0..1] : TM_DateAndTime = "now") : Currency
NS_MonetaryCost :: cost(
      route : NT_Route, t[0..1] : TM_DateAndTime = "now") : Currency
NS_MonetaryCost :: cost(
      maneuver : NT_Maneuver, t[0..1] : TM_DateAndTime = "now") : Currency
```

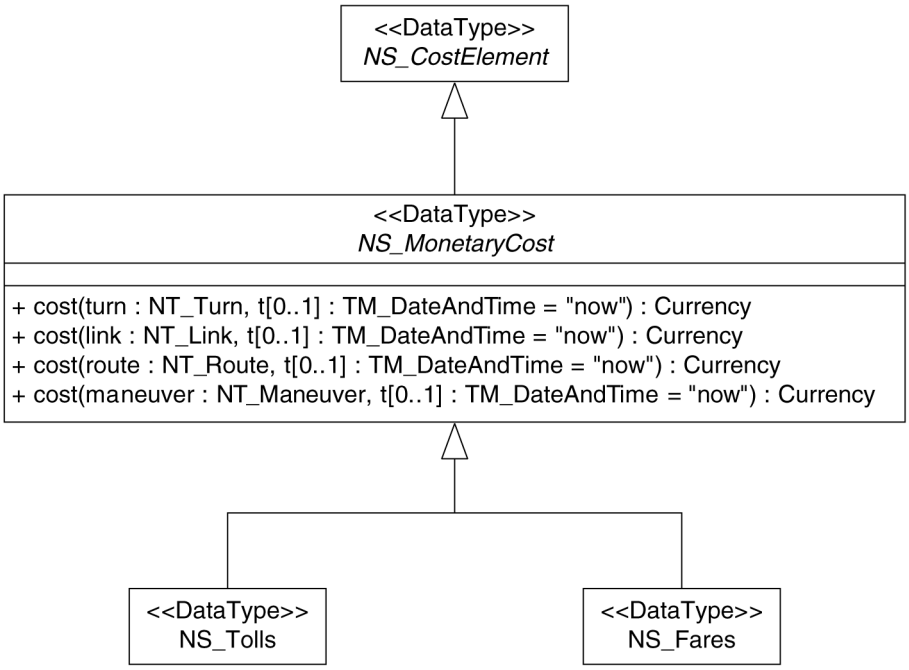

Figure 55 — Context Diagram: NS_MonetaryCost

7.4.5 NS_Tolls

The data type “NS_Tolls” is used for monetary cost arising from the payment of tolls or other use fees. The UML for NS_Tolls is given in Figure 56.

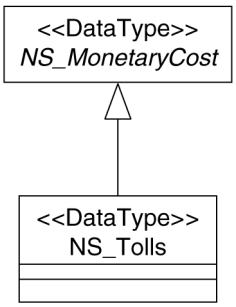

Figure 56 — Context Diagram: NS_Tolls

7.4.6 NS_Fares

The data type “NS_ Fares” is used for monetary cost arising from the payment of fares (such as to a ferry) or other transport fees. The UML for NS_Fares is given in Figure 57.

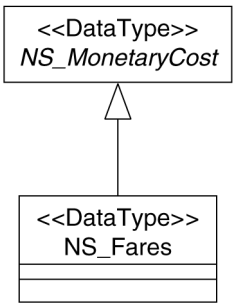

Figure 57 — Context Diagram: NS_Fares

7.4.7 NS_Time

7.4.7.1 Semantics

The data type “NS_ Time” is the root class for all types that have a time-valued cost. The UML for NS_Time is given in Figure 58.

7.4.7.2 Operation: cost

The various operations called “cost” are overridden so that their return type is a type of time.

```
NS_Time :: cost(turn : NT_Turn, t[0..1] : TM_DateAndTime = "now") : Time
NS_Time :: cost(link : NT_Link, t[0..1] : TM_DateAndTime = "now") : Time
NS_Time :: cost(route : NT_Route, t[0..1] : TM_DateAndTime = "now") : Time
NS_Time :: cost(maneuver : NT_Maneuver, t[0..1] : TM_DateAndTime = "now") :
      Time
```

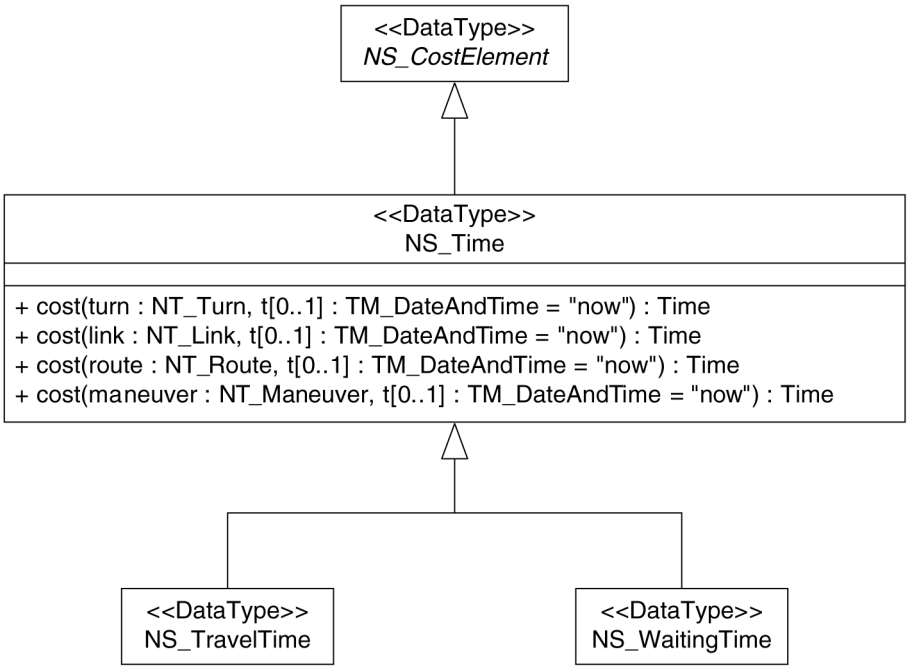

Figure 58 — Context Diagram: NS_Time

7.4.8 NS_TravelTime

The data type “NS_TravelTime” is used for time cost arising from time expended in travel along the route. The UML for NS_TravelTime is given in Figure 59.

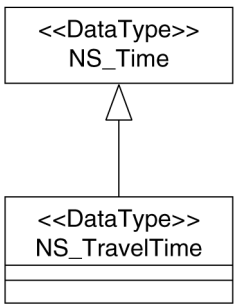

Figure 59 — Context Diagram: NS_TravelTime

7.4.9 NS_WaitingTime

The data type “NS_WaitingTime” is used for time cost arising from time expended waiting or stopped at points along the route. The UML for NS_WaitingTime is given in Figure 60.

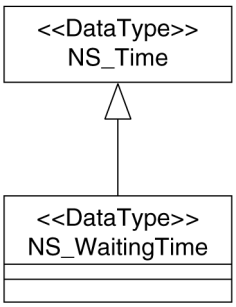

Figure 60 — Context Diagram: NS_WaitingTime

7.4.10 NS_Counts

7.4.10.1 Semantics

The data type “NS_ Counts” is the root class for all types that have a count-valued cost. The UML for NS_Counts is given in Figure 61.

7.4.10.2 Operation: cost

The various operations called “cost” are overridden so that their return type is a type of time.

```
NS_Counts :: cost(
      turn : NT_Turn, t[0..1] : TM_DateAndTime = "now") : Integer
NS_Counts :: cost(
      link : NT_Link, t[0..1] : TM_DateAndTime = "now") : Integer
NS_Counts :: cost(
      route : NT_Route, t[0..1] : TM_DateAndTime = "now") : Integer
NS_Counts :: cost(
      maneuver : NT_Maneuver, t[0..1] : TM_DateAndTime = "now") : Integer
```

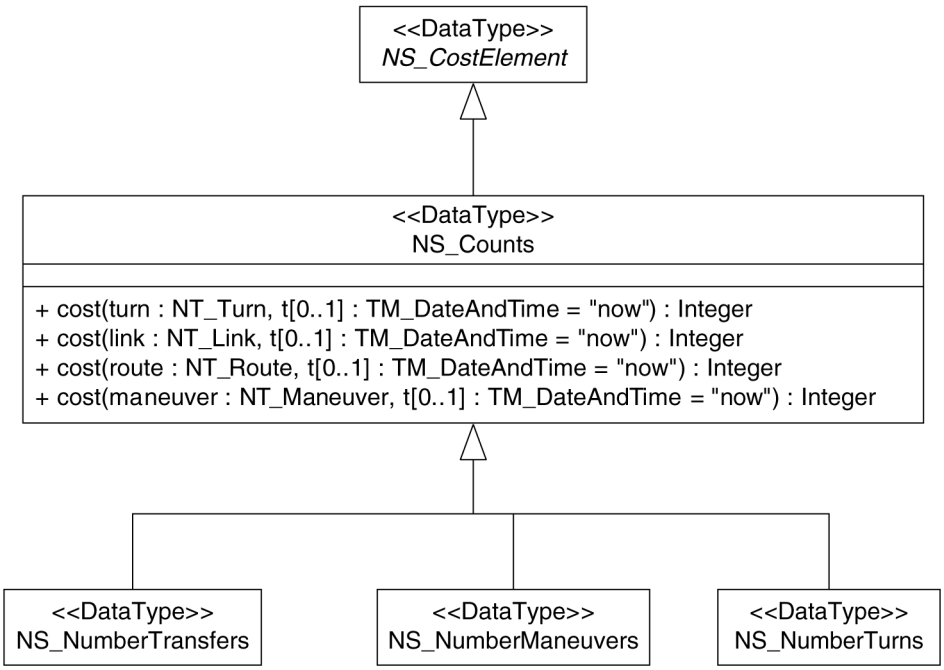

Figure 61 — Context Diagram: NS_Counts

7.4.11 NS_NumberManeuvers

The data type “NS_NumberManeuvers” is the cost element used for counting maneuvers. Normally its value on a maneuver should be 1, and 0 elsewhere, unless “in-link” maneuvers (such as lane changes) are counted. The UML for NS_NumberManeuvers is given in Figure 62.

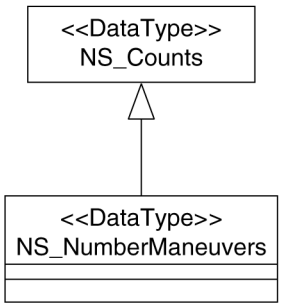

Figure 62 — Context Diagram: NS_NumberManeuvers

7.4.12 NS_NumberTurns

The data type “NS_NumberTurns” is the cost element used for counting turns. Normally its value on a turn should be 1, unless “straight” turns are not counted. The UML for NS_NumberTurns is given in Figure 63.

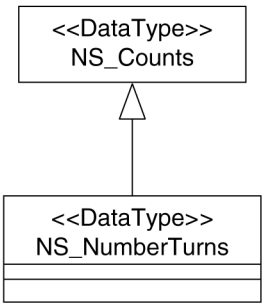

Figure 63 — Context Diagram: NS_NumberTurns

7.4.13 NS_NumberTransfers

The data type “NS_NumberTransfers” is the cost element used for counting transfers. Transfers could include mode changes (see ISO 19134) or changes of network or link type such as from a road to a ferry. The UML for NS_NumberTransfers is given in Figure 64.

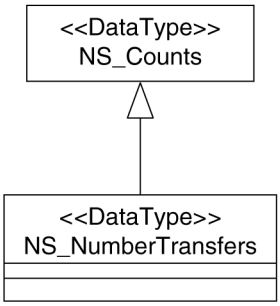

Figure 64 — Context Diagram: NS_NumberTransfers

7.4.14 NS_Distance

7.4.14.1 Semantics

The data type “NS_ Distance” is the root class for all types that have a distance-valued cost. The UML for NS_Distance is given in Figure 65.

7.4.14.2 Operation: cost

The various operations called “cost” are overridden so that their return type is a type of time.

```
NS_Distance :: cost(
      turn : NT_Turn, t[0..1] : TM_DateAndTime = "now") : Distance
NS_Distance :: cost(
      link : NT_Link, t[0..1] : TM_DateAndTime = "now") : Distance
NS_Distance :: cost(
      route : NT_Route, t[0..1] : TM_DateAndTime = "now") : Distance
NS_Distance :: cost(
      maneuver : NT_Maneuver, t[0..1] : TM_DateAndTime = "now") : Distance
```

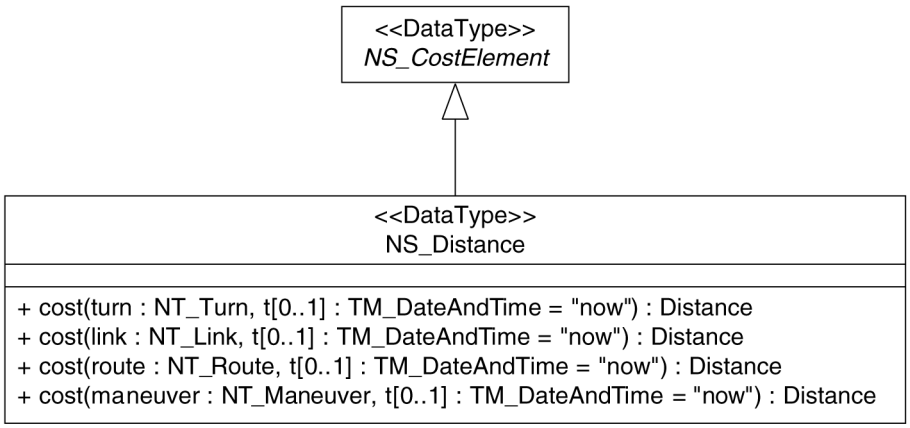

Figure 65 — Context Diagram: NS_Distance

7.4.15 NS_WeightedCost

7.4.15.1 Semantics

The type “NS_WeightedCost” is a cost function that is the sum of the numeric values of other cost functions with weights. These weights are used to put the various types of cost on a common basis. Since this is dependent on personal preferences, the weights will most often be dependent on a traveller’s profile. The handling of such personalization is in the domain of the implementation. This International Standard will not discuss the determination of weights, but will discuss how such cost functions are used once defined. The UML for NS_WeightedCost is given in Figure 66.

7.4.15.2 Role: term : NS_CostFunctionTerm

The association role “term” aggregates the various weighted terms of this cost function.

```
NS_WeightedCost :: term : NS_CostFunctionTerm
```

7.4.15.3 Attribute: targetUnit[0..1] : UnitOfMeasure

The optional attribute “targetUnit” defines the units to be associated with the numeric answer of this cost function. For example, if the cost function were designed to put all costs on a monetary basis, then the target unit would be a monetary type (such as the US dollar). This is a common practice based on “time is money” metaphors.

```
NS_WeightedCost :: targetUnit[0..1] : UnitOfMeasure
```

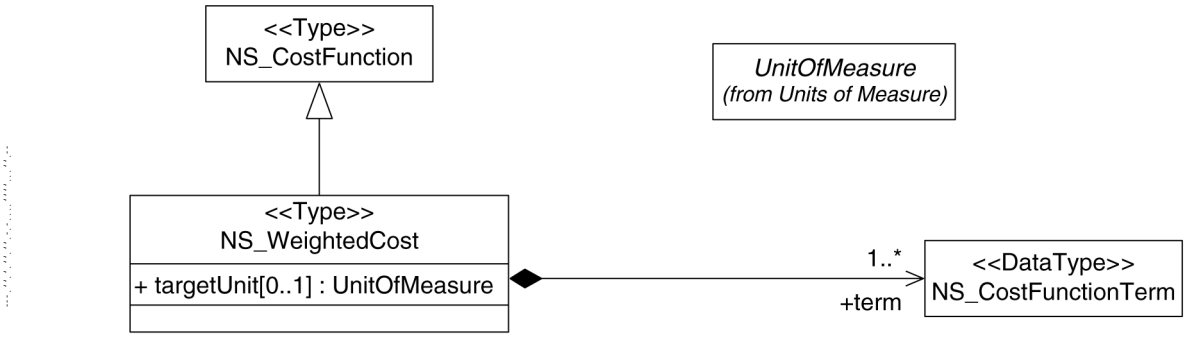

Figure 66 — Context Diagram: NS_WeightedCost

7.4.16 NS_CostFunctionTerm

7.4.16.1 Semantics

The data type "NS_CostFunctionTerm" is a component of a cost function that is the sum of the numeric values of other cost functions with weights. The UML for NS_CostFunctionTerm is given in Figure 67.

7.4.16.2 Attribute: unit : UnitOfMeasure

The attribute "unit" is the unit for the term that corresponds to the conversion weight. All costs of the type listed are converted to this unit of measure before the weight is applied.

```
NS_CostFunctionTerm :: unit : UnitOfMeasure
```

7.4.16.3 Attribute: type : NS_CostElementType

The attribute "type" is the type of cost for the term.

```
NS_CostFunctionTerm :: type : NS_CostElementType
```

7.4.16.4 Attribute: weight : Number

The attribute "weight" is the weight for the term that will be used to multiply the numeric value of the cost (in the units indicated) to get a weighted contribution for this cost function.

```
NS_CostFunctionTerm :: weight : Number
```

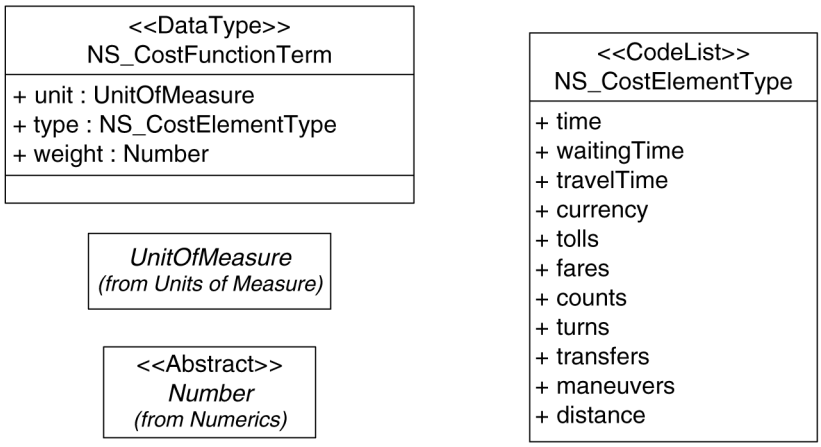

Figure 67 — Context Diagram: NS_CostFunctionTerm

7.4.17 NS_CostElementType

The code list "NS_CostElementType" enumerates the types of cost function elements supported. The initial values of this list are "time", "waitingTime", "travelTime", "currency", "tolls", "fares", "counts", "turns", "transfers", "maneuvers", and "distance". The UML for NS_CostElementType is given in Figure 67.

7.5 Package: Preferences

7.5.1 Semantics

The package "Preferences" contains classes and code list for the specification of traveller preference that are outside of the cost function. Such preference will be used to modify the network used before the cost function is minimized. As such, some of these modifications will be heuristic in nature, and are therefore the domain of the implementation.

7.5.2 NS_RoutePreferences

The code list "NS_RoutePreferences" specifies the types of route segment that the traveller prefers. The initial values are "scenic", "easiest", "majorRoadsOnly" and "avoidMajorRoads". The "scenic" preference is difficult to quantify (how much scenery is worth how much time), and taken to extreme could create anomalous results (such as all routes going through the Grand Canyon, even if they are from New York to Boston). The "easiest" preference could probably be quantified in the cost function by grading the maneuvers and links by their "difficulty". The "majorRoadsOnly" preference produces a route that uses major roads if at all possible (again doable by a modification to the cost function that penalizes minor roads). The "avoidMajorHighways" does the opposite by penalizing major roads. The UML for NS_RoutePreferences is given in Figure 68.

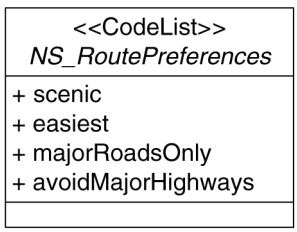

Figure 68 — Context Diagram: NS_RoutePreferences

7.5.3 NS_AvoidList

7.5.3.1 Semantics

The type “NS_AvoidList” specifies a list of places that the traveller wishes to avoid during his travels. For example, if the avoid list for a route from Boston to Washington, DC, were to carry “greater New York” on its avoid list, the candidate routes would have to avoid Manhattan and its surrounding urban areas. The UML for NS_AvoidList is given in Figure 69.

7.5.3.2 Role: avoidFeature : FD_Feature

The association role “avoidFeature” aggregates individual features to be avoided in candidate routes.

```
NS_AvoidList  ::  avoidFeature : FD_Feature
```

7.5.3.3 avoidFeatureType : GF_FeatureType

The association role “avoidFeatureType” aggregates entire feature types to be avoided in candidate routes.

```
NS_AvoidList  ::  avoidFeatureType : GF_FeatureType
```

7.5.3.4 Attribute: pointElement[0..*] : NT_WayPoint

The attribute “pointElement” is a list of waypoints to be avoided.

```
NS_AvoidList  ::  pointElement[0..*] : NT_WayPoint
```

7.5.3.5 Attribute: mode[0..*] : CharacterString

The attribute “mode” is a list of travel modes to be avoided. The most common entry in this list would be link types.

```
NS_AvoidList  ::  mode[0..*] : CharacterString
```

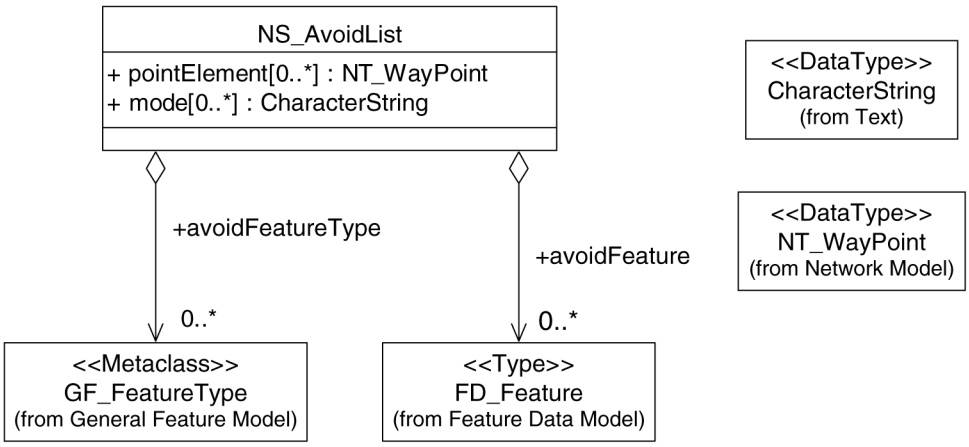

Figure 69 — Context Diagram: NS_AvoidList

8 Address Model

8.1 Semantics

Addresses are the most common manner in which locations are transmitted to navigation services for vehicles. Unfortunately, there is not an applicable current international address standard, and address formats vary from country to country and from culture to culture. For this reason, this International Standard adopts a design strategy very loosely based on the Electronic Commerce Code Management Association (ECCMA) International Address Element Code (IAEC). The basis for this strategy is a set of simple assumptions.

— Addresses are made up of a sequence of text elements with well-known semantic content (valid because of the use of ISO 11180:1993, *Postal addressing*).

— These elements are not universal to all countries or cultures, but there is a reasonable collection of such elements that can be gathered to cover all situations.

— Address elements are used and represented differently in different countries and cultures, but these local formats can be captured in “style templates” that describe how the elements of the national address can be placed in an ISO 11180:1993 compliant postal address.

This package contains two leaf packages that describe a tentative model for a beginning set of address elements, generic addresses consisting of aggregations of those elements (applicable to contributing member countries), and a description of the required system for maintaining this part of the standard.

The components of the address model are two leaf packages: one (Address) for the overall address model as an aggregate of elements, and another (Address Elements) for the address element model. This structure is depicted in Figure 70.

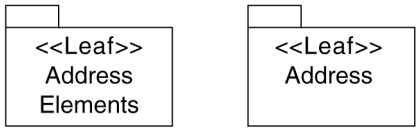

Figure 70 — Leaf packages of the Address Model

8.2 Package: Address

8.2.1 Semantics

The Address package contains data types based on postal street address suitable for use as location input to location-based functions. The classes consist of an abstract superclass for all addresses (AD_AbstractAddress), a generic class for address (AD_Address) that will fit the information content of any instance of the abstract class, and a set of national address examples. All the component members of these classes are address elements and are defined in the next package. This structure is depicted in Figure 71.

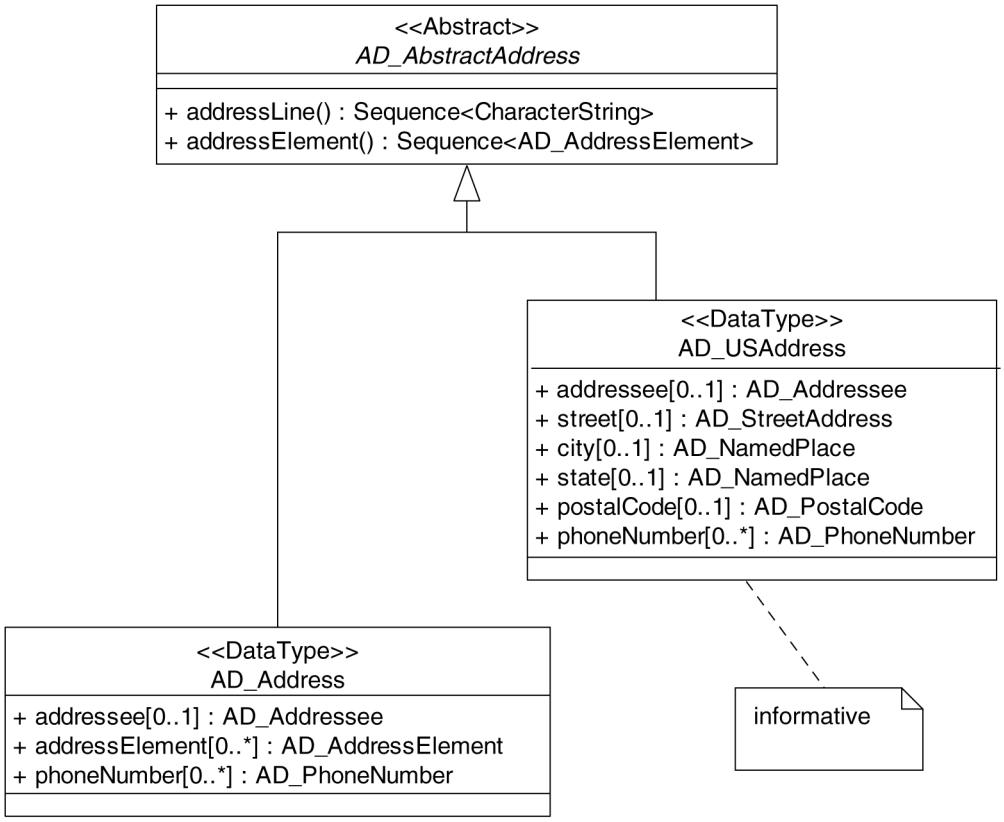

Figure 71 — Basic Address classes

8.2.2 AD_Address

8.2.2.1 Semantics

The datatype “AD_Address” is a concrete subclass of AD_AbstractAddress, whose data structure is consistent with any applicable subtype of abstract address. Instances of AD_Address consist of any number of AD_Address elements. The UML for AD_Address is given in Figure 72.

8.2.2.2 Attribute: addressee[0..1] : AD_Addressee

The attribute “addressee” is the contact person or designated recipient of items using this address. Non-postal addresses, which are common in navigation locations, would normally not have a specific addressee.

```
AD_Address :: addressee[0..1] : AD_Addressee
```

8.2.2.3 Attribute: addressElement[0..*] : AD_AddressElement

Any address compliant with this International Standard will be representable as a sequence of address elements. The attribute “addressElement” will be any number of instances of subtypes of AD_AddressElements. Normally, each address element will correspond to a single, complete line in the address template.

```
AD_Address :: addressElement [0..*] : AD_AddressElement
```

8.2.2.4 Attribute: phoneNumber[0..*] : AD_PhoneNumber

Since AD_PhoneNumber is an address element, the “addressElement” attribute is sufficient to allow this to be included in the address. Since phone numbers are of particular importance in mobile subscriber network applications, the attribute “phoneNumber” makes this explicit.

```
AD_Address :: phoneNumber[0..*] : AD_PhoneNumber
```

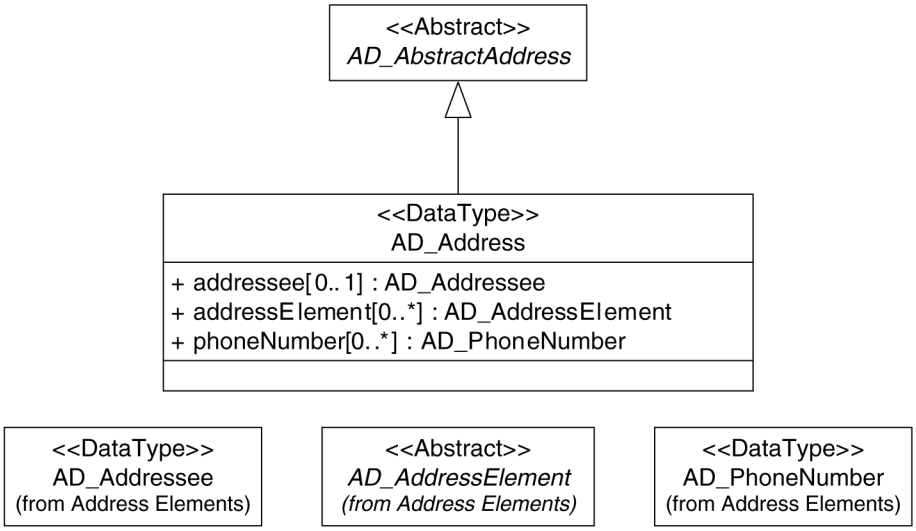

Figure 72 — Context Diagram: AD_Address

8.2.3 AD_AbstractAddress

The class “AD_AbstractAddress”, an empty abstract class, is the root of the address-subclassing tree. The design strategy is to define “address elements” commonly used across multiple cultures and countries, and to allow any country to define an address type of its own by combinations of these elements. New elements are allowed if they have character string representations. The printed address for any country would be a template of the strings specified in its subtype of address. Note that any viable instance for any subtype of abstract address would also be a viable AD_Address, as long as all of its address elements are defined. To allow for total freedom of specification, this class has no members. Its use as a member of any class or type in this specification would most likely be replaced by a country-specific subclass in implementation profiles. The UML for AD_AbstractAddress is given in Figure 73.

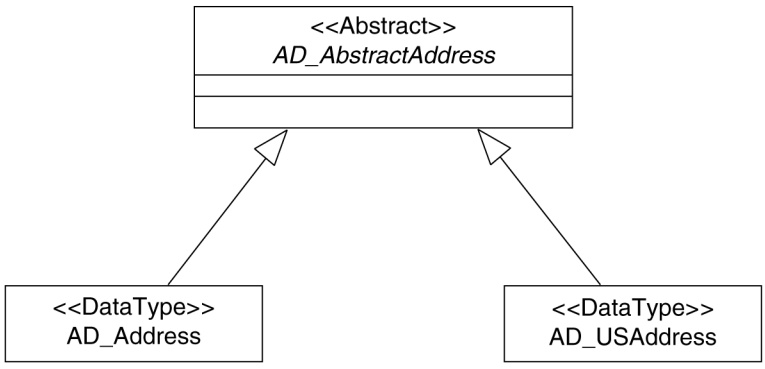

Figure 73 — Context Diagram: AD_AbstractAddress

8.2.4 AD_USAddress

8.2.4.1 Semantics

The data type “AD_USAddress” is supplied in this International Standard as an example of how a country specific subclass of AD_AbstractAddress might be formed. It is meant to reflect a standard United States street address, but specification of such is the purview of a US profile of this International Standard. As such, this class definition is informative. This is a “street” address (as opposed to postal boxes which are not normally associated with street locations). The UML for AD_USAddress is given in Figure 74.

8.2.4.2 Attribute: addressee[0..1] : AD_Addressee

The attribute “addressee” is the contact person or designated recipient of items using this address. Non-postal addresses, which are common in navigation locations, would normally not have a specific addressee.

```
AD_Address :: addressee[0..1] : AD_Addressee
```

8.2.4.3 Attribute: street[0..1] : AD_StreetAddress

The attribute “street” contains the street address of the AD_USAddress.

```
AD_USAddress :: street[0..1] : AD_StreetAddress
```

8.2.4.4 Attribute: city[0..1] : AD_NamedPlace

The attribute “city” contains the city, town, village or borough element of the AD_USAddress.

```
AD_USAddress :: city[0..1] : AD_NamedPlace
```

8.2.4.5 Attribute: state[0..1] : AD_NamedPlace

The attribute “state” contains the state element of the AD_USAddress.

```
AD_USAddress :: state[0..1] : AD_NamedPlace
```

8.2.4.6 Attribute: postalCode[0..1] : AD_PostalCode

The attribute “postalCode” contains the “ZIP code” element of the AD_USAddress. This is a 5-digit number in the US. Within the AD_PostalCode, the “addonCode” is the extended “Postal Code” containing the “ZIP+4 code” extension. This is a 4-digit number.

```
AD_USAddress :: postalCode[0..1] : AD_PostalCode
```

8.2.4.7 Attribute: phoneNumber[0..*] : AD_PhoneNumber

The attribute “phoneNumber” contains the phone number element of the AD_USAddress. In the US, this would be a character string of the form “+1 (nnn) nnn-nnnn”, where “n” is any numeric character consistent with its position (area codes and exchanges never begin with “0” for example).

NOTE Historically, the US phone number consists, in order, of a 3-digit area code, a 3-digit exchange, and a 4-digit local number. The meaning of these subcomponents is generally being lost, especially with regard to mobile devices [(cell phones, pagers, personal digital assistants (PDA)].

```
AD_USAddress :: phoneNumber[0..*] : AD_PhoneNumber
```

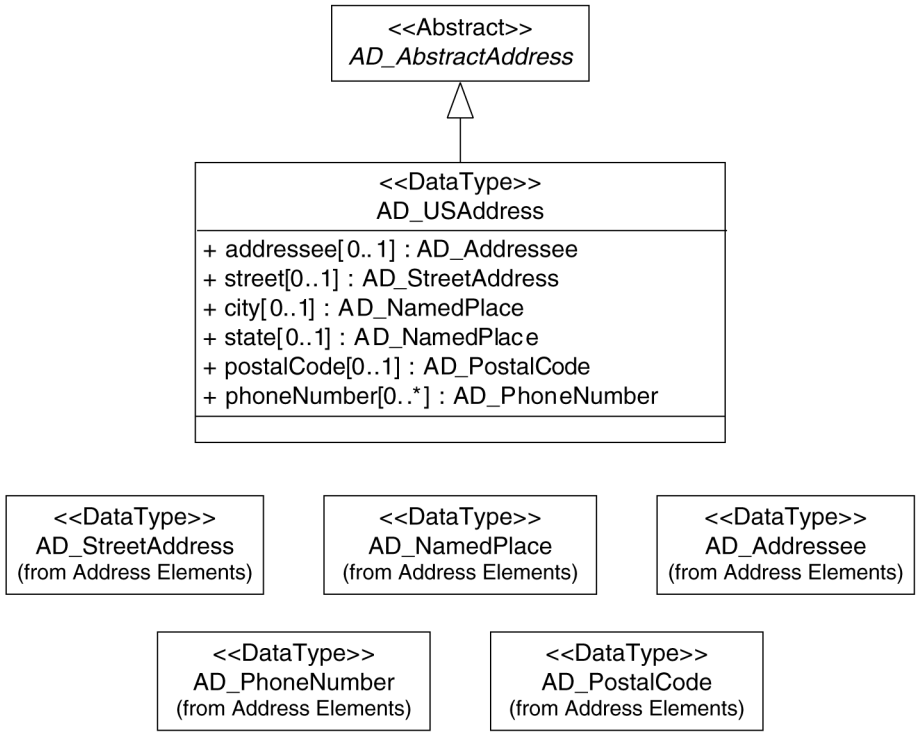

Figure 74 — Context Diagram: AD_USAddress

8.3 Package: Address Elements

8.3.1 Semantics

Address elements are components of address, associated with a single line or less of the textual street address of a location.

8.3.2 AD_AddressElement

The abstract class “AD_AddressElement” is the root class of all elements, and allows the address package to define a generic address type by aggregating any number of elements. The UML for AD_AddressElement is given in Figure 75.

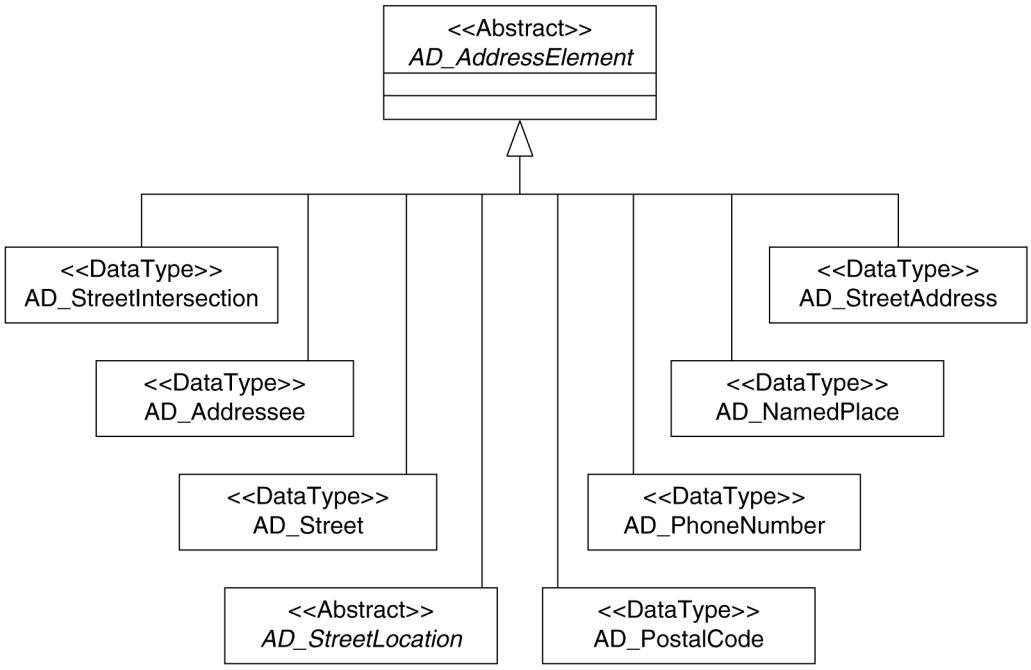

Figure 75 — Context Diagram: AD_AddressElement

8.3.3 AD_Addressee

8.3.3.1 Semantics

The data type “AD_Addressee” in an address specifies the recipient of any message sent to this address. In a navigation application, addressee would not normally be needed, as it is not part of most specific locations. If the navigation system is associated with directories (such as “yellow pages” (business directory) or “white pages” (non-business directory)) then the addressee may be all that is needed. The UML for AD_Addressee is given in Figure 76.

8.3.3.2 Attribute: name : CharacterString

The attribute “name” is the addressee’s name as a character string.

```
AD_Addressee :: name : CharacterString
```

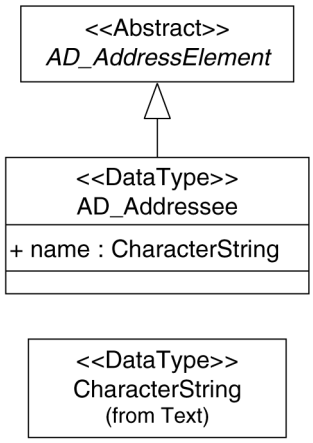

Figure 76 — Context Diagram: AD_Addressee

8.3.4 AD_StreetIntersection

8.3.4.1 Semantics

The data type “AD_StreetIntersection” is an alternative to addressing. It specifies the point where the component streets intersect (normally one unique point). Some areas accept intersection specifications as postal addresses, and intersections are often useful in cases where address information is unknown. The UML for AD_StreetIntersection is given in Figure 77.

8.3.4.2 Attribute: streets[0..*] : AD_Street

The attribute “streets” contains specifications for each of the streets in the intersection. There are normally only two streets required, but other cardinalities are encountered.

```
AD_StreetIntersection :: streets[0..*] : AD_Street
```

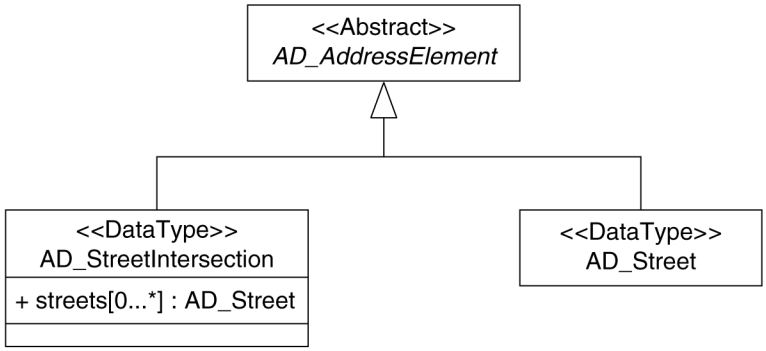

Figure 77 — Context Diagram: AD_StreetIntersection

8.3.5 AD_Street

8.3.5.1 Semantics

The datatype “AD_Street” specifies a street by “name”. The UML for AD_Street is given in Figure 78.

8.3.5.2 Attribute: directionalPrefix[0..1] : CharacterString

The attribute “directionalPrefix” specifies directional information held before the street name. Normal values of this prefix are “north”, “south”, “east” or “west”, their abbreviations, or their local language equivalents.

```
AD_Street :: directionalPrefix[0..1] : CharacterString
```

8.3.5.3 Attribute: typePrefix[0..1] : CharacterString

The attribute “typePrefix” specifies the type of the street. In English, this is usually done as a suffix, but prefixes are not unknown, especially if the name was inherited from another language such as in “Camino Real”, which is Spanish for Royal Road.

```
AD_Street :: typePrefix[0..1] : CharacterString
```

8.3.5.4 Attribute: officialName[0..1] : CharacterString

The attribute “officialName” specifies the actual name of the street.

```
AD_Street :: officialName[0..1] : CharacterString
```

8.3.5.5 Attribute: trailingSpaces[0..1] : Boolean = "TRUE"

The attribute “trailingSpaces” is a Boolean that specifies the use of spacing in a street name.

```
AD_Street :: trailingSpaces [0..1] : CharacterString
```

8.3.5.6 Attribute: typeSuffix[0..1] : CharacterString

The attribute “typeSuffix” specifies the type of the street. This is the same information as the “typePrefix” but the formatting of the final character string name is affected depending on which attribute is used.

```
AD_Street :: typeSuffix [0..1] : CharacterString
```

8.3.5.7 Attribute: directionalSuffix[0..1] : CharacterString

The attribute “directionalSuffix” specifies directional information held after the street name. Normal values of this prefix are “north”, “south”, “east” or “west”, their abbreviations, or their local language equivalents. This is the same information as the “directionalPrefix” but the formatting of the final character string name is affected depending on which attribute is used.

```
AD_Street :: directionalSufffix [0..1] : CharacterString
```

8.3.5.8 Attribute: muniQuadrant[0..1] : AD_ MuniQuadrant

The attribute “muniQuadrant” is used in some addresses much like the directional attributes. Here the town is divided into sections based on major east–west and north–south divisions. The effect is as if multiple directionals were used. These are almost always used as suffixes.

```
AD_Street :: muniQuadrant [0..1] : AD_ MuniQuadrant
```

8.3.5.9 Attribute: postalCode[0..1] : AD_PostalCode

The attribute “postalCode” contains the postal code applicable to this street element.

```
AD_Street :: postalCode[0..1] : AD_PostalCode
```

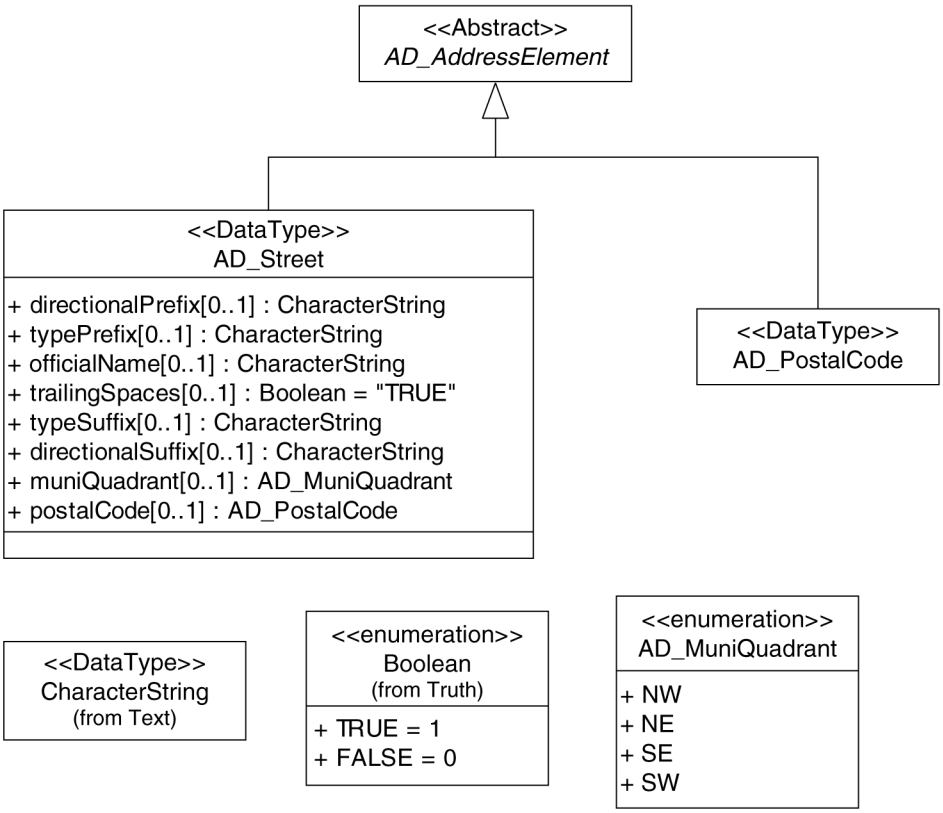

Figure 78 — Context Diagram: AD_Street

8.3.6 AD_PostalCode

8.3.6.1 Semantics

The data type “AD_PostalCode” is used to store the postal code. The UML for AD_PostalCode is given in Figure 79.

8.3.6.2 Attribute: code: CharacterString

The attribute “code” is the primary postal code as a character string.

```
AD_PostalCode :: code: CharacterString
```

8.3.6.3 Attribute: addonCode: CharacterString

The optional attribute “addonCode” is any secondary postal code as a character string.

```
AD_PostalCode :: addonCode[0..1] : CharacterString
```

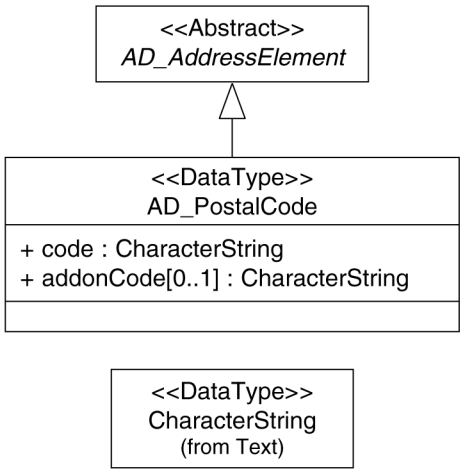

Figure 79 — Context Diagram: AD_PostalCode

8.3.7 AD_StreetLocation

The abstract class “AD_StreetLocation” is the root for any address element that specifies a position on a street. A number is the most common pattern. The UML for AD_StreetLocation is given in Figure 80.

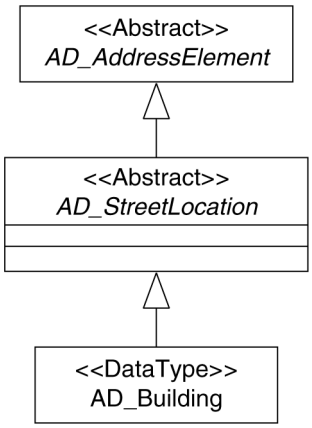

Figure 80 — Context Diagram: AD_StreetLocation

8.3.8 AD_PhoneNumber

8.3.8.1 Semantics

The data type AD_PhoneNumber contains a character string representation of the phone number of the address. The UML for AD_PhoneNumber is given in Figure 81.

8.3.8.2 Attribute: number : CharacterString

The attribute “number” is the phone number.

```
AD_PhoneNumber :: number : CharacterString
```

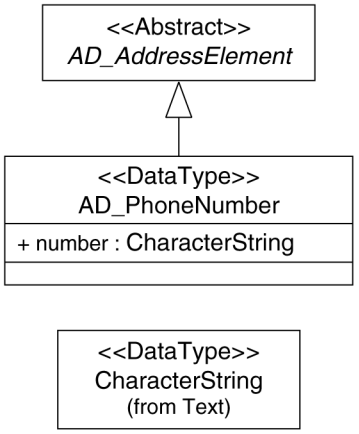

Figure 81 — Context Diagram: AD_PhoneNumber

8.3.9 AD_NamedPlace

8.3.9.1 Semantics

The class “AD_NamedPlace” contains a place name, as might be found in a gazetteer. Place names are usually hierarchical in nature, with larger political units containing smaller ones. The roots of these hierarchies are the countries (or similar political units, such as protectorates, territories or colonies). The UML for AD_NamedPlace is given in Figure 82.

8.3.9.2 Attribute: name : CharacterString

The attribute “name” specifies the name of the place as a character string.

```
AD_NamedPlace :: name : CharacterString
```

8.3.9.3 Attribute: regionOrCountry[0..1] : AD_RegionCode

The optional attribute “regionOrCountry” identifies the country or region of the country in which the place is contained. Implementations are free to use ISO 3166-1, ISO 3166-2 or any other equivalent standardized list.

```
AD_NamedPlace :: regionOrCountry [0..1]: AD_RegionCode
```

8.3.9.4 Attribute: level[0..1] : Integer

The attribute “level” specifies the depth of the place hierarchy where this name belongs.

```
AD_NamedPlace :: level[0..1] : Integer
```

8.3.9.5 Attribute: type : AD_NamedPlaceClassification

The attribute “type” specifies the type of the named place. While related to the level, the types found at a particular level are often a function of the local political organization model. If a named place is not specified as being embedded in a larger place, up to the country level, then it may not represent a unique place and is considered as “out of context”. Such context-free names can normally be placed in context by system assumptions, such as “nearest to client”. Such assumptions are often at the core of location-based services.

```
AD_NamedPlace :: type : AD_NamedPlaceClassification
```

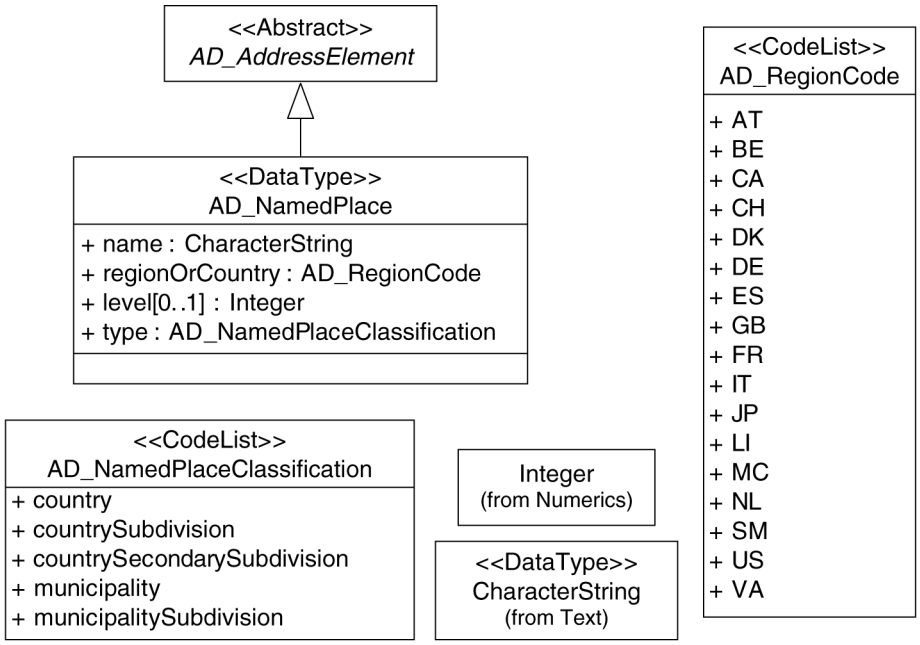

Figure 82 — Context Diagram: AD_NamedPlace

8.3.10 AD_StreetAddress

8.3.10.1 Semantics

The class “AD_StreetAddress” specifies a location by naming a street and a location on that street, usually by numbered address. The UML for AD_StreetAddress is given in Figure 83.

8.3.10.2 Attribute: location : AD_StreetLocation

The attribute “location” specifies the location on the named street. This is normally a “house number” or some other building designation.

```
AD_StreetAddress :: location : AD_StreetLocation
```

8.3.10.3 Attribute: street : AD_Street

The attribute “street” specifies the street involved in this location.

```
AD_StreetAddress :: street : AD_Street
```

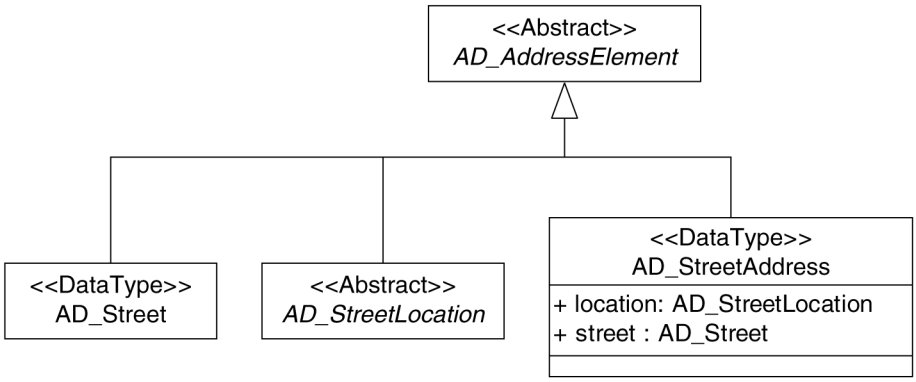

Figure 83 — Context Diagram: AD_StreetAddress

8.3.11 AD_NamedPlaceClassification

The code list "AD_NamedPlaceClassification" specifies the type of named place. Local profiles of this International Standard may extend or modify this code list to reflect the local standard. The initial code list consists of "country", "countrySubdividsion", "countrySecondarySubdivision", "municipality" and "municipalitySubdivision". The UML for AD_NamedPlaceClassification is given in Figure 84.

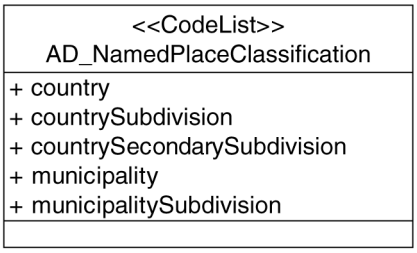

Figure 84 — Context Diagram: AD_NamedPlaceClassification

8.3.12 AD_Building

8.3.12.1 Semantics

The class "AD_Building" is a type of street location. It specifies the location on a street by specifying the building name, or number, with an optional building subdivision name. Normally, one and only one of name and number will be NULL. The UML for AD_Building is given in Figure 85.

8.3.12.2 Attribute: number[0..1] : CharacterString

The optional attribute "number" specifies the number of the building, as it would be in a postal address.

```
AD_Building :: number[0..1] : CharacterString
```

8.3.12.3 Attribute: subdivision[0..1] : AD_NamedPlace

The optional attribute "subdivision" specifies the named-place subdivision within which the building lies.

```
AD_Building :: subdivision[0..1] : AD_NamedPlace
```

8.3.12.4 Attribute: buildingName[0..1] : CharacterString

The optional attribute “buildingName” is the name of the building as a character string.

```
AD_Building :: number[0..1] : CharacterString
```

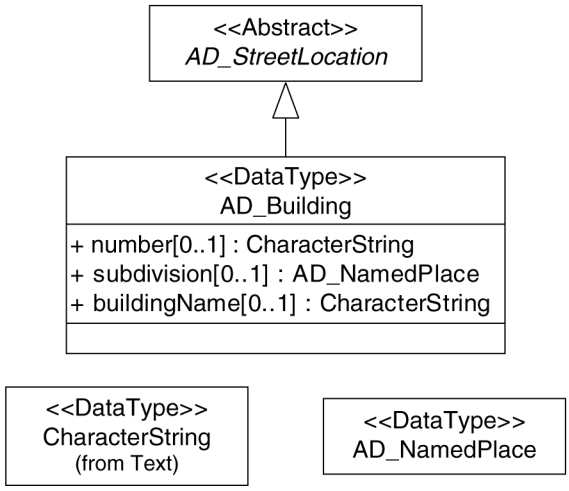

Figure 85 — Context Diagram: AD_Building

8.3.13 AD_MuniQuadrant

The enumeration “AD_MuniQuadrant” is a subdivision mechanism used for many places based on a north–south line and an east–west line which intersect at a politically defined “city center”, thereby dividing the municipality into four quadrants. The enumerated list (in English) is NW, NE, SE, and SW. The UML for AD_MuniQuadrant is given in Figure 86.

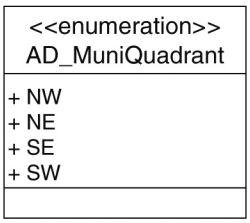

Figure 86 — Context Diagram: AD_MuniQuadrant

8.3.14 AD_RegionCode

The code list “AD_RegionCode” consists of all country codes as defined by ISO 3166-1. Implementations are free to use ISO 3166-1, ISO 3166-2 or any other equivalent standardized list. The UML for AD_RegionCode is given in Figure 87.

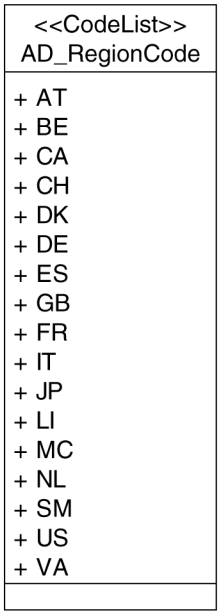

Figure 87 — Context Diagram: AD_RegionCode

8.3.15 AD_NumberRange

8.3.15.1 Semantics

The data type “AD_NumberRange” is used to specify a sequential list of integers, used here to represent address ranges. The interval is assumed to be closed, i.e. it includes the first and last numbers (if present). Fractional addresses are included in the range if both the lower and upper bounds of the fractional number are included. Augmented ranges (extended by a letter or some other suffix), are included if their base number and the next integer is included, thus “221½” and “221B” are treated in the same manner. The UML for AD_NumberRange is given in Figure 88.

8.3.15.2 Attribute: first : Integer, last[0..1] : Integer

The two attributes “first” and “last” are the end points of the range. If the last is not given, the range extends to all numbers larger than “first”.

```
AD_NumberRange :: first : Integer,
AD_NumberRange :: last[0..1] : Integer
```

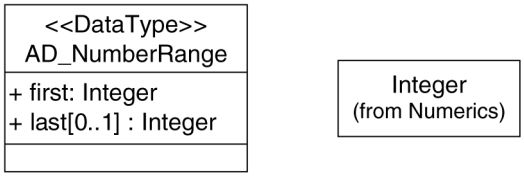

Figure 88 — Context Diagram: AD_NumberRange

8.3.16 AD_ListNamedPlaces

8.3.16.1 Semantics

The data type “AD_ListNamedPlaces” is an ordered list (nested from smallest to largest) of names of named places, each name providing a context for the previous name. For example, “Dupont Circle, Washington, DC, USA” is representable as a 4-long list. The largest place is the country, “USA”. The second is the “District of Columbia” or “DC” as normally abbreviated. The next is “Washington”, which is the name of the city (the only one in DC). The smallest place is the area of the traffic circle called “Dupont Circle” in northwest Washington.

8.3.16.2 Attribute: name[0..9] : AD_NamedPlace

The repeating attribute “name” is the sorted list of named places. The UML for AD_ListNamedPlaces is given in Figure 89.

```
AD_ListNamedPlaces :: name[0..9] : AD_NamedPlace
```

Figure 89 — Context Diagram: AD_ListNamedPlaces

9 Network

9.1 Semantics

The Network package builds on the topological model presented in ISO 19107. It adds information used in specifying location and in navigation.

9.2 Package: Network Model

9.2.1 NT_Network

9.2.1.1 Semantics

An instantiation of the type NT_Network is essentially two separate topologies. Its geometric topology is the value of the association role “geometry”. The second topology is the graph of the NT_Link, NT_Junction, and NT_Turn entities that comprise it. Although the links, junctions and turns have the underlying geometry of the

TP_Complex, they have their own connectivity based on usable “vehicle” routes. If a link comes to a cross-roads and U-turns are allowed, there are up to four turns which exit that link and enter one of the links associated with a directed edge leaving that node, including the one that reversed the incoming link. This microtopology does not have a different geometric existence from the "road” network but represents the potential traffic patterns through each of the “intersections” represented by the TP_Nodes of the associated TP_Complex. The UML for NT_Network is given in Figure 90.

As a topological complex, a Network is dimension 1.

```
NT_Network:
      { complex.dimension() = 1 }
```

9.2.1.2 Role: link : NT_Link

The association role “link” aggregates all the links contained in the network. Links are the instantiation of directed edges for networks.

```
NT_Network :: link: NT_ Link
```

9.2.1.3 Role: element : NT_Junction

The association role “element”, inherited and restricted from TP_Complex, aggregates all the junctions contained in the network. Thus, network nodes are junctions.

```
NT_Network :: element : NT_Junction
```

9.2.1.4 Role: turn : NT_Turn

The association role “turn” aggregates all the turns contained in the network. Turns represent navigable paths through the junctions, and will be composed of the entry link, the junction node, and the exit link.

```
NT_Network :: turn : NT_Turn
```

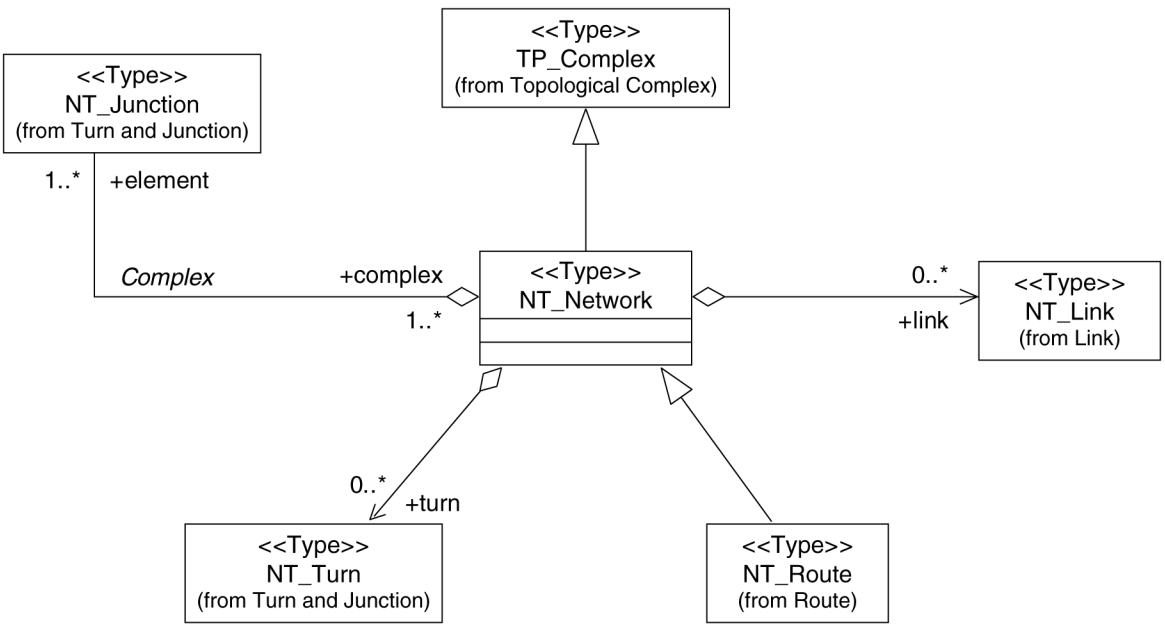

Figure 90 — Context Diagram: NT_Network

9.2.2 NT_WayPoint

9.2.2.1 Semantics

Waypoints are used in route requests to indicate features or positions that shall be either traversed or avoided. The link attribute “avoid” determines which. The UML for NT_WayPoint is given in Figure 91.

9.2.2.2 Attribute: avoid : Boolean

The attribute “avoid” indicates whether or not the waypoint is to be visited or avoided.

```
NT_WayPoint :: avoid : Boolean
```

9.2.2.3 Attribute: position : TK_Position

The attribute “position” specifies the position of the waypoint.

```
NT_WayPoint :: position : TK_Position
```

9.2.2.4 Role: restriction : NT_Constraint

The association role “restriction” aggregates all constraints applicable to this waypoint.

```
NT_WayPoint :: restriction : NT_Constraint
```

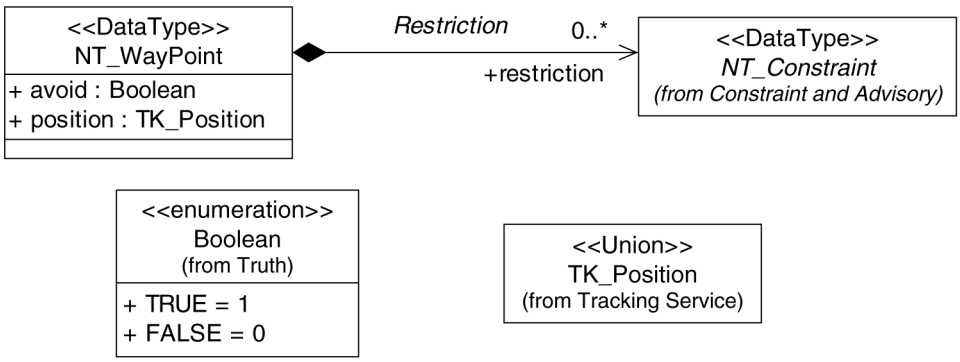

Figure 91 — Context Diagram: NT_WayPoint

9.2.3 NT_WayPointList

9.2.3.1 Semantics

This class is a list of descriptions of waypoints that a route shall either pass through or avoid. The route satisfies the target list if it passes through each location so indicated and avoids each target so indicated. The types of description may vary although most common descriptions are either coordinate positions or street-level addresses. The UML for NT_WayPointList is given in Figure 92.

9.2.3.2 Attribute: startPoint : NT_WayPoint

The attribute “startPoint” indicates the waypoint from which candidate routes shall begin. The “avoid” attribute of the startPoint is ignored (false would not make sense in this usage).

```
startPoint : NT_WayPoint
```

9.2.3.3 Attribute: viaPoint [0..*] : NT_WayPoint

The attribute “viaPoint” is an array of waypoints and indicates the waypoints through which candidate routes shall pass or avoid depending on their “avoid” attribute value.

```
viaPoint[0..*] : NT_WayPoint
```

9.2.3.4 Attribute: endPoint : NT_WayPoint

The attribute “endPoint” indicates the waypoint at which candidate routes shall end. The “avoid” attribute of the endPoint is ignored (false would not make sense in this usage).

```
endPoint : NT_WayPoint
```

9.2.3.5 Attribute: isOrdered : Boolean = TRUE

The attribute “isOrdered” specifies whether or not the order of the viaPoint list is part of the route requirements.

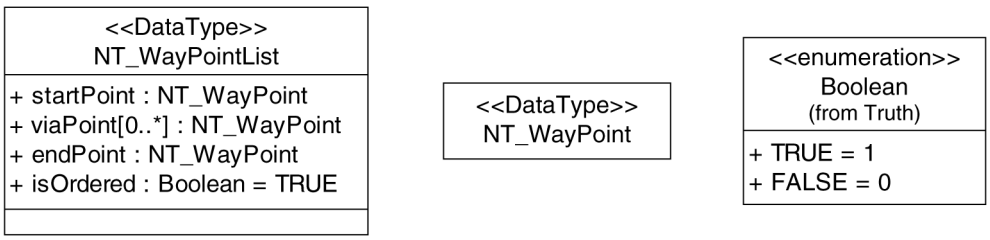

Figure 92 — Context Diagram: NT_WayPointList

9.3 Package: Turn and Junction

9.3.1 Semantics

The package “Turn and Junction” specifies the mechanisms required for the navigation of nodes in a network or route. The UML for turns and junctions is given in Figure 93.

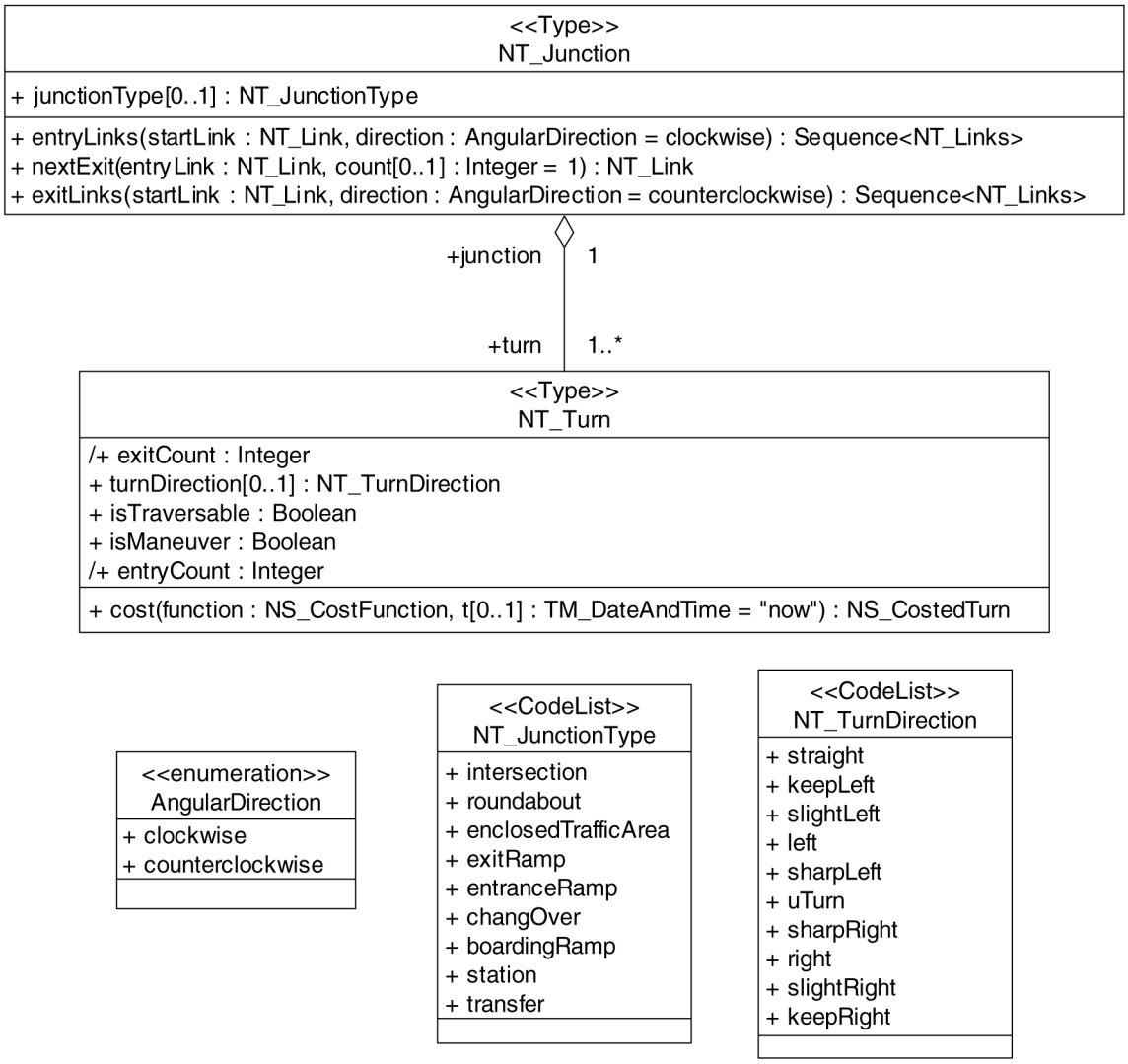

Figure 93 — Junction and turns

9.3.2 NT_Turn

9.3.2.1 Semantics

The NT_Turn type represents a mechanism for traversing from one link to another. Each turn will be located at a node of the underlying topology and will form the bridge for a link entering the node and a link exiting the node. The use of turns allows a restricted network (one with constraints) to be represented by an unrestricted one. The use of the word “turn” does not necessarily reflect a change of direction. Going straight through an intersection is one such viable option and would be represented by an “NT_Turn”. In public transport such as

a rail system the “NT_Turns” are routes through “stations” connecting one incoming “train” with one outgoing “train”. NT_Turns are thus called “connections” when speaking of airports and have temporal constraints based on the availability of the “links”, i.e. on flight times. The UML for NT_Turn is given in Figure 94.

9.3.2.2 Attribute: exitCount : Integer

The attribute “exitCount” indicates how far around the intersection the exit link occurs relative to the entrance link. In right-hand lane systems the count is counter-clockwise. In left-hand systems the count is clockwise. In non-road systems, the number can often be ignored. Like many of the attributes of turn, the exit count is mainly to aid in the creation of instructions for navigation.

```
NT_Turn :: exitCount : Integer
```

9.3.2.3 Attribute: turnDirection [0..1] : NT_TurnDirection

The attribute “turnDirection” indicates what sort of change in direction is accomplished in taking the turn. This attribute is mainly to aid in the creation of instructions for navigation.

```
NT_Turn :: turnDirection [0..1] : NT_TurnDirection
```

9.3.2.4 Attribute: isTraversable : Boolean

The attribute “isTraversable” indicates whether this turn is usable from the incoming link in any circumstances.

```
NT_Turn :: isTraversable : Boolean
```

A turn marked as not traversable (isTraversable = FALSE), is never usable as part of a route. A turn usable in some conditions but not in others is marked with constraints which explain the conditions when the turn is usable or not, see 9.4.2.

9.3.2.5 Attribute: isManeuver : Boolean

The attribute “isManeuver” indicates whether this turn is always usable from the incoming link. Essentially, if “isManeuver” is true, then the turn constitutes a maneuver than can always be used regardless of how the incoming link was entered.

```
NT_Turn :: isManeuver: Boolean
```

9.3.2.6 Attribute: entryCount : Integer

The attribute “entryCount” indicates how far around the intersection the entry link occurs relative to the exit link. In right-hand lane systems the count is counter-clockwise. In left-hand systems the count is clockwise. In non-road systems, the number can often be ignored. Like many of the attributes of turn, the entry count is mainly to aid in the creation of instructions for navigation.

```
NT_Turn :: entryCount: Integer
```

9.3.2.7 Role: toLink : NT_Link

The association role “toLink” specifies the link into which this turn navigates.

```
NT_Turn :: toLink : NT_Link
```

9.3.2.8 Role: fromLink : NT_Link

The association role “fromLink” specifies the link from which this turn navigates.

```
NT_Turn :: fromLink: NT_Link
```

9.3.2.9 Role: constraint : NT_Constraint

The association role “constraint” aggregates all constraints that are associated with this turn.

```
NT_Turn :: constraint : NT_Constraint
```

9.3.2.10 Role: junction : NT_Junction

The association role “junction” specifies the junction (node) at which this turn occurs.

```
NT_Turn :: junction : NT_Junction
```

9.3.2.11 Role: advisory : NT_Advisory

The association role “advisory” specifies the advisories associated with this turn.

```
NT_Turn :: advisory : NT_Advisory
```

9.3.2.12 Role: maneuver : NT_Maneuver

The association role “maneuver” specifies the maneuvers associated with this turn.

```
NT_Turn :: advisory : NT_Advisory
```

9.3.2.13 Operation: cost

The operation “cost” returns a costed turn given a cost function.

```
NT_Turn :: cost(function : NS_CostFunction) : NS_CostedTurn
```

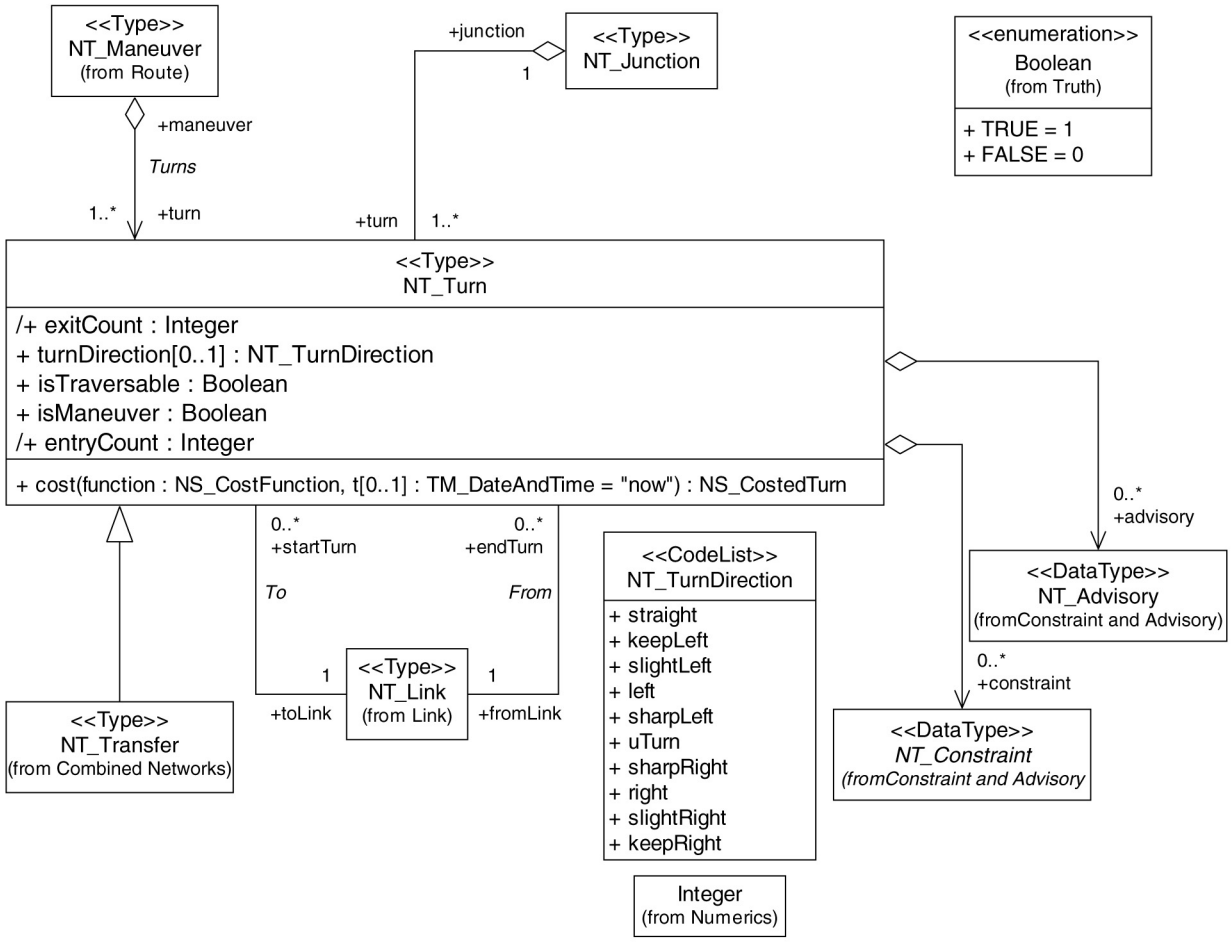

Figure 94 — Context Diagram: NT_Turn

9.3.3 NT_TurnDirection

The code list “NT_TurnDirection” enumerates the types of turns used in a route. The descriptions should be adjusted for the cultural and application context. The default values in the code list are as follows: straight, keepLeft, slightLeft, left, sharpLeft, uTurn, sharpRight, right, slightRight, and keepRight. The UML for NT_TurnDirection is given in Figure 95.

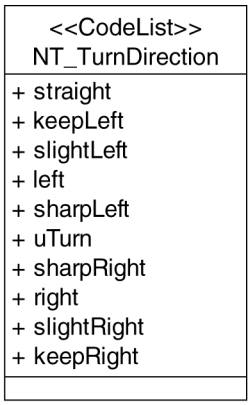

Figure 95 — Context Diagram: NT_TurnDirection

9.3.4 NT_Junction

9.3.4.1 Semantics

The type “NT_Junction” replaces its supertype “TP_Node” for NT_Network instances. In terms of navigation, a junction is where turns occur. The UML for NT_Junction is given in Figure 96.

9.3.4.2 Attribute: junctionType[0..1] : NT_JunctionType

The attribute “junctionType” describes the type of this junction.

```
NT_Junction :: junctionType[0..1] : NT_JunctionType
```

9.3.4.3 Role: turn : NT_Turn

The role “turn” aggregates all NT_Turns that occur at this junction.

```
NT_Junction :: turn : NT_Turn
```

9.3.4.4 Role: spoke : NT_Link

The role “spoke” is inherited from NT_Node. In a junction, the spokes are links.

```
NT_Junction :: spoke : NT_Link
```

9.3.4.5 Role: maximalComplex : NT_Network

The role “maximalComplex” is inherited from NT_Node. In a junction, the maximalComplex is a network.

```
NT_Junction :: maximalComplex : NT_Network
```

9.3.4.6 Operation: entryLinks

The operation “entryLinks” returns a list of potential entry links that are valid for a particular exit link. Thus, for each element “entry” in the returned sequence, the pair “entry, startLink” constitutes a valid turn. The parameters of the operation are the following:

startLink the first link to act as a starting point for the count, and list;

direction the angular direction in which to count.

```
NT_Junction :: entryLinks(
      startLink : NT_Link, direction : NT_AngularDirection = clockwise) :
    Sequence<NT_Links>
```

9.3.4.7 Operation: nextExit

The function “nextExit” returns an exit link at a particular offset from a given entry link.

entryLink the entry link from which to start the search;

count the offset of the requested exit link, default “1” which is the next available exit link.

```
NT_Junction ::
      nextExit(entryLink : NT_Link, count[0..1] : Integer = 1) : NT_Link
```

9.3.4.8 Operation: exitLinks

The operation “exitLinks” returns a list of potential exit links that are valid for a particular entry link. Thus, for each element “exit” in the returned sequence, the pair “startLink, exit” constitutes a valid turn. The parameters of the operation are the following:

startLink the first link to act as a starting point for the count, and list;

direction the angular direction in which to count.

```
NT_Junction :: exitLinks(
      startLink : NT_Link, direction : NT_AngularDirection =
      counterclockwise) : Sequence<NT_Links>
```

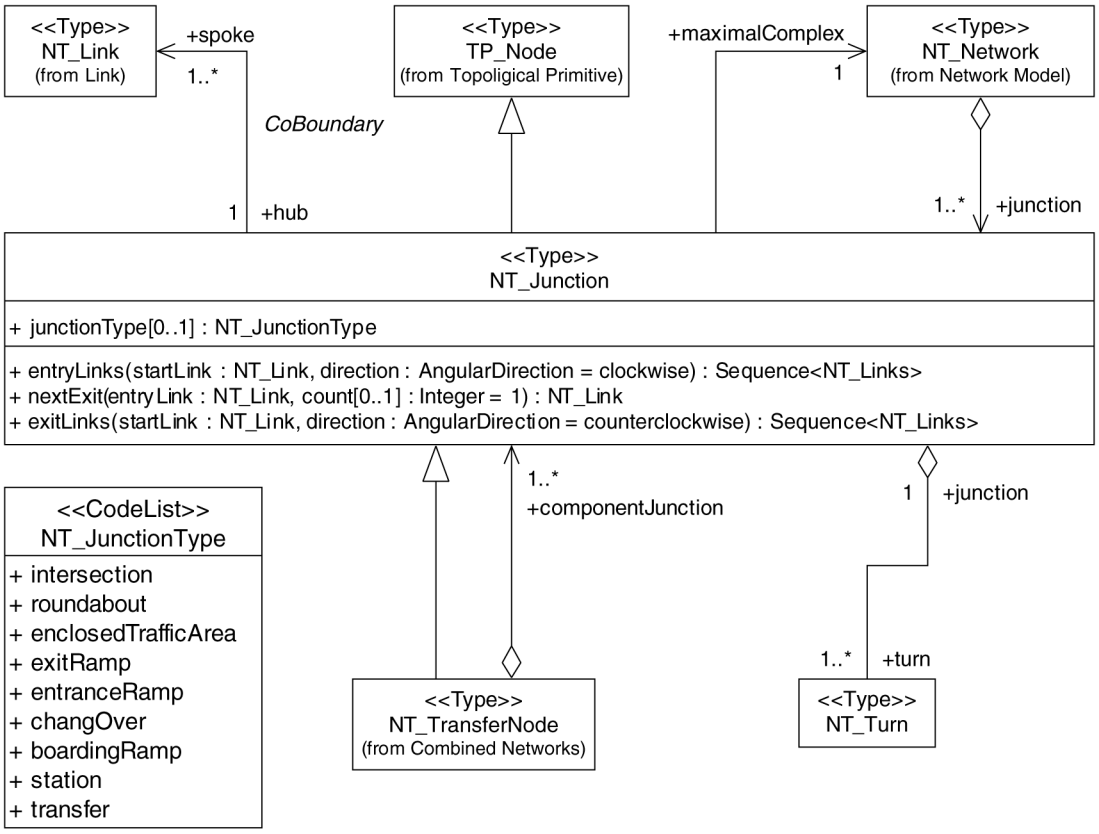

Figure 96 — Context Diagram: NT_Junction

9.3.5 NT_JunctionType

The code list “NT_JunctionType” provides a value domain for the types of junctions. The current values of junction type are “intersection”, “roundabout”, “enclosedTrafficArea”, “exitRamp”, “entranceRamp”, “changeOver”, “boardingRamp”, “station”, and “transfer”. The UML for NT_JunctionType is given in Figure 97.

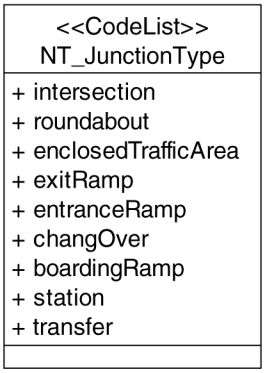

Figure 97 — Context Diagram: NT_JunctionType

9.3.6 NT_AngularDirection

The enumeration “NT_AngularDirection” provides the value domain for describing in which manner angles are measured. The possible values are “clockwise” and “counterclockwise”. As in ISO 19107, this refers to the view of the angle from above. At ground level, clockwise angles would increase towards the right, and counterclockwise angles increase towards the left. The UML for NT_AngularDirection is given in Figure 98.

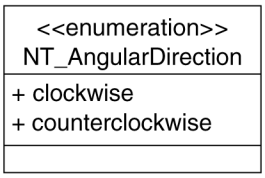

Figure 98 — Context Diagram: NT_AngularDirection

9.4 Package: Constraint and Advisory

9.4.1 Semantics

The package “Constraints and Advisory” in the model provides classes and types useful for associating constraints and advisory items to objects, usually turns, maneuvers and links.

9.4.2 NT_Constraint

9.4.2.1 Semantics

The datatype “NT_Constraint” is an abstract class that is the root class of those types use to specify turn and link restrictions. The UML for NT_Constraint is given in Figure 99.

9.4.2.2 Attribute: temporalValidity[0..1] : TM_Primitive

The attribute “temporalValidity” specifies the times at which this constraint is valid. In a temporal constraint, the temporalValidity may be used to indicate a time frame within which the temporal constraint is valid. For example, the temporalValidity may be “April-September” and the temporal constraint list “Monday-Friday, 8AM to 10AM”, meaning that the constraint is active only during those months, those days and between those times.

```
NT_Constraint :: temporalValidity[0..1] : TM_Primitive
```

9.4.2.3 Attribute: description[0..1] : CharacterString

The attribute “description” is a natural language description of the constraint, and may contain additional advisory information to be used in creating instructions in which this constraint plays a role.

```
NT_Constraint :: description[0..1] : CharacterString
```

9.4.2.4 Role: coConstraint[0..*] : CharacterString

The association role “coConstraint” associates this constraint with various other primitive constraints to further define this constraint.

```
NT_Constraint :: coConstraint[0..*] : NT_Constraint
```

The interpretation of a set of constraints is a Boolean intersection of all conditions (all conditions must be met for the constraint to apply). Boolean unions are given by separate constraints.

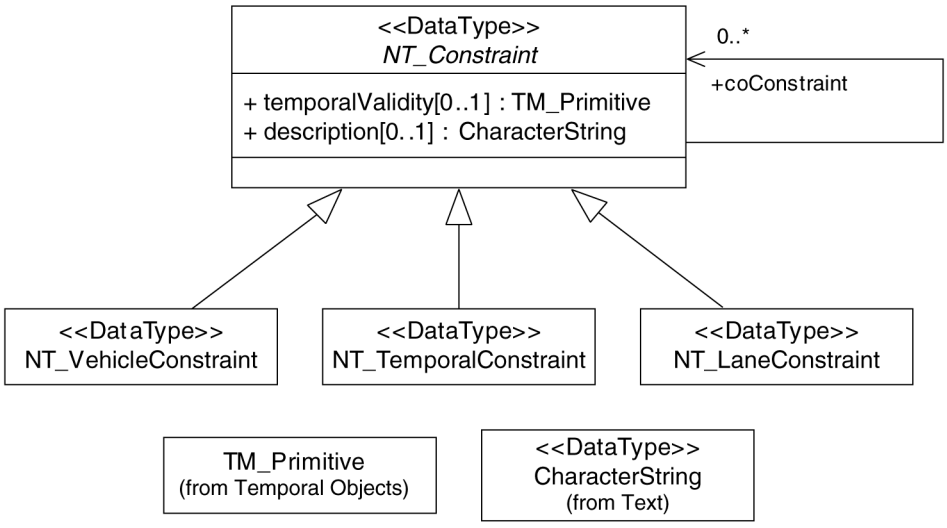

Figure 99 — Context Diagram: NT_Constraint

9.4.3 NT_VehicleConstraint

9.4.3.1 Semantics

The data type “NT_VehicleConstraint” is used to specify constraints on the type or size of a vehicle that is allowed to traverse a link or a turn. The UML for NT_VehicleConstraint is given in Figure 100.

9.4.3.2 Attribute: allowedVehicle[0..*] : NT_Vehicle

The attribute “allowedVehicle” is used to specify which vehicle types are allowed. If blank, the default assumption is that all vehicles except those specifically disallowed are allowed.

```
NT_VehicleConstraint :: allowedVehicle[0..*] : NT_Vehicle
```

9.4.3.3 Attribute: disallowedVehicle[0..*] : NT_Vehicle

The attribute “disallowedVehicle” is used to specify which vehicle types are specifically disallowed. If blank, the default assumption is that all vehicles except those specifically allowed are disallowed. If both allowedVehicle and disallowedVehicle are blank, then the default assumption is that all vehicle types are allowed.

```
NT_VehicleConstraint :: disallowedVehicle [0..*] : NT_Vehicle
```

9.4.3.4 Attribute: turnRadius[0..1] : Distance

The attribute “turnRadius” is the minimum radius of curvature of the turn or link under restriction. Vehicles incapable of turns of this radius are disallowed.

```
NT_VehicleConstraint :: turnRadius [0..1] : Distance
```

9.4.3.5 Attribute: grade[0..2] : Angle

The attribute “grade” is the maximum (in absolute value) slope of the turn or link under restriction. Two values are given if the link has both significant upward and downward segments. Positive angles are upward slopes, and negative angles are downward slopes. Vehicles incapable of navigating slopes of this grade are disallowed. If the application needs more precise information on grade, then the associated geometry object should be queried directly.

```
NT_VehicleConstraint :: grade [0..2] : Angle
```

9.4.3.6 Attribute: maxClearance[0..*] : Distance

The attribute “maxClearance” specifies the maximum heights of a vehicle allowed.

```
NT_LaneConstraint :: maxClearance[0..1] : Distance
```

9.4.3.7 Attribute: maxWeight[0..*] : Weight

The attribute “maxWeight” specifies the maximum weights of a vehicle allowed.

```
NT_LaneConstraint :: maxClearance[0..1] : Distance
```

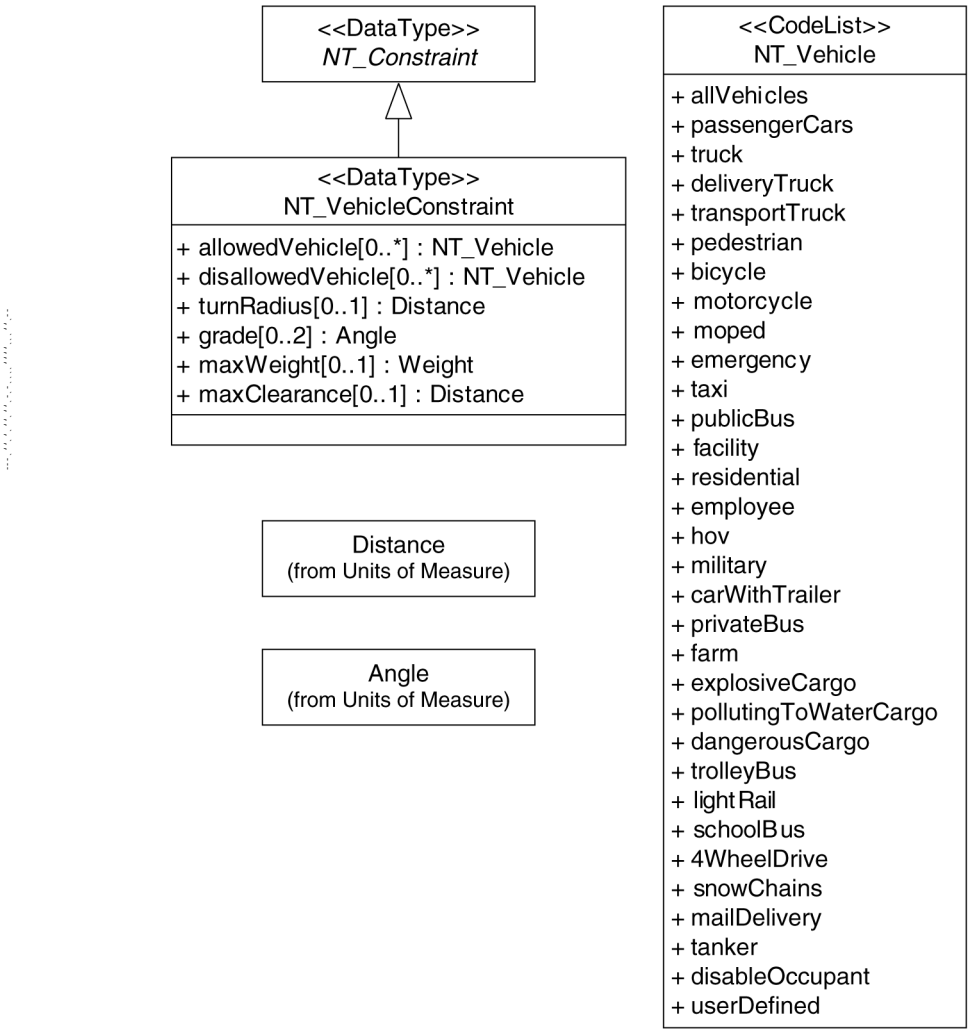

Figure 100 — Context Diagram: NT_VehicleConstraint

9.4.4 NT_TemporalConstraint

9.4.4.1 Semantics

The data type “NT_TemporalConstraint” specifies when a network entity may be traversed. The constraint may be listed in two distinct manners: when the network entity is closed to traffic or when it is opened. A co-constraint (see 9.4.2.4) on this entity may further specify vehicle types. The UML for NT_TemporalConstraint is given in Figure 101.

9.4.4.2 Attributes: allowedTime[0..1] : TM_Primitive

The attribute “allowedTime” is used to specify the time when the entity is open for use. If NULL, the value is assumed to be the complement of the “forbiddenTime”.

```
NT_TemporalConstraint :: allowedTime[0..1] : TM_Primitive
```

9.4.4.3 Attributes: forbiddenTime[0..1] : TM_Primitive

The attribute “forbiddenTime” is used to specify the time when the entity is closed for use. If NULL, the value is assumed to be the complement of the “allowedTime”.

```
NT_TemporalConstraint :: forbiddenTime[0..1] : TM_Primitive
```

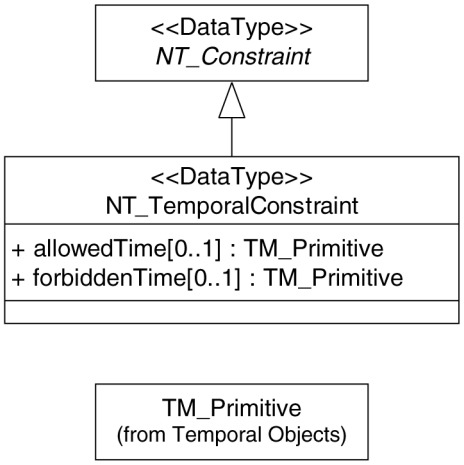

Figure 101 — Context Diagram: NT_TemporalConstraint

9.4.5 NT_LaneConstraint

9.4.5.1 Semantics

This datatype is used to list lane restrictions. Lane numbers are counted from the normal outside edge of allowable travel (rightmost in right-hand lane systems, leftmost in left-hand lane systems). Allowed lanes are ones from which the turn can be made both legally and physically (if the vehicle has a large turning radius). The UML for NT_LaneConstraint is given in Figure 102.

9.4.5.2 Attribute: allowedLanes[0..*] : Integer

The attribute “allowedLanes” specifies the allowed lanes from which the turn can be made. Lanes counted from the left are assigned positive numbers; the leftmost lane is “+1”. Lanes counted from the right are assigned negative numbers; the rightmost lane is “-1”.

```
NT_LaneConstraint :: allowedLanes[0..*] : Integer
```

9.4.5.3 Attribute: turningRadius[0..*] : Distance

The turning radii, if given, are ordered in the same manner as the allowed lanes and are the minimum turning radius required of a vehicle making this turn from this lane.

```
NT_LaneConstraint :: turningRadius[0..*] : Distance
```

9.4.5.4 Attribute: disallowedLanes[0..*] : Integer

The attribute “disallowedLanes” specifies the disallowed lanes from which the turn cannot be made.

```
NT_LaneConstraint :: disallowedLanes [0..*] : Integer
```

9.4.5.5 Attribute: applicableVehicleTypes[0..*] : NT_Vehicle

The attribute “applicableVehicleTypes” specifies the vehicle types to which this constraint applies.

```
NT_LaneConstraint :: applicableVehicleTypes[0..*] : NT_Vehicle
```

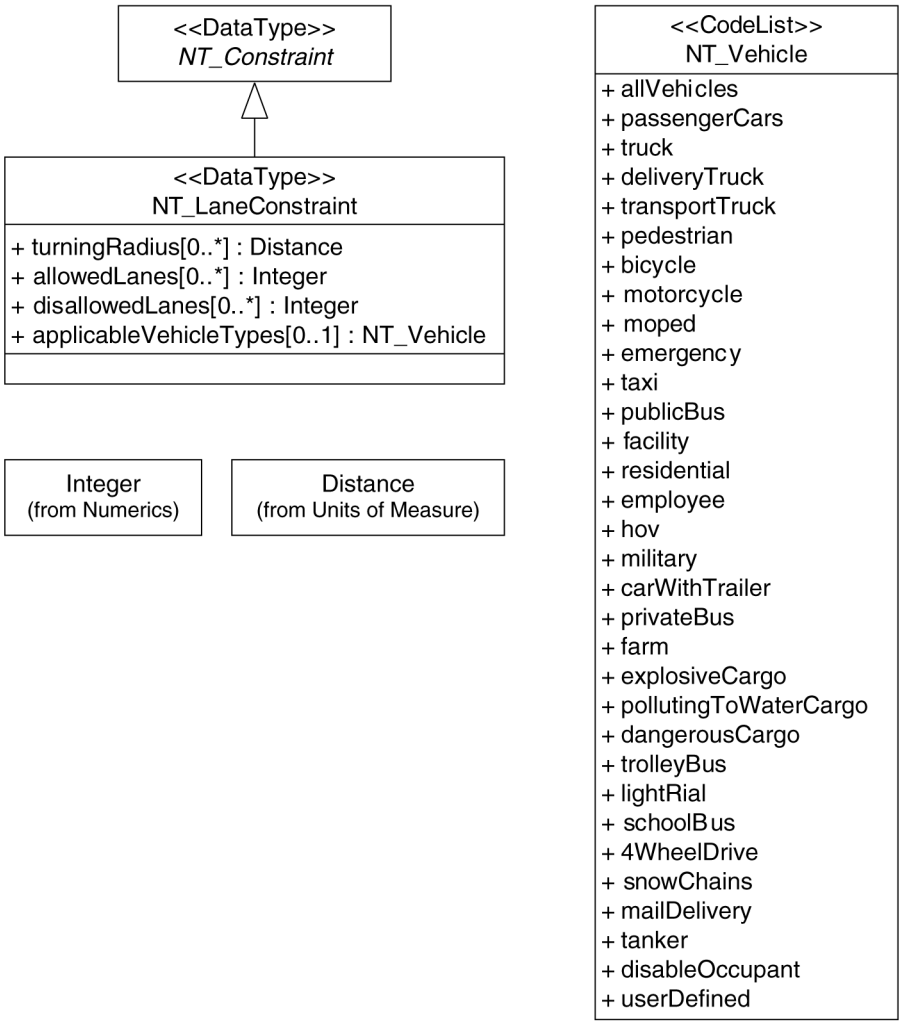

Figure 102 — Context Diagram: NT_LaneConstraint

9.4.6 NT_Vehicle

The code list “NT_Vehicle” lists the type of vehicle or traveller to which data can be applied. The current values are: “allVehicles”, “passengerCars”, “truck”, “deliveryTruck”, “transportTruck”, “pedestrian”, “bicycle”, “motorcycle”, “moped”, “emergency”, “taxi”, “publicBus”, “facility”, “residential”, “employee”, “hov”, “military”, “carWithTrailer”, “privateBus”, “farm”, “explosiveCargo”, “pollutingToWaterCargo”, “dangerousCargo”, “trolleyBus", “lightRail”, “schoolBus”, “4WheelDrive”, “snowChains”, “mailDelivery”, “tanker”, “disableOccupant”, and “userDefined”. The UML for NT_Vehicle is given in Figure 103.

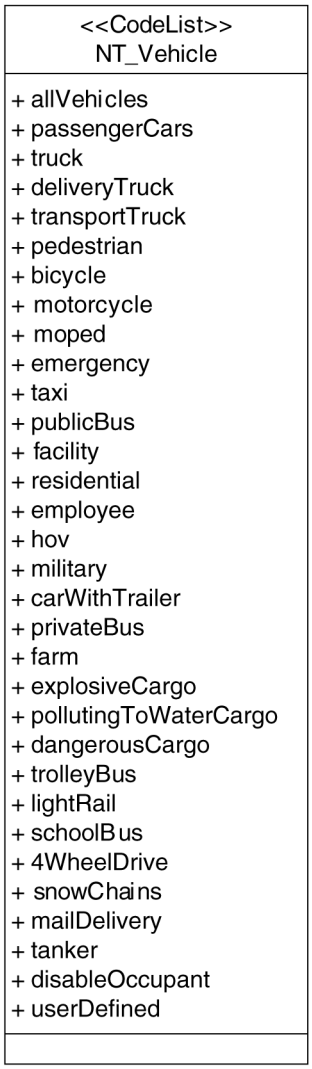

Figure 103 — Context Diagram: NT_Vehicle

9.4.7 NT_Advisory

9.4.7.1 Semantics

The data type “NT_Advisory” is used to attach free text information that might be useful in understanding navigation instructions involving an associated object. The UML for NT_Advisory is given in Figure 104.

9.4.7.2 Attribute: category : NT_AdvisoryCategory

The attribute ” NT_AdvisoryCategory” specifies the type of advisory being used.

```
NT_Advisory :: category : NT_AdvisoryCategory
```

9.4.7.3 Attribute: description[0..1] : CharacterString

The attribute “description” is the free text used to describe the nature of the advisory.

```
NT_Advisory :: description[0..1] : CharacterString
```

9.4.7.4 Attribute: element[0..*] : NT_AdvisoryElement

The attribute “element” is a list of additional information used to describe the nature of the advisory or the position of the items or features described in the advisory text in “description”.

```
NT_Advisory :: element[0..*] : NT_AdvisoryElement
```

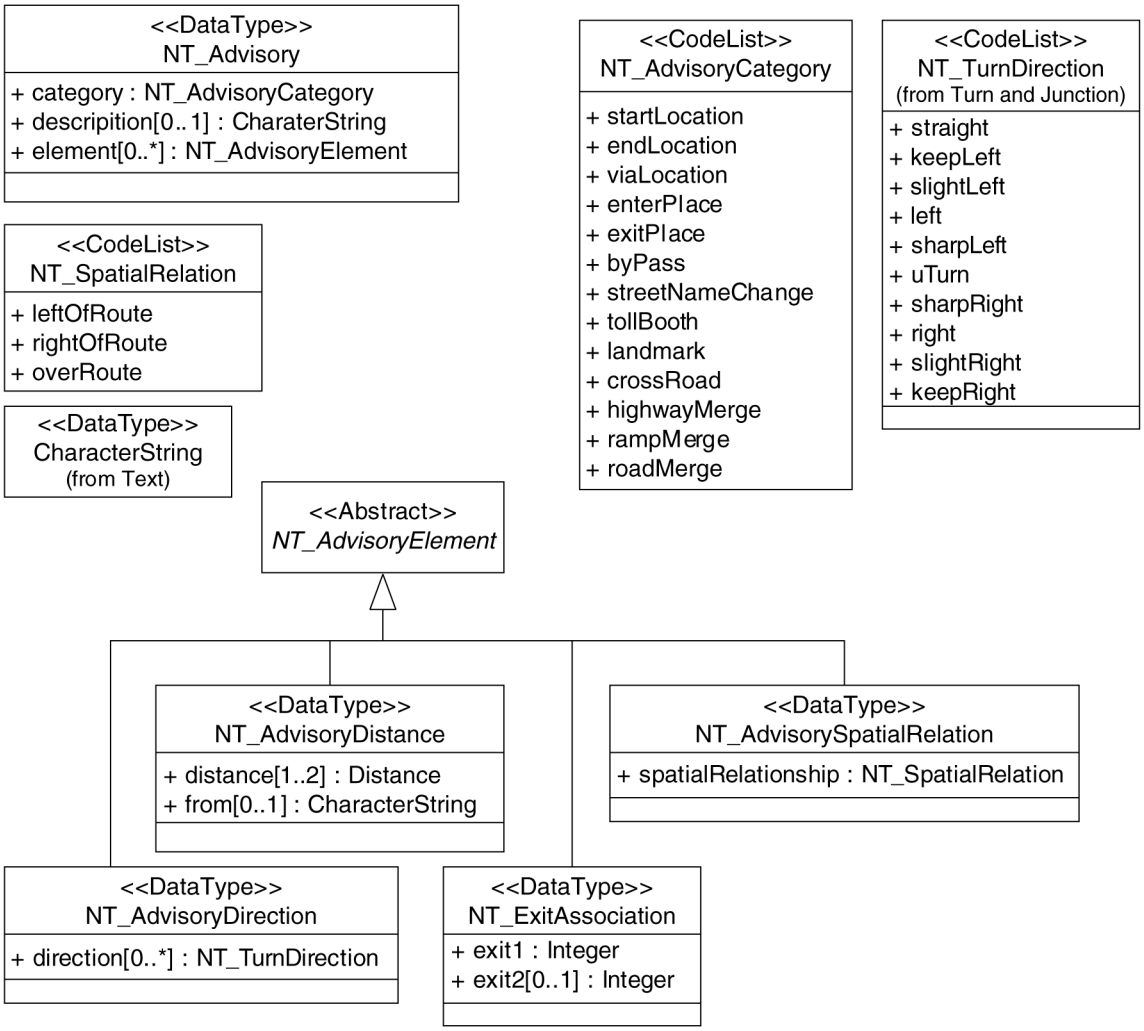

Figure 104 — Context Diagram: NT_Advisory

9.4.8 NT_SpatialRelation

The code list “NT_SpatialRelation” is used to describe the spatial relation between the route and the items mentioned in an advisory, constraint or other data. The current value domains consist of “leftOfRoute”, “rightOfRoute”, and “overRoute”. The UML for NT_SpatialRelation is given in Figure 105.

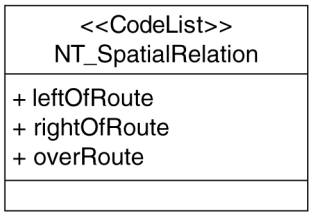

Figure 105 — Context Diagram: NT_SpatialRelation

9.4.9 NT_AdvisoryCategory

The code list “NT_AdvisoryCategory” is used to list the types of advisories used. The current possible values of “NT_AdvisoryCategory” and their meanings are listed below.

“startLocation”	beginning of route
“endLocation”	end of route
“viaLocation”	requested stopping point
“enterPlace”	entry of a start, via or end location
“exitPlace”	exit of a start, via or end location
“byPass”	bypass of a urban area or other high density area
“streetNameChange”	indication of a name change of the street even though a maneuver has not taken place
“tollBooth”	toll taking facility
“landmark”	landmark visible from the route
“crossRoad”	crossroads or intersection
“highwayMerge”	merging traffic between two highways
“rampMerge”	merging traffic from a ramp, access road, or slip road
“roadMerge”	merging traffic from a road

The UML for NT_AdvisoryCategory is given in Figure 106.

Figure 106 — Context Diagram: NT_AdvisoryCategory

9.4.10 NT_AdvisoryElement

The abstract class “NT_AdvisoryElement” is the root of a subtyping hierarchy of data types used to describe advisories. The UML for NT_AdvisoryElement is given in Figure 107.

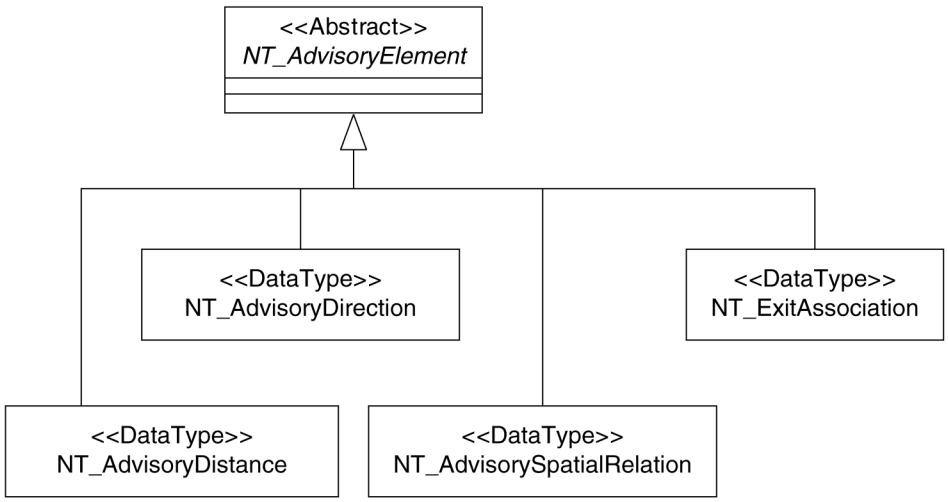

Figure 107 — Context Diagram: NT_AdvisoryElement

9.4.11 NT_ExitAssociation

9.4.11.1 Semantics

The data type “NT_ExitAssociation”, a subtype of “NT_AdvisoryElement”, is used to associate an advisory with an exit or pair of exits at a junction. The UML for NT_ExitAssociation is given in Figure 108.

EXAMPLE Toll roads often have temporally restricted exits that require the use of exact change when the tollbooth is not manned.

9.4.11.2 Attribute: exit1 : Integer

The attribute “exit1” is the exit count, from the entry point of the route into the junction, of the primary exit associated with this advisory.

```
NT_ExitAssociation :: exit1 : Integer
```

9.4.11.3 Attribute: exit2[0..1] : Integer

The optional attribute “exit2” is the exit count from the entry point of the route into the junction, of a secondary exit associated with this advisory.

```
NT_ExitAssociation :: exit2[0..1]: Integer
```

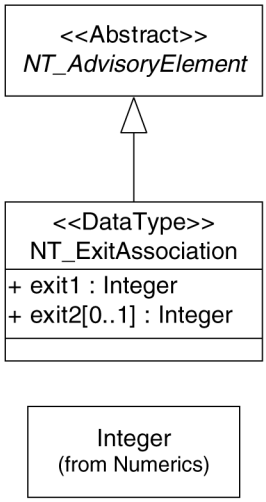

Figure 108 — Context Diagram: NT_ExitAssociation

9.4.12 NT_AdvisoryDirection

9.4.12.1 Semantics

The data type “NT_AdvisoryDirection”, subtyped under “NT_AdvisoryElement”, is used to describe the direction associated with an item mentioned in the descriptive text of the associated advisory. The UML for NT_AdvisoryDirection is given in Figure 109.

9.4.12.2 Attribute: direction[0..*] : NT_TurnDirection

The attribute “direction” specifies the direction of the item mentioned in the descriptive text of the associated advisory using the same terminology that is used for turn angles.

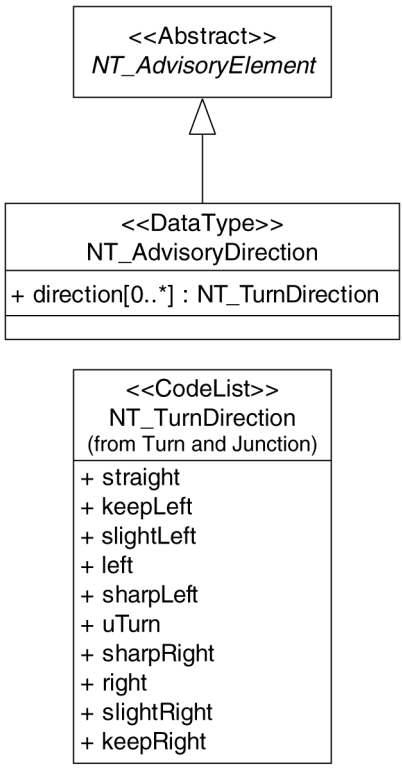

Figure 109 — Context Diagram: NT_AdvisoryDirection

9.4.13 NT_AdvisoryDistance

9.4.13.1 Semantics

The data type “NT_AdvisoryDistance”, subtyped under “NT_AdvisoryElement”, describes the distance at which the item mentioned in the descriptive text of the associated advisory occurs. The UML for NT_AdvisoryDistance is given in Figure 110.

9.4.13.2 Attribute: distance[1..2] : Distance

The attribute “distance” specifies the distance at which the item mentioned in the descriptive text of the associated advisory occurs. Two values represent a range for the distance.

```
NT_AdvisoryDistance :: distance[1..2] : Distance
```

9.4.13.3 Attribute: from[0..1] : CharacterString

The attribute “from” specifies the point from which the measurement of the distance is made. If “blank” then the default assumption is that the measurement is from the approximate location of the vehicle at the time the advisory takes effect.

NOTE The relationship between this value and the offset measure in linear reference systems is currently left to the application semantics.

```
NT_AdvisoryDistance :: from[0..1] : CharacterString
```

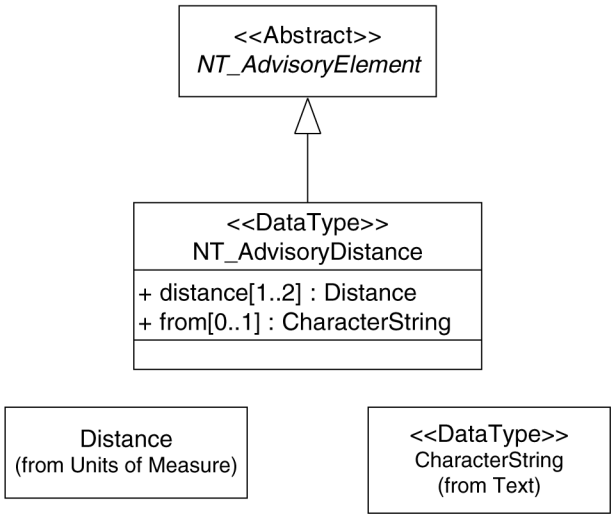

Figure 110 — Context Diagram: NT_AdvisoryDistance

9.4.14 NT_AdvisorySpatialRelation

9.4.14.1 Semantics

The data type “NT_AdvisorySpatialRelation”, subtyped under “NT_AdvisoryElement”, describes the spatial relationship between the route and the item mentioned in the descriptive text of the associated advisory. The UML for NT_AdvisorySpatialRelation is given in Figure 111.

9.4.14.2 Attribute: spatialRelationship : NT_SpatialRelation

The attribute “spatialRelationship” specifies the spatial relationship, using the code list NT_SpatialRelation as its value domain.

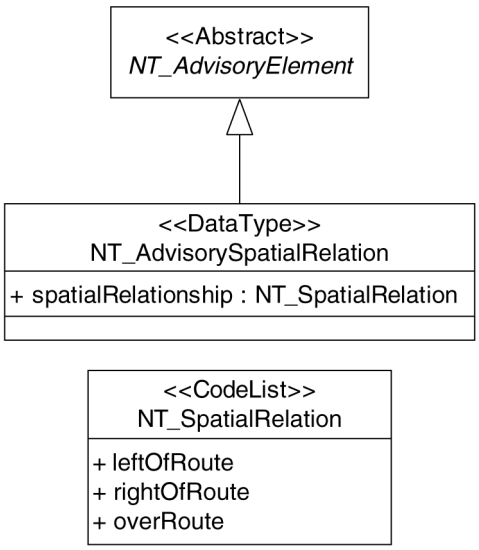

Figure 111 — Context Diagram: NT_AdvisorySpatialRelation

9.5 Package: Link

9.5.1 Semantics

The package “Link” contains classes for the description of links.

9.5.2 NT_Link

9.5.2.1 Semantics

The type “NT_Link”, which is a subtype of “TP_DirectedEdge”, is used in networks for all of the directed edges. Thus, links are the basic oriented curve elements of a network topology. Links are always associated with a topological complex, via the TP_Primitive :: complex association role inherited from TP_Primitive through TP_DirectedEdge. They are also potentially associated with a geometric complex through the role TP_Primitive :: geometry inherited from TP_Primitive. Because of the dimensionality constraints on this role, it will be an association with an instance of GM_Curve. In navigation applications, this associated curve is almost always, but not necessarily, associated with a linear reference system (LRS) as part of its coordinate reference system (CRS). The LRS allows attributes to be associated with parts of the link through intervals in the measure and thus, indirectly, to subsets of the curve that may be independent of its defining segmentation (hence the name “dynamic segmentation”).

If the underlying edge is not traversable in at least one direction, then it should not be part of the network. This can be expressed in OCL as follows.

```
NT_Link:
-- At least one direction of the NT_Link is traversable in the network
      {Not(NT_Link.isTraversable) implies NT_Link.negate().isTraversable}
      {Not(NT_Link.negate().isTraversable) implies NT_Link.isTraversable}
```

The UML for NT_Link is given in Figure 112.

9.5.2.2 Attribute: isTraversable : Boolean

The attribute “isTraversable” indicates whether this link is navigable in this direction.

```
NT_Link :: isTraversable : Boolean
```

9.5.2.3 Attribute: isUturnPossible : Boolean

The attribute “isUturnPossible” indicates whether it is possible for a vehicle to execute a U-turn along this link, and thus transfer to its opposite directed edge link.

NOTE This is usually not needed unless during execution the vehicle makes an error, or the navigation software decides to use a U-turn to overcome a turning restriction at the previous junction. For example, if a left turn is illegal, the same effect can be accomplished by a right turn followed immediately by a U-turn.

```
NT_Link :: isUturnPossible: Boolean
```

9.5.2.4 Attribute: routeSegmentCategory[0..*] : NT_RouteSegmentCategory

The attribute “routeSegmentCategory” is used to specify the category of the route segment.

```
NT_Link :: routeSegmentCategory[0..*] : NT_RouteSegmentCategory
```

9.5.2.5 Role : startTurn [0..*] : NT_Turn

The role “startTurn” associates this link with all potential turns that can be used to enter this link. Maneuvers can be accessed through the turns.

```
NT_Link :: startTurn [0..*] : NT_Turn
```

9.5.2.6 Role : endTurn[0..*] : NT_Turn

The role “endTurn” associates this link with all potential turns that can be used to exit this link.

```
NT_Link :: endTurn [0..*] : NT_Turn
```

9.5.2.7 Role : constraint : NT_Constraint

The role “constraint” aggregates all the constraints associated with this link.

```
NT_Link :: constraint [0..*] : NT_Constraint
```

9.5.2.8 Role : advisory : NT_Advisory

Means: The role “advisory” aggregates all the constraints associated with this link.

```
NT_Link :: constraint [0..*] : NT_ Advisory
```

9.5.2.9 Operation: cost

The operation “cost” dynamically associates this link with a costed link based upon the choice of a cost function.

```
NT_Link :: cost(function : NS_CostFunction) : NS_CostedLink
```

9.5.2.10 Operation: exitTurns

The operation “exitTurns” creates a list of exit turns, sorted by angle (closest first), for the terminal junction of this link that can be used to navigate from this link to another.

```
NT_Link :: exitTurns() : Sequence<NT_Turn>
```

9.5.2.11 Operation: entranceTurns

The operation “entranceTurns” creates a list of entrance turns, sorted by angle (closest first), for the initial junction of this link that can be used to navigate to this link from another link.

```
NT_Link :: entranceTurns () : Sequence<NT_Turn>
```

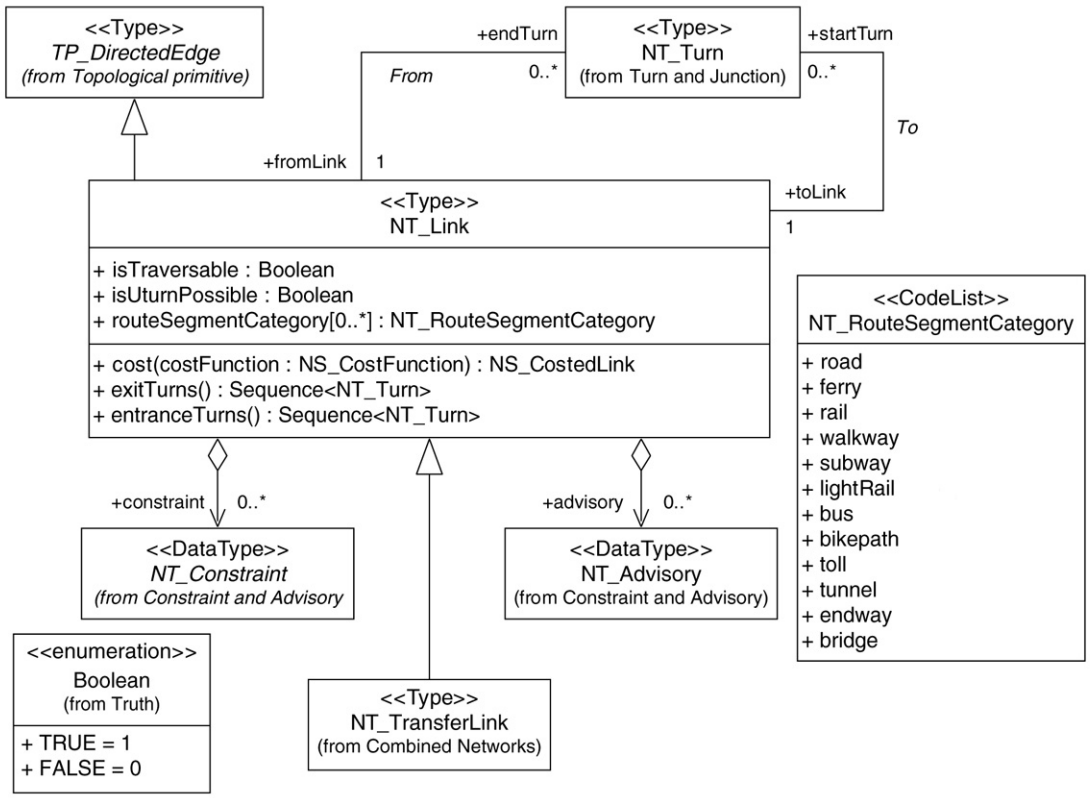

Figure 112 — Context Diagram: NT_Link

9.5.3 NT_RouteSegmentCategory

The code list “NT_RouteSegmentCategory” is the value domain for segment categories. The current list includes: “road”, “ferry”, “rail”, “walkway”, “subway”, “airway”, “lightRail”, “bus”, “airRoute”, “bikepath”, “toll”, “tunnel”, “endway” and “bridge”. The UML for NT_RouteSegmentCategory is given in Figure 113.

Figure 113 — Context Diagram: NT_RouteSegmentCategory

9.6 Package: Network Position

9.6.1 Semantics

The package “Network Position” contains classes for specifying position within a network, using a linear reference system (LRS).

9.6.2 NT_LinkPosition

9.6.2.1 Semantics

The data type “NT_LinkPosition” is used to describe position along a link by using an LRS. The UML for NT_LinkPosition is given in Figure 114.

9.6.2.2 Role: link : NT_Link

The role “link” indicates which link the position is on. The measure used is actually against the underlying geometric object, but the link is used to give an entry point into the network. The link specified is the nearest navigable link to the actual location (where “side of road” is taken into account).

```
NT_LinkPosition :: link : NT_Link
```

9.6.2.3 Attribute: linkMeasure : MemberName

The attribute “linkMeasure” is the name of the measure axis from the LRS that is used to specify this link position. This is the same as the name in the CRS for the underlying geometric curve.

```
NT_LinkPosition :: linkMeasure : MemberName
```

9.6.2.4 Attribute: marker : Measure

The attribute “marker” is the measurement in the LRS for this link position. Looking at the underlying geometry and finding the position of this marker will determine the spatial coordinates of the position.

```
NT_LinkPosition :: marker : Measure
```

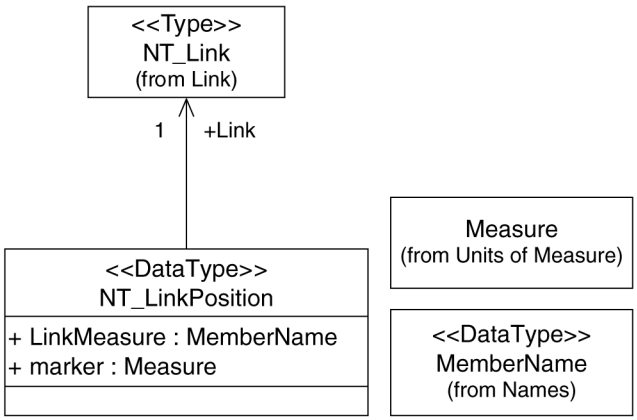

Figure 114 — Context Diagram: NT_LinkPosition

9.6.3 NT_NetworkPosition

9.6.3.1 Semantics

The union class “NT_NetworkPosition” unites the value domains of the two mechanisms for specifying a network position. The UML for NT_NetworkPosition is given in Figure 115.

9.6.3.2 Attribute: node : NT_Junction

If the network position is given by a node, then the attribute “node” is the NT_Junction (the subtype of node used for networks) for the position.

```
NT_NetworkPosition :: node : NT_Junction
```

9.6.3.3 Attribute: linkPosition : NT_LinkPosition

If the network position is given by a link position, then the attribute “linkPosition” is the NT_LinkPosition for the position.

```
NT_NetworkPosition :: linkPosition : NT_LinkPosition
```

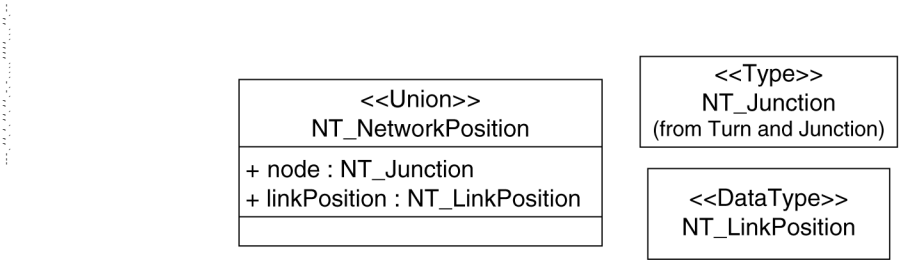

Figure 115 — Context Diagram: NT_NetworkPosition

9.7 Package: Route

9.7.1 Semantics

The package “Route” contains classes for specifying a route within a network. Since a route is simply a special type of 1-dimensional complex (a composite line if realized by geometry), it is a subtype of network.

9.7.2 NT_Route

9.7.2.1 Semantics

The type “NT_Route” describes routes defined by a geometric object (a composite curve) and starts and stops positions on that geometry. A route is calculated from and optimizes a cost function from the set of possible routes that traverse a set of target positions. The recalculate function determines a new route that optimizes the new cost function from the passed target position to the end of the route. If the cost function has not changed, the new route should coincide with old one.

The two canonical representations of a route are

— an ordered list of directed links to be traversed,

— an ordered list of maneuvers to be made.

Each maneuver advisory or route instruction is associated with a link, a maneuver or a turn of the route. The UML for NT_Route is given in Figure 116.

9.7.2.2 Attribute: summary : NT_RouteSummary

The attribute “summary” contains general information about the route.

```
NT_Route :: summary : NT_RouteSummary
```

9.7.2.3 Attribute: geometry : GM_CompositeCurve

The attribute “geometry” contains the geometry of the route expressed as a composite curve. The intent is that the components of the curve would be the major segments of the route, but this level of organization is application specific.

```
NT_Route :: geometry : GM_CompositeCurve
```

9.7.2.4 Role: maneuver : NT_Maneuver

The association role “maneuver” aggregates an ordered set of maneuvers that constitute the major executable portions of the route. As a composite curve, the route is an ordered alternating set of links and turns. Critical sequences of turns and links are aggregated into maneuvers; thus, the route can be viewed also as an alternating set of links and maneuvers.

```
NT_Route :: maneuver : NT_Maneuver
```

9.7.2.5 Operation: recalculate

The operation “recalculate” causes the route to be recalculated from some point along it (given by a waypoint). The return value of the function is the new route. The returned route follows the original at least up to and through the passed waypoints. The inputs are as follows:

from	point along the current route where the returned route can diverge,
cost	“new” cost function for the remainder of the route.

```
NT_Route ::
      recalculate(from : NT_WayPoint, cost : NS_CostFunction) : NT_Route
```

9.7.2.6 Operation: asLinks

The operation “asLinks” returns a version of the route as a sequence of links (directed edges).

```
NT_Route :: asLinks() : Sequence<NT_Link>
```

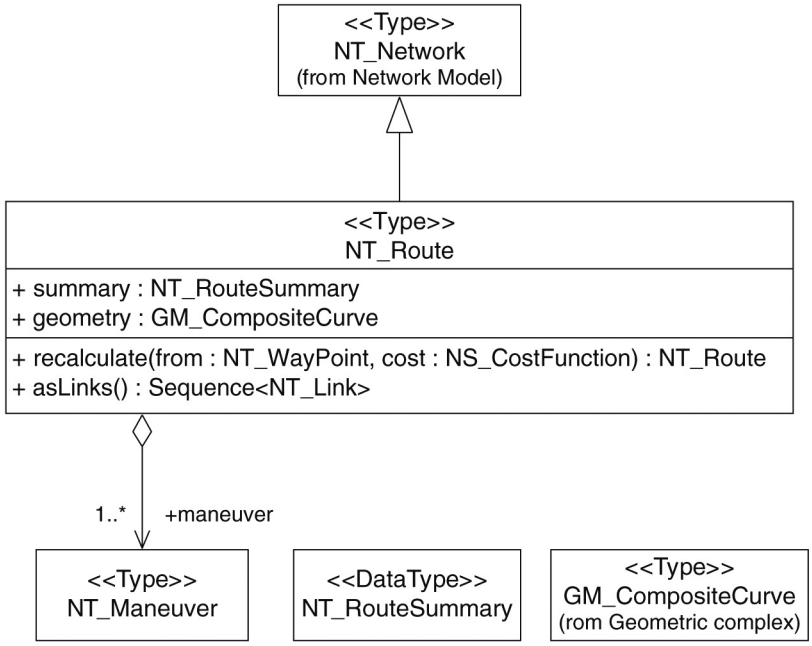

Figure 116 — Context Diagram: NT_Route

9.7.3 NT_RouteSummary

9.7.3.1 Semantics

The data type “NT_RouteSummary” contains route summary information. The UML for NT_RouteSummary is given in Figure 117.

9.7.3.2 Attribute: time : TM_Duration

The attribute “time” returns the expected time required for execution of the route.

```
NT_RouteSummary :: time : TM_Duration
```

9.7.3.3 Attribute: distance : Distance

The attribute “distance” returns the length of the route.

```
NT_RouteSummary :: distance : Distance
```

9.7.3.4 Attribute: extent : EX_GeographicExtent

The attribute “extent” returns the geographic extent of the route.

```
NT_RouteSummary :: extent : EX_GeographicExtent
```

9.7.3.5 Attribute: begin : NT_NetworkPosition

The attribute “begin” returns the location of the first point on the route, as a network position.

```
NT_RouteSummary :: begin : NT_NetworkPosition
```

9.7.3.6 Attribute: stops[0..1] : NT_WayPointList

The attribute “stops” returns the location of the stops made on the route, as network positions. These are part of the route request.

```
NT_RouteSummary :: stops[0..1] : NT_WayPointList
```

9.7.3.7 Attribute: end : NT_NetworkPosition

The attribute “end” returns the location of the last point on the route, as a network position.

```
NT_RouteSummary :: end : NT_NetworkPosition
```

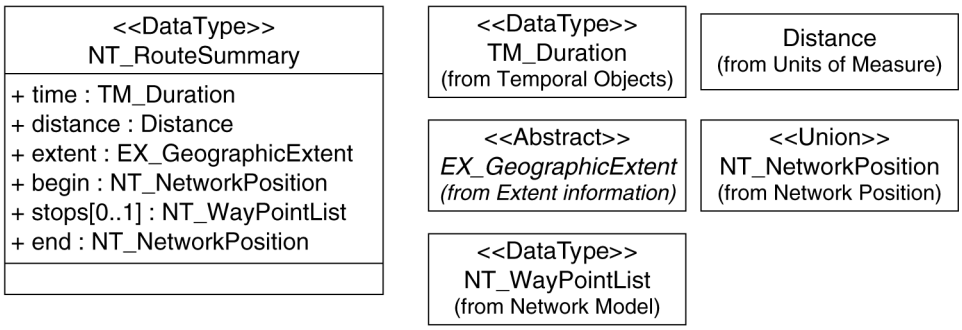

Figure 117 — Context Diagram: NT_RouteSummary

9.7.4 NT_Maneuver

9.7.4.1 Semantics

The type “NT_Maneuver” is used to describe maneuvers. A maneuver is a legal sequence of actions, given by a sequence of turns, each of which terminates on the link that is the start of the next turn. Most maneuvers are single turns; but, for some turns, traversability is dependent on route history (such as an ending turn in a U-turn across a divided highway). Since in this case, the turn’s traversability cannot be decided by a single link’s history, the maneuver extends that history to a sufficient length to determine traversability. The total length of most maneuvers is quite short.

Maneuvers can also contain U-turns, which is slightly counter-intuitive since, under most circumstances, optimal routes would not use a U-turn (which causes a visit to the same node twice). The problem occurs because of turn restrictions. If a left turn is the optimal choice at a junction, but a left turn is illegal, then a sequence of a right turn, a U-turn and a straight-through would be a legal approximation of the optimal route.

NOTE U-turns create a costing problem, since if not done at an intersection, the cost of the two links is not fully realized. How to apportion costs for U-turns and the links involved is an issue left to the application.

The UML for NT_Maneuver is given in Figure 118.

9.7.4.2 Attribute: isTraversable : Boolean

The attribute “isTraversable” specifies if this maneuver is usable. Normally, non-traversable maneuvers would be of little use, but in an active database where a maneuver might become invalid for an indeterminate time, or be needed for historical purposes, a non-traversable maneuver may be useful.

```
NT_Maneuver :: isTraversable : Boolean
```

9.7.4.3 Role: turns : NT_Turn

The association role “turns” aggregates the ordered set of turns involved in this maneuver. Starting and costing a maneuver at a turn is a computational convenience, allowing the standard graph theory algorithms to operate more or less in the normal manner on the graph of links and maneuvers, as they would in a non-restricted graph of links.

```
NT_Maneuver :: turns : NT_Turn {ordered}
```

9.7.4.4 Role: advisory : NT_Advisory

The association role “advisory” aggregates all advisories associated with this maneuver as a whole. Other advisories may be associated with its turns and spanning links separately.

```
NT_Maneuver :: advisory : NT_Advisory
```

9.7.4.5 Role: constraint : NT_Constraint

The association role “constraint” aggregates all constraints associated with this maneuver as a whole. Other constraints may be associated with its turns and spanning links separately.

```
NT_Maneuver :: constraint : NT_Constraint
```

9.7.4.6 Operation: cost

The operation “cost” calculates the cost for this maneuver given a cost function. The default reasoning is that the cost of a maneuver is the sum of the cost of its components, but this may not be the case. If a U-turn is involved, or if there is a dependency in the timing of its parts, the cost of a maneuver can be significantly different (either greater or less) than the sum of the costs of its components. The operation shall receive a cost function as an input parameter. The type of the measure returned will be consistent with the cost function definition.

```
NT_Maneuver :: cost(function : NS_CostFunction) : Measure
```

9.7.4.7 Operation: startTurn

The operation “startTurn” returns the first turn of the maneuver.

```
NT_Maneuver :: startTurn() : NT_Turn
```

9.7.4.8 Operation: endTurn

The operation “endTurn” returns the last turn of the maneuver.

```
NT_Maneuver :: endTurn() : NT_Turn() :
```

9.7.4.9 Operation: startLink

The operation “startLink” returns the first link of the maneuver. This is the link that is the entry for the start turn. This function allows an easy algorithm for building the input graph for optimization.

```
NT_Maneuver :: startLink() : NT_Link
```

9.7.4.10 Operation: endLink

The operation “endLink” returns the last link of the maneuver. This is the link that is the exit for the end turn. This function allows an easy algorithm for building the input graph for optimization.

```
NT_Maneuver :: endLink() : NT_Link
```

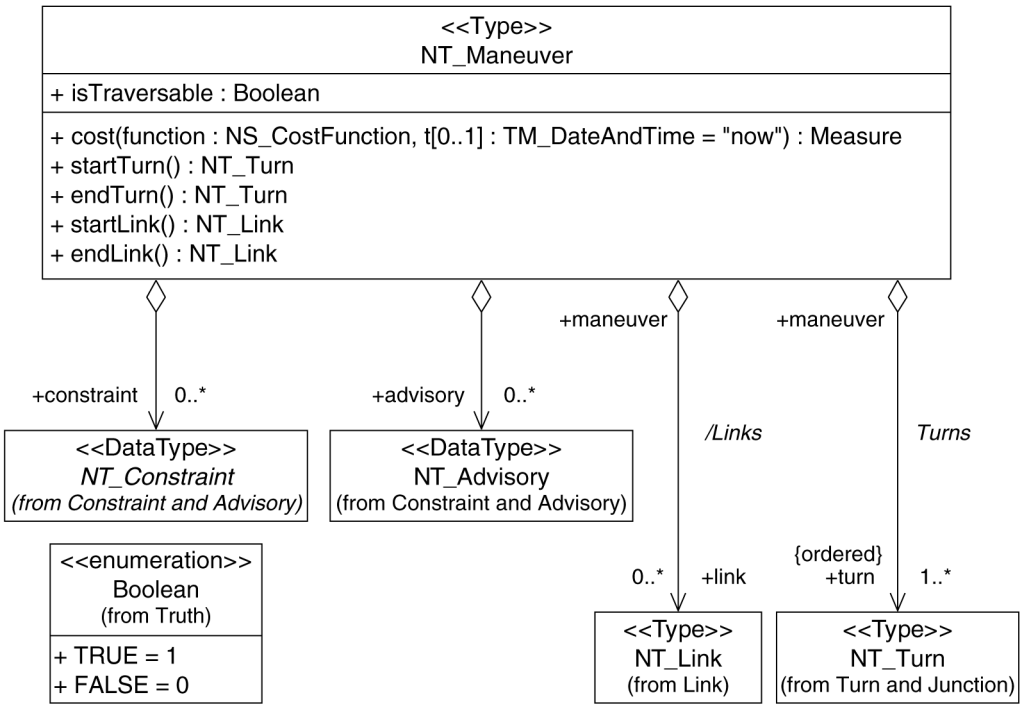

Figure 118 — Context Diagram: NT_Maneuver

9.8 Package: Combined Networks

9.8.1 Semantics

The package “Combined Networks” contains types and classes useful in creating larger networks by joining smaller ones. The general purpose is to supply mechanisms for navigation across networks from different sources. These techniques are valid both for single-mode and multi-mode networks, depending on whether the joining classes include a mode transition. A summary of the combined network package is given in Figure 119.

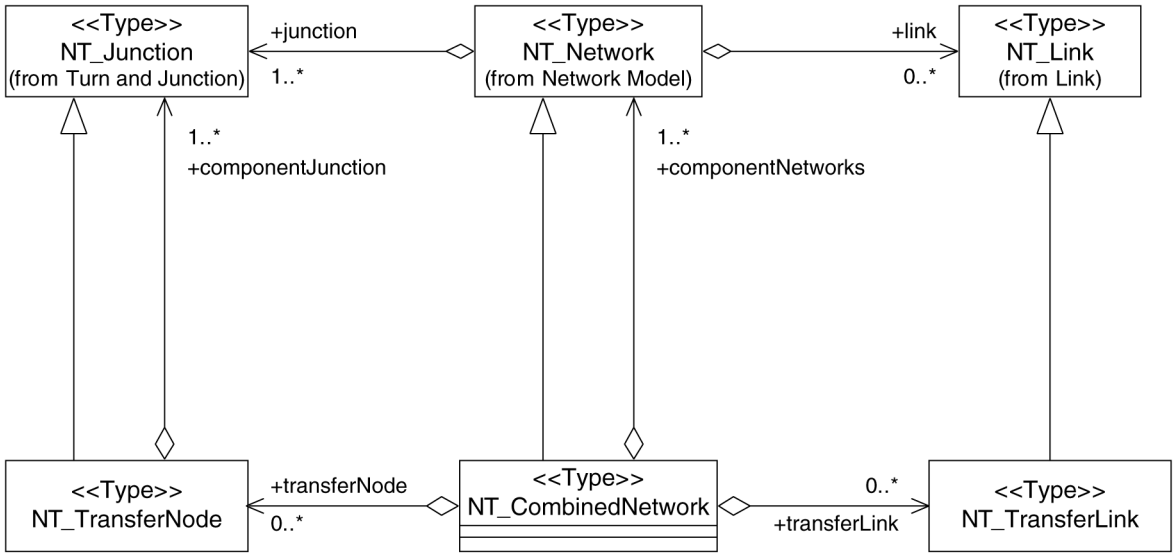

Figure 119 — Combined Networks

9.8.2 NT_CombinedNetwork

9.8.2.1 Semantics

The type “NT_CombinedNetwork” models networks built from smaller component networks through the use of transfer links and transfer nodes, which are described below. If the component networks are all of the same mode, the combined network will be single mode. If not, the transfers would be mode changes within the combined multimodal network. The UML for NT_CombinedNetwork is given in Figure 120.

9.8.2.2 Role: componentNetworks : NT_Network

The association role “componentNetworks” specifies the simpler networks from which this one is created.

```
NT_CombinedNetwork :: componentNetworks : NT_Network
```

9.8.2.3 Role: transferLink : NT_TransferLink

The association role “transferLink” specifies the transfer links used by the combined network to link other networks from its component network list.

```
NT_CombinedNetwork :: : transferLink : NT_TransferLink
```

9.8.2.4 Role: transferNode : NT_TransferNode

The association role “transferNode” specifies the transfer nodes used by the combined network to join other networks from its component network list.

```
NT_CombinedNetwork :: : transferNode : NT_TransferNode
```

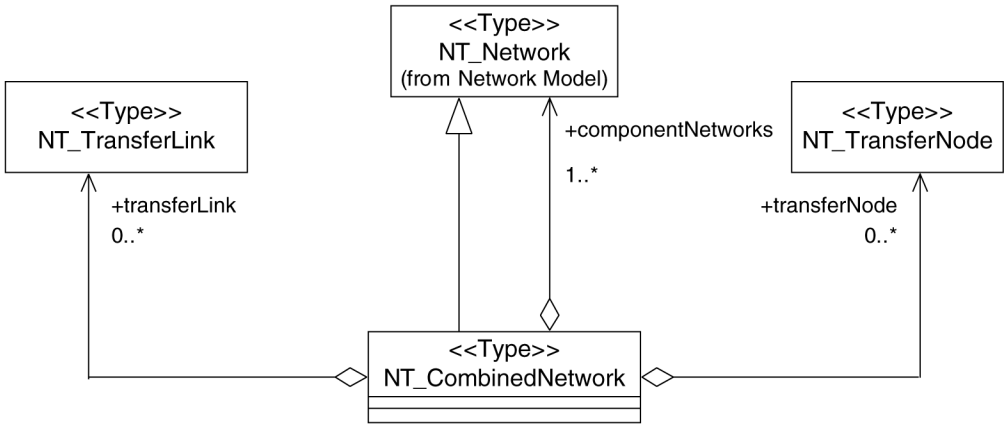

Figure 120 — Context Diagram: NT_CombinedNetwork

9.8.3 NT_TransferNode

9.8.3.1 Semantics

A transfer Node is a node (junction) in a combined network that is the union of several component junctions. Its links and turns contain all the links and turns of the component junctions plus transfers that move from a link in one component network to a link in one of the other component networks. The combined network can be recast (alternately represented) by a single network in which all component junctions of a single transfer node have been united into a single node (of type junction), whose links consist of all the component links and whose turns consist of all the component turns combined with the transfers. The UML for NT_TransferNode is given in Figure 121.

9.8.3.2 Role: componentJunction : NT_Junction

The association role “componentJunction” specifies the simpler junctions from which this one is created.

```
NT_ TransferNode :: componentJunction : NT_Junction
```

9.8.3.3 Role: turn : NT_Transfer

The association role “turn” specifies the turns (subtyped as NT_Transfers) that are located at this transfer node.

```
NT_ TransferNode :: turn : NT_Transfer
```

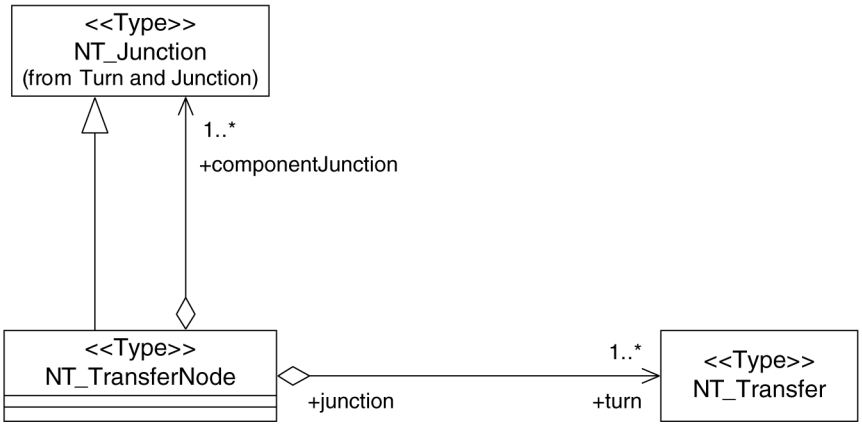

Figure 121 — Context Diagram: NT_TransferNode

9.8.4 NT_Transfer

The type “NT_Transfer” is used to represent a turn that occurs at a transfer node and which has entry and exit links connected to separate component junctions. The UML for NT_Transfer is given in Figure 122.

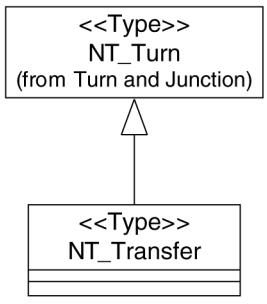

Figure 122 — Context Diagram: NT_Transfer

9.8.5 NT_TransferLink

The type “NT_TransferLink”, a subtype of NT_Link, is used to represent links in a combined network whose boundary nodes are in different component networks. The UML for NT_TransferLink is given in Figure 123.

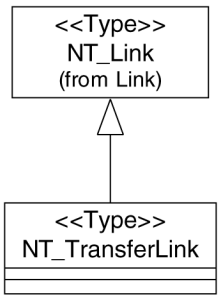

Figure 123 — Context Diagram: NT_TransferLink

10 Basic implementation packages

10.1 Package: Feature Data Model

10.1.1 Semantics

The classes in this package are used as stand-ins for any implementation of features as defined in ISO 19109. The summary of this package is given in Figure 124.

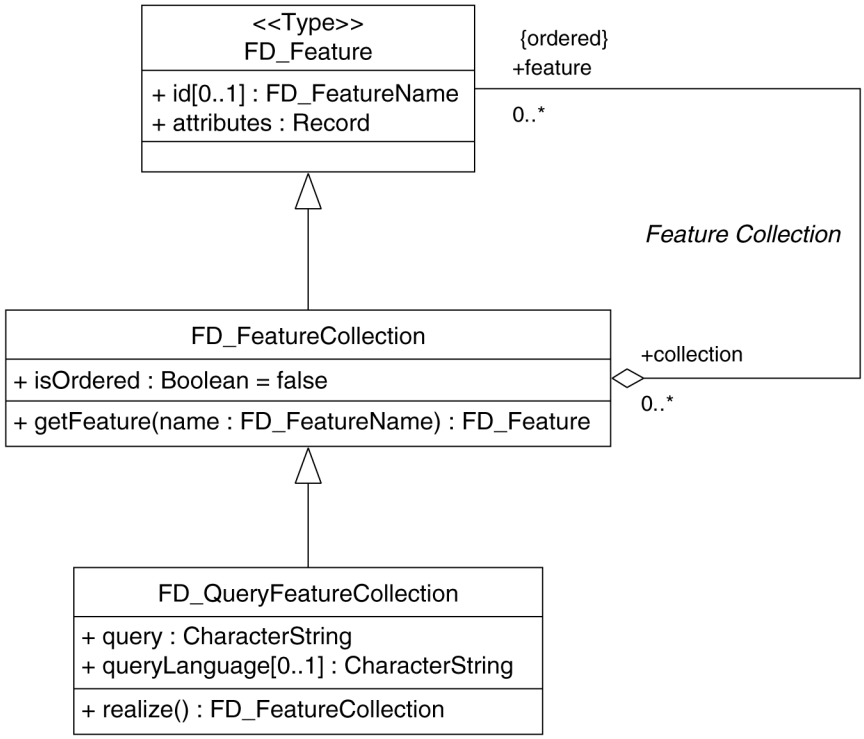

Figure 124 — Feature data classes

10.1.2 FD_Feature

10.1.2.1 Semantics

The type “FD_Feature” is used to represent an implementation of features. The UML for FD_Feature is given in Figure 125.

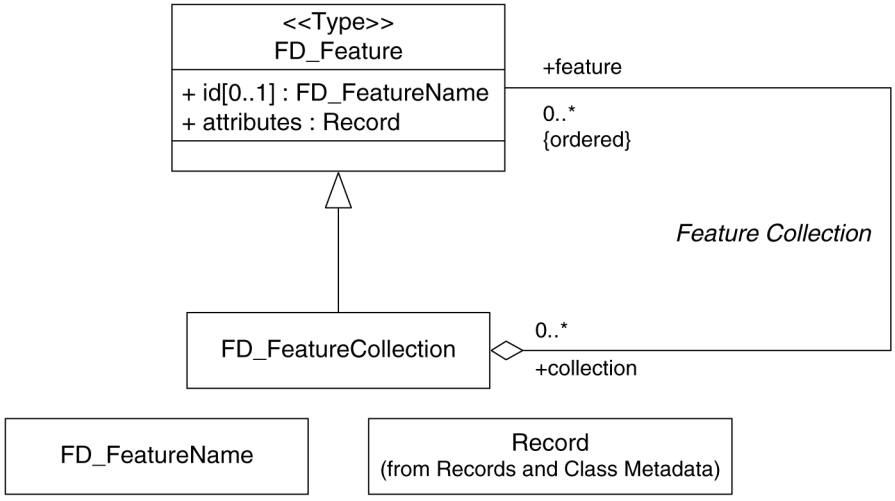

Figure 125 — Context Diagram: FD_Feature

10.1.2.2 Attribute: id[0..1] : FD_FeatureName

The attribute “id” is the local name, or identity of the feature. This will usually be listed in the name space for the container feature collection for this feature.

```
FD_Feature:: id[0..1] : FD_FeatureName
```

10.1.2.3 Attribute: attributes : Record

The attribute “attributes” is a record containing all the attributes of this feature. This mechanism is used to assure that any combination of attributes can be associated with a feature. This will support most type systems semantics normally used for feature representations, including, strong, weak, dynamic, and untyped.

```
FD_Feature:: attributes : Record
```

10.1.3 FD_FeatureCollection

10.1.3.1 Semantics

The type “FD_FeatureCollection” is the basic class for feature collections The UML for FD_FeatureCollection is given in Figure 126.

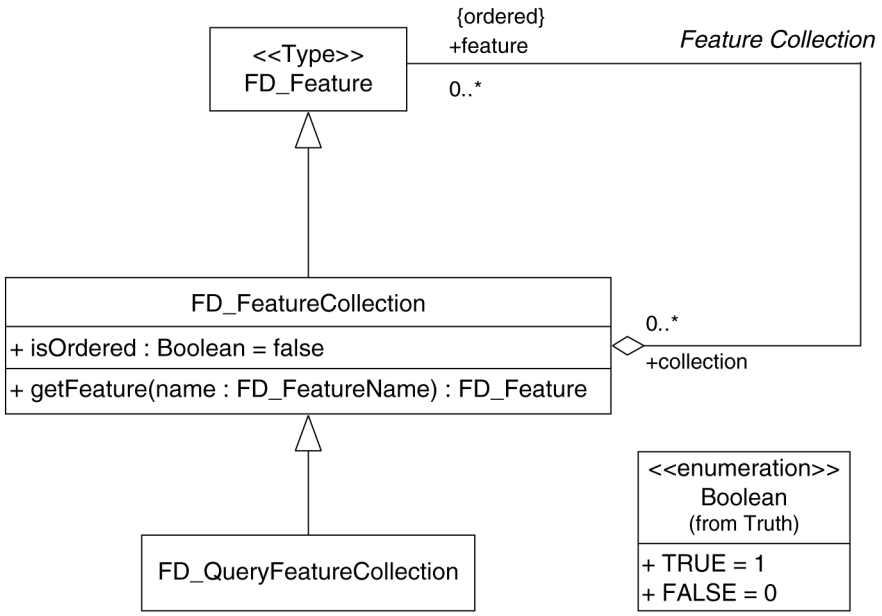

Figure 126 — Context Diagram: FD_FeatureCollection

10.1.3.2 Attribute: isOrdered : Boolean = false

The Boolean attribute “isOrdered” indicates whether there is any meaning in the order of this collection’s “feature” role.

```
FD_FeatureCollection :: isOrdered : Boolean = false
```

10.1.3.3 Role: feature[0..*] : FD_Feature

The association role “feature” aggregates the features in this collection. The ordering may or may not be of semantic importance depending on the “ordered” attribute of the feature collection.

```
FD_FeatureCollection :: feature[0..*] : FD_Feature
```

10.1.3.4 Operation: getFeature

The operation “getFeature” retrieves the feature on the association role “feature” having a given name.

```
FD_FeatureCollection :: getFeature(name : FD_FeatureName) : FD_Feature
```

10.1.4 FD_QueryFeatureCollection

10.1.4.1 Semantics

The type “FD_QueryFeatureCollection” specifies a feature collection that is defined by a query. The association “Feature” might not be populated; but when it is, it represents the query results at the time the query was last executed. The UML for FD_QueryFeatureCollection is given in Figure 127.

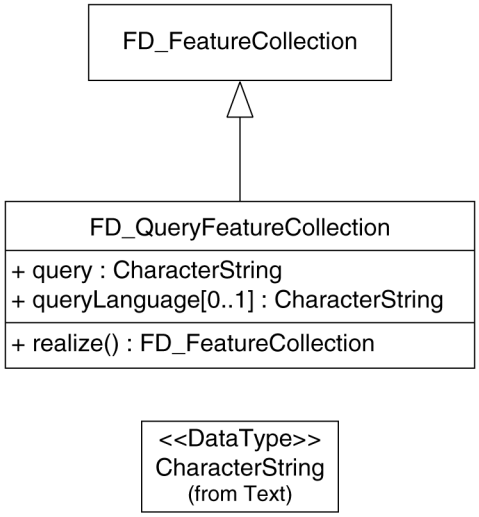

Figure 127 — Context Diagram: FD_QueryFeatureCollection

10.1.4.2 Attribute: query : CharacterString

The attribute “query” is the query, represented by a character string that defines containment in this collection.

```
FD_QueryFeatureCollection :: query : CharacterString
```

10.1.4.3 Attribute: queryLanguage[0..1] : CharacterString

The attribute “queryLanguage” is the query language used by the query.

```
FD_QueryFeatureCollection :: queryLanguage[0..1 : CharacterString
```

10.1.4.4 Operation: realize

The operation “realize” populates the Feature Collection association with the features that satisfies this query. Normally, this will reuse this FD_QueryFeatureCollection, but that is dependent on the semantics of the implementation.

```
FD_QueryFeatureCollection :: realize() : FD_FeatureCollection
```

10.1.5 FD_FeatureName

10.1.5.1 Semantics

The class “FD_FeatureName” is used to specify the name of a feature or feature collection for use in target lists for navigation requests. The UML for FD_FeatureName is given in Figure 128.

10.1.5.2 Attribute: id : GenericName

The attribute “id” contains the identity of the feature in the name space used for features in the datastore.

```
FD_FeatureName :: id : GenericName
```

10.1.5.3 Attribute: type : TypeName

The attribute “type” contains the type name for the feature in the name space used for feature types in the datastore.

```
FD_FeatureName :: type : TypeName
```

10.1.5.4 Operation: getObject

The operation “getObject” retrieves the feature associated with this feature name.

```
FD_FeatureName :: getObject() : FD_Feature
```

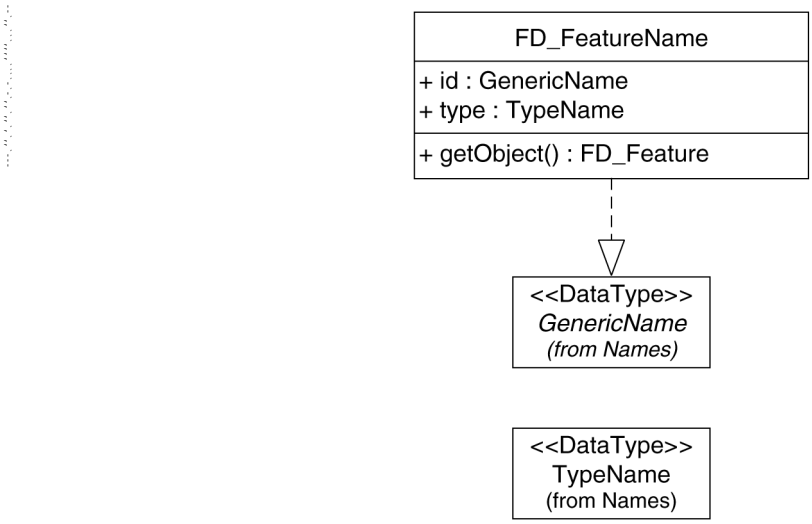

Figure 128 — Context Diagram: FD_FeatureName

10.2 Package: New Basic Types

10.2.1 Semantics

The package “New Basic Types” loosely defines some basic types that this International Standard requires. These are generic types that exist in various forms, and this International Standard does not place any special requirements on them.

10.2.2 VoiceStream

The class “VoiceStream” is any binary array representing an encoded voice. This is used for user interfaces. The UML for VoiceStream is given in Figure 129.

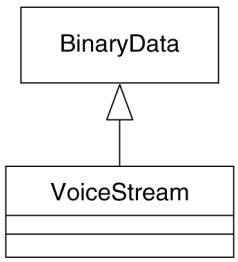

Figure 129 — Context Diagram: VoiceStream

10.2.3 BinaryData

The class “BinaryData” is any data encoded as a binary stream. The UML for BinaryData is given in Figure 130.

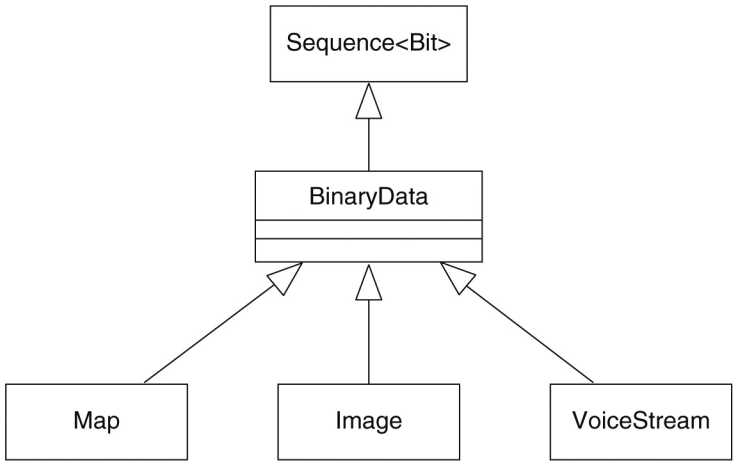

Figure 130 — Context Diagram: BinaryData

10.2.4 Map

The class “Map” is any encoded data representing a map, usually as an image, such as a JPEG or GIF file. The UML for Map is given in Figure 131.

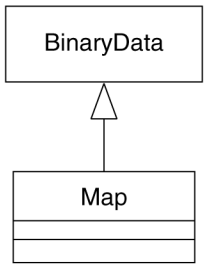

Figure 131 — Context Diagram: Map

10.2.5 Image

The class “Image” is any encoded data representing an image, such as a JPEG or GIF file. The UML for Image is given in Figure 132.

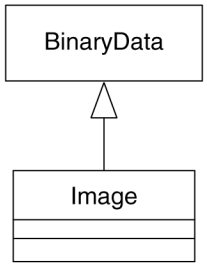

Figure 132 — Context Diagram: Image

Annex A
(normative)

Abstract test suite

A.1 Semantics

Conformance to this International Standard shall consist of either service conformance or data conformance.

Data conformance includes the usage of data types from application schemas (“application type”) that are mappable into types in this International Standard (“standard type”). In this context, “mappable” means that there is a correspondence between the standard types in the appropriate part of this International Standard, and the application types of the application schema in such a way that each standard type can be considered as a supertype of the application type designated by the correspondence. This means that an application type corresponding to a standard type contains sufficient data to recreate that standard type’s information content.

Service conformance includes both the consistent use of message-based request–response interfaces and data conformance for the message packages used by those interfaces.

A.2 Data Types

A.2.1 Common Data Type

A.2.1.1 TK_Position

Most of the modules in this conformance annex require the use of some application class for TK_Position. Since this is a union type, a conformant application schema implementation or profile need only implement a viable subset of the optional representations.

a) Test Purpose: to verify an adequate set of application classes for the expressions of position. The conformance class shall support at least one of the forms listed in the references below.

b) Test Method: Inspect the documentation of the application schema or profile and exhibit the required correspondence.

c) References: ISO 19107, ISO 19108, ISO 19109, ISO 19112, and the subclauses specified below from this International Standard.

1)	Temporal Data:	ISO 19108
2)	TK_Position:	6.2.2
3)	TK_PositionType:	6.2.6
4)	DirectPostion:	ISO 19107
5)	SI_LocationName:	ISO 19112
6)	FD_FeatureName:	ISO 19109
7)	LR_PositionExpression:	6.6.2

8) NT_NetworkPosition: 9.6.3

9) AD_AbstractAddress: 8.2.3

10) phone::CharacterString

A.2.2 Address

a) Test Purpose: to verify an adequate set of application classes for the expressions of Addresses.

b) Test Method: Inspect the documentation of the application schema or profile and exhibit the required correspondence.

c) References: ISO 19133; 8.2 and 8.3 of this International Standard, including the subclauses specified below.

1) AD_Address: 8.2.2

2) AD_AbstractAddress: 8.2.3

3) AD_AddressElement: 8.3.2

4) AD_Addressee: 8.3.3

5) AD_StreetIntersection: 8.3.4

6) AD_Street: 8.3.5

7) AD_PostalCode: 8.3.6

8) AD_StreetLocation: 8.3.7

9) AD_PhoneNumber: 8.3.8

10) AD_NamedPlace: 8.3.9

11) AD_StreetAddress: 8.3.10

12) AD_NamedPlaceClassification: 8.3.10

13) AD_Building: 8.3.12

14) AD_MuniQuadrant: 8.3.13

15) AD_RegionCode: 8.3.14

16) AD_NumberRange: 8.3.15

17) AD_ListNamedPlaces: 8.3.16

A.2.3 Linear Reference System

a) Test Purpose: to verify an adequate set of application classes for the expressions of positions within a Linear Reference System.

b) Test Method: Inspect the documentation of the application schema or profile and exhibit the required correspondence.

c) References: ISO 19108; ISO 19133; 6.6 of this International Standard, including the subclauses specified below.

1)	Temporal Data:	ISO 19108
2)	LR_PositionExpression:	6.6.2
3)	LR_LinearReferenceMethod:	6.6.3
4)	LR_OffsetDirection:	6.6.4
5)	LR_ReferenceMarker:	6.6.5
6)	LR_OffsetReference:	6.6.6
7)	LR_Feature:	6.6.7
8)	LR_Element:	6.6.8
9)	LR_OffsetExpression:	6.6.9

A.2.4 Network

a) Test Purpose: to verify an adequate set of application classes for the expressions within a Network Data Model.

b) Test Method: Inspect the documentation of the application schema or profile and exhibit the required correspondence.

c) References: ISO 19108; ISO 19133; Clause 9, through 9.7 of this International Standard, including the subclauses specified below.

1)	Temporal Data:	ISO 19108
2)	NT_Network:	9.2.1
3)	NT_WayPoint:	9.2.2
4)	NT_WayPointList:	9.2.3
5)	NT_Turn:	9.3.2
6)	NT_TurnDirection:	9.3.3
7)	NT_Junction:	9.3.4
8)	NT_JunctionType:	9.3.5
9)	NT_AngularDirection:	9.3.6
10)	NT_Constraint:	9.4.2

11) NT_VehicleConstraint: 9.4.3

12) NT_TemporalConstraint: 9.4.4

13) NT_LaneConstraint: 9.4.5

14) NT_Vehicle: 9.4.6

15) NT_Advisory: 9.4.7

16) NT_SpatialRelation: 9.4.8

17) NT_AdvisoryCategory: 9.4.9

18) NT_AdvisoryElement: 9.4.10

19) NT_ExitAssociation: 9.4.11

20) NT_AdvisoryDirection: 9.4.12

21) NT_AdvisoryDistance: 9.4.13

22) NT_AdvisorySpatialRelation: 9.4.14

23) NT_Link: 9.5.2

24) NT_RouteSegmentCategory: 9.5.3

25) NT_LinkPosition: 9.6.2

26) NT_NetworkPosition: 9.6.3

27) NT_Route: 9.7.2

28) NT_RouteSummary: 9.7.3

29) NT_Maneuver: 9.7.4

A.2.5 Combined Network

a) Test Purpose: to verify an adequate implementation classes for the expressions of a Combined Network Data Model.

b) Test Method: Inspect the documentation of the application schema or profile and exhibit the required correspondence.

c) References: ISO 19108; ISO 19133; Clause 9 of this International Standard, including the subclauses specified below.

1) Network: A.2.4

2) NT_CombinedNetwork: 9.8.2

3) NT_TransferNode: 9.8.3

4) NT_Transfer: 9.8.4

5) NT_TransferLink: 9.8.5

A.3 Services

A.3.1 Tracking service

A.3.1.1 Interfaces

a) Test Purpose: to verify the use of the appropriate interfaces for a tracking service.

b) Test Method: Inspect the documentation of the service interface to verify the use of interfaces defined in the references below.

c) References: ISO 19108; ISO 19133; 6.2 and 6.3 of this International Standard, including the subclauses specified below.

1) Temporal Data:	ISO 19108
2) TK_Position:	A.2.1
3) TK_MobileSubscriber:	6.2.3
4) TK_TrackingLocation:	6.2.4
5) TK_TrackingService:	6.2.5
6) TK_TrackingLocationSequence:	6.2.7
7) TK_Trigger:	6.2.8
8) TK_PeriodicTrigger:	6.2.9
9) TK_TransitionTrigger:	6.2.10
10) TK_TransitionType:	6.2.11
11) TK_TrackingLocationMetadata:	6.2.12
12) TK_Transition:	6.2.13
13) TK_QualityOfPosition:	6.2.14
14) TK_Accuracy:	6.2.15
15) TK_AccuracyType:	6.2.16
16) TK_AccuracyStatement:	6.2.17
17) EG_PointEstimateCircle:	6.3.2
18) EG_PointEstimateEllipse:	6.3.4
19) EG_PointEstimateArc:	6.3.5
20) EG_PointEstimate:	6.3.2
21) EG_PointEstimateSphere:	6.3.6
22) EG_PointEstimateEllipsoid:	6.3.7

A.3.2 Location Transformation

a) Test Purpose: to verify the use of the appropriate interfaces for a Location Transformation service

b) Test Method: Inspect the documentation of the service interface to verify the use of interfaces defined in the references below.

c) References: ISO 19108 and ISO 19133, 6.4 and 6.5 of this International Standard, including the subclauses specified below.

1)	Temporal Data:	ISO 19108
2)	LT_LocationTransformationService:	6.4.2
3)	MC_MeasurePosition:	6.5.2
4)	MC_CoordinateSystem:	6.5.3
5)	MC_CoordinateReferenceSystem:	6.5.4

A.3.3 Navigation Services

a) Test Purpose: to verify the use of the appropriate interfaces for a Navigation service.

b) Test Method: Inspect the documentation of the service interface to verify the use of interfaces defined in the references below.

c) References: ISO 19108; ISO 19133; 7.3, 7.4 and 7.5 of this International Standard, including the subclauses specified below.

1)	Temporal Data:	ISO 19108
2)	NS_NavigationService:	7.3.2
3)	NS_RouteRequest:	7.3.3
4)	NS_Instruction:	7.3.4
5)	NS_InstructionList:	7.3.5
6)	NS_RouteResponse:	7.3.6
7)	NS_CostedTurn:	7.3.7
8)	NS_RenderingService:	7.3.8
9)	NS_RenderingRequest:	7.3.9
10)	NS_RenderingResponse:	7.3.10
11)	NS_RenderingType:	7.3.11
12)	NS_CostedLink:	7.3.12
13)	NS_CostFunctionCode:	7.3.13
14)	NS_RouteRequestType:	7.3.14

Annex B
(informative)

Directed weighted graphs and their algorithms

B.1 Introduction

Much of this International Standard was written with a programming approach in mind. While it is not the intent of this International Standard to specify the particular algorithms used to support the service interfaces described here, it should aid in the understanding of those services if the algorithms envisaged by the authors are described.

Most of these algorithms are based on directed weighted graphs. While graphs have an obvious mapping to physical features, they have other applications as well. A graph in this context is a set of places or states called “nodes” that are connected by 1-way channels called “links”. Two-way channels are described as pairs of links, one in each direction. In a 1-dimensional topological complex, the nodes and directed edges form a directed graph. The weights of a graph are numbers assigned to the links. These weights are usually representations of some sort of marginal cost – either in distance, time or money – incurred in traversing or executing that link. The semantics of cost function require that the weights be non-negative numbers. Graph theory in general is independent of semantics, and can treat with negative weights in some particularly limited cases. Nodes are not costed.

The following sections define this structure in a more formal manner and describe problems and some of their potential solutions.

B.2 Directed weighted graphs

A directed graph consists of two sets, ***N*** and ***E***. The set ***N*** contains the “nodes” of the graph. ***E*** contains the directed links of the graph and is a subset of the cross product ***N*** × ***N*** called the edges of the graph. Each edge **(*n*$_1$, *n*$_2$)** is a mechanism to traverse from node ***n*$_1$** to node ***n*$_2$**. Thus, the graph can be thought of as the states and transitions of a finite state machine.

A **path** in the graph consists of a sequence of nodes (***n*$_0$**, ***n*$_1$**, ***n*$_2$**, ***n*$_3$**, ***n*$_4$**, … , ***n***$_k$) such that for each i with $0 \leqslant i < k$, (***n***$_i$, ***n***$_{i+1}$) ∈ ***E***. If a graph is thought of as a description of a finite state machine, then a path is a legal description of a legal program for that machine. Since programs are most commonly thought of as sequences of actions, a path is alternatively represented by a sequence of edges {(***n***$_i$, ***n***$_{i+1}$) | $0 \leqslant i < k$ and (***n***$_i$, ***n***$_{i+1}$) ∈ ***E*** }. The trivial paths contain only one node and no edge and are thus of the form (***n*$_0$**).

A cycle is a path (***n*$_0$**, ***n*$_1$**, ***n*$_2$**, ***n*$_3$**, ***n*$_4$**, … , ***n***$_k$) with ***n*$_0$** = ***n***$_k$. Thus, a cycle is a path that begins and ends at the same node.

A weighted graph is one in which each edge (***n***$_i$, ***n***$_j$) is assigned a number or “weight” ***w***$_{i,j}$. Since the weights are often considered as representing costs, most graphs will have ***w***$_{i,j} > 0$ for each edge (***n***$_i$, ***n***$_j$). If such a restriction holds, the graph is positively weighted. The weight of a nontrivial path is the sum of the weights of its links, so:

$$\text{weight}\,(\boldsymbol{n}_0, \boldsymbol{n}_1, \ldots, \boldsymbol{n}_k) = \sum_{i=0}^{k} \text{weight}\,(\boldsymbol{n}_i, \boldsymbol{n}_{i+1}).$$

B.3 Shortest path problems

The most common problems encountered in this International Standard are variants of the "shortest path problem". The problem statement is as follows.

Given a source node ***a*** and a target node ***b***

find a path $\boldsymbol{P} = (\boldsymbol{n_0}, \boldsymbol{n_1}, \boldsymbol{n_2}, \boldsymbol{n_3}, \boldsymbol{n_4}, \ldots, \boldsymbol{n_k})$

of minimum weight with $\boldsymbol{a} = \boldsymbol{n_0}$ and $\boldsymbol{b} = \boldsymbol{n_k}$.

The criteria for the existence of a solution to the shortest path problem in a weighted directed graph are as follows.

— The graph is connected, i.e. there is a path from each node to every other node in the graph.

— No cycle has a negative weight.

The first criterion ensures that some path exists, and the second ensures that a minimum exists.

There are several algorithms for this problem, but all of them are based on the same concept: Label the graph based on successively better approximations to the distance from the source node. When the destination ***b*** is labelled for the last time (according to a stopping criteria of the algorithm), its label is the distance from ***a*** to ***b***. By keeping track of the predecessor of each node, the path that realizes that distance can be traced backwards for ***b***. These algorithms are all based on a trivial observation which is true for non-negative weights: If a path **p** optimizes travelling from ***a*** to ***b***, then it also optimizes travelling from ***a*** to ***n***, for any ***n*** on that path. This means that **p** is an ever-longer sequence of shortest paths:

$\{(\boldsymbol{n_0}, \boldsymbol{n_1}), (\boldsymbol{n_0}, \boldsymbol{n_1}, \boldsymbol{n_2}), (\boldsymbol{n_0}, \boldsymbol{n_1}, \boldsymbol{n_2}, \boldsymbol{n_3}), (\boldsymbol{n_0}, \boldsymbol{n_1}, \boldsymbol{n_2}, \boldsymbol{n_3}, \boldsymbol{n_4}), \ldots, (\boldsymbol{n_0}, \boldsymbol{n_1}, \boldsymbol{n_2}, \boldsymbol{n_3}, \boldsymbol{n_4}, \ldots, \boldsymbol{n_k})\}$

Dijkstra's algorithm is the simplest of the type. The essentials of the algorithm are:

```
Begin with a path sequence P containing the trivial path (a). This is the
      shortest path (length 0) starting at a.
For all neighbors of endpoint of the paths in the path sequence, look at
      links exiting this point and create a candidate path from the current
      path and the new link. You need not visit nodes already in paths
      in P.
Take the minimum length path from among the candidates and add it to P.
P now contains the |P| (number of paths in P) shortest paths beginning at a.
Iterate through the 3rd step until the path added ends at b. This newest
      path is the shortest path from a to b.
```

The drawback to Dijkstra's algorithm is the building of the path sequence ***P***. It tends to grow in all directions at once forming an ever-growing network neighbourhood of **a**. The simplest technique to improve this is to bias the growth of ***P*** towards paths whose endpoints get nearer and nearer to **b**. This is done with an estimator function $d: \boldsymbol{N} \times \boldsymbol{N} \rightarrow \boldsymbol{R}^{+}$ that estimates the distance between nodes. Instead of sorting ***P*** on "weight(**p**)", you sort on "weight(***p***) + d(end(**p**), **b**)". If you know that d always underestimates the distance between nodes along paths, then the first path to reach **b** is the shortest path as in Dijkstra's algorithm. This is an instance of the "A*star" search algorithm.

There are other variants of this "path tree search" mechanism, but they all essentially do the same thing, search for paths starting at the source node. They all put the same requirements on the graph, but they do use widely variant mechanisms for storing that graph during calculations.

B.4 Mapping real road networks into positively weighted directed graphs

The naïve assumption is that a road network is a weighted graph. If the road network were mapped directly to the graph, node for node and directed edge for link, then we would have to interpret the properties of the graph in terms of the properties of the road network. Unfortunately, directed graphs have several properties that do not always reflect the reality of a road network.

First, in a graph problem, traversing a node is always free. The naïve interpretation would then require that all turns be free. In a time-weighted graph, this would not always be the case, since time would have to be allowed for delays caused by traffic controls (such as traffic lights).

Second, the directed graph assumes that the choice of the exit from a node is dependent only on the node, and not how that node was approached. This is not consistent with turn restrictions. For example, if a sign "no left turn" appears at some entrance to an intersection where a left turn would otherwise be legal, then vehicles approaching from that direction are not allowed use of that exit link. A similar problem arises for "U-turns" which are executed as two consecutive lefts. If the turns are independent, then the U-turn is always legal.

Third, directed graphs do not allow decision points along the link, making U-turns other than at intersections difficult to model.

There are two basic methods to work around this type of problem. The naïve one is to modify the algorithm to match reality. The less obvious one is to modify the network to match the definitions. In fact, most of the algorithm modifications are equivalent to network modifications. For the purposes of this analysis, we can assume that the networks are modified to match the definition of a weighted graph given above.

To understand how this is done, consider the finite state machine metaphor. In such a machine, the graph nodes are the decision points and the links are the uninterrupted execution of those decisions.

B.5 Choosing decision points

The decision points in a road network are the positions at which several alternatives exist and the driver must commit himself to one of them.

B.6 Turns and maneuvers

As a vehicle approaches a junction, the driver must make a choice from a set of combinations that will take him beyond that junction. If there are no restrictions placed upon him, then this is equivalent to choosing the exit link at the next intersection (which is the set of turns). If there are restrictions, especially in a complex junction consisting of multiple intersections, then the maneuvers are sequences of turns (and their intervening links).

The easiest way to model this is to place decision-point nodes at the end and beginning of all road segments long enough to remove historical restriction. At each ending decision node, the exiting graph links would consist of the maneuvers legal at that point. Each maneuver ends at the beginning point of its last link. If non-intersection U-turns are possible, then they can be placed at the beginning decision node of each road link as a maneuver ending at the ending decision point of the opposite directed-edge road link (possibly a different edge if the road is divided and the other lanes are digitized separately). Non-traversable directed edges would not appear as links in the navigation graph.

Annex C
(informative)

View of Standard in terms of RM-ODP Services

C.1 Engineering viewpoint

The basic Engineering Viewpoint assumption is that the services described in this International Standard will be made available on the web to be accessed by mobile devices, whose web connection may be transient, in a manner similar to permanently on-web clients. The exception is that the mobile client can either update or request an update of its own geographic location at one or more times during the process of the service interaction. There are no specific requirements on the network platform, and the interface and data definitions in this International Standard are platform neutral.

A web-resident, and persistent, proxy application for the mobile client is required to make this possible. This proxy acts as a device transformer for messages and embedded data flowing between the service and the mobile client. The interface between the mobile client and the on-web proxy is not within the scope of this International Standard and is covered by International Standards written by and within ISO/TC 204. This conceptual architecture is shown in Figure C.1. In that diagram, thin and medium client nodes appear at the top of the diagram. The other nodes on the network are persistent, on-web services available to mobile clients through their “Proxy Application and Device Transformer”. Services specifically defined in this International Standard are marked as such. Other services in Figure C.1 are for example, but may represent functionality required for the marked service. For example, “Gazetteer Service” should be compliant with ISO 19112.

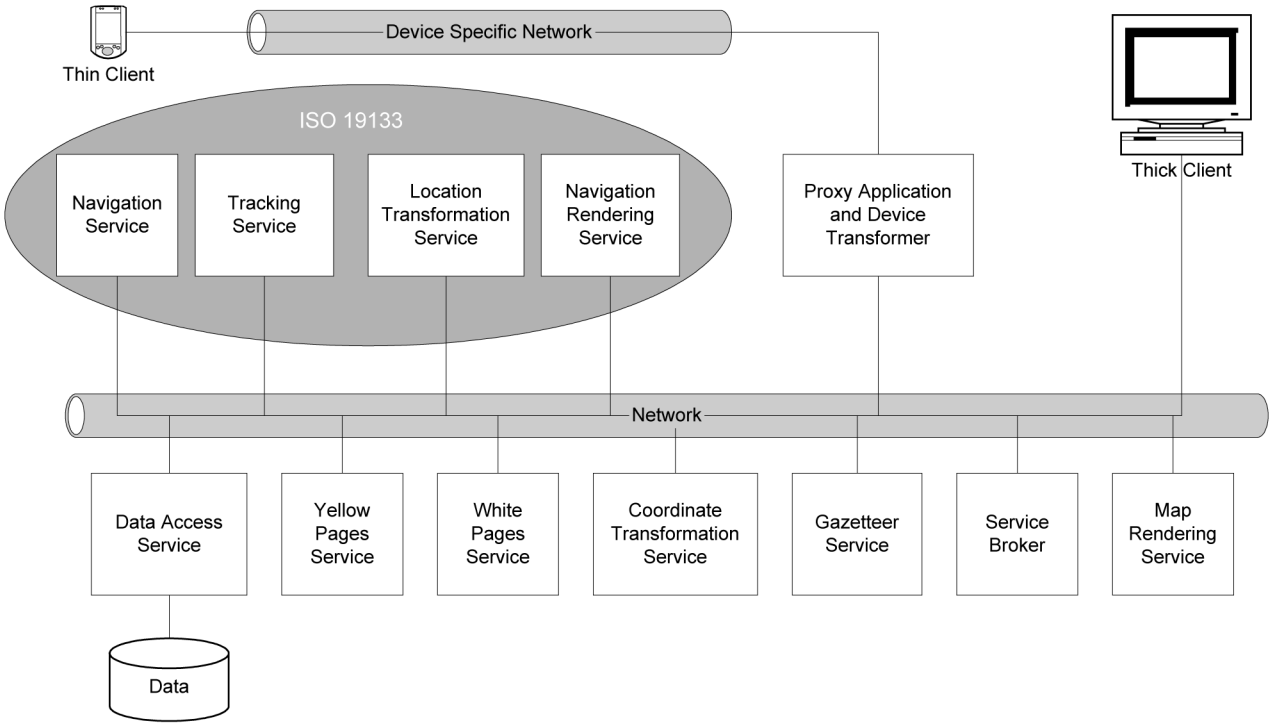

Figure C.1 — Conceptual architecture equating mobile and non-mobile services

The second assumption is that the state of the mobile client will be maintained by the client application or by its on-web proxy application. This means that all requests for services will be totally encapsulated in a request–response pair. The operations will all be prototypically represented as

```
<serviceType> :: <svrOperation>(<serviceRequest>) : <serviceResponse>
```

Thus, we have a service model based on sets of three basic types:

— a service type (listing of service operations);

— a set of service request data types associated with some number of operations;

— a set of service response data types associated with some number of operations.

The data types will have a core set of required components and another set of optional components that can affect the outcome and semantics of the operations. For example, the simplest form of navigation requires simply a “from target position” and a “to target position”, but can be modified by sending an optional description of a different cost function.

C.2 Information viewpoint

The packages in this International Standard specify components of the information model for the services defined here or related to those defined here. Each information model component is a type that will be used in one or more of the services defined in this International Standard. In many cases, they will also be output types for services that support the services defined here.

C.3 Service taxonomy

All services in this International Standard are represented by <<Type>> stereotyped classifiers and are processing services.

Bibliography

[1] National Cooperative Highway Research Program (NCHRP), 1974, *Highway Location Reference Methods*, Synthesis of Highway Practice 21, National Academy of Sciences, Washington, D.C.

[2] ABADI, M. and CARDELLI, L., *A Theory of Objects*, Springer-Verlag, New York, 1996

[3] AHUJA, R.K., MAGNANTI, T.L. and ORLIN, J.B., *Network Flows: Theory, Algorithms and Applications*, Englewood Cliffs, NJ: Prentice Hall, 1993

[4] BELLMAN, R.E., 1958, *On a routing problem*, Quarterly of Applied Mathematics, Vol. 16, pp. 87-90

[5] CHERKASSKY, B.V., GOLDBERG, A.V. and RADZIK, T., 1993, *Shortest Paths Algorithms: Theory and Experimental Evaluation*, Technical Report 93-1480, Computer Science Department, Stanford University

[6] CHRISTENSEN, E., CURBERA, F., MEREDITH, G. and WEERAWARAMA, S., 15 March 2001, *Web services Description Language (WSDL) 1.1*, W3C Note, <http://www.w3.org/TR/wsdl>

[7] DIJKSTRA, E., 1959, A *note on two problems in connection with graphs*, Numerische Mathematik, Vol. 1, pp. 269-271

[8] FORD, L.R. Jr., 1956, *Network Flow Theory*, Paper P-923, RAND Corporation, Santa Monica, California

[9] HOROWITZ, A.J., 1991, *Delay-Volume Relations for Travel Forecasting: Based on the 1985 Highway Capacity Manual,* Federal Highway Administration, Washington, D.C., March 1991

[10] SCARPONCINI, P., 2002, *Generalized Model for Linear Referencing in Transportation*, GeoInformatica, Vol. 6, pp. 35-55

[11] SUH, S., PARK, C.H. and KIM, T.J., 1990, *A Highway Capacity Function in Korea: Measurement and Calibration*, Transportation Research, 24A, pp. 176-186

[12] YOU, J. and KIM, T.J., 2000, *Development and Evaluation of a Hybrid Travel Time Forecasting Model*, Transportation Research C. 8, pp. 231-256. [Also published as a chapter in THILL, J.-C., (ed), Geographic Information Systems in Transportation Research, 2000, Elsevier Science Ltd, Oxford, UK.]

[13] ZHAN, F.B. and NOON, C.E., 2000, *A Comparison Between Label-Setting and Label-Correcting Algorithms for Computing One-to-One Shortest Paths*, Journal of Geographic Information and Decision Analysis, Vol. 4, No. 2, pp. 1-13

[14] ZHAN, F.B., and NOON, C.E., 1996, *Shortest Path Algorithms: An Evaluation Using Real Road Networks.* Transportation Science, 32, pp. 65-73

[15] ZHAN, F.B., 1997, *Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedure*, Journal of Geographic Information and Decision Analysis, Vol. 1, No. 1, pp. 69-82. Available at <http://publish.uwo.ca/~jmalczew/gida_1/Zhan/Zhan.htm>

[16] ISO/IEC 11404, *Information technology — Programming languages, their environments and system software interfaces — Language-independent datatypes*

[17] ISO 14825, *Intelligent transport systems — Geographic Data Files (GDF) — Overall data specification*

ISO 19116, *Geographic information — Positioning services*

ICS 35.240.70

Price based on 139 pages