Acta Polytechnica CTU Proceedings **58**:33–49, 2026
https://doi.org/10.14311/APP.2026.58.0033
© 2026 The Author(s). Licensed under a CC-BY 4.0 licence
Published by the Czech Technical University in Prague

RAILTOPOMODEL AND RAILSYSTEMMODEL COMMONALITIES AND DIFFERENCES

Adam Hlubuček

Czech Technical University in Prague, Faculty of Transportation Sciences, Department of Transport Telematics, Konviktská 20, 110 00 Prague 1, Czech Republic

correspondence: hlubuada@fd.cvut.cz

Abstract. This article focuses on comparing two generic data models of railway infrastructure. The first of these models is the RailTopoModel. Although originally created under the patronage of the UIC, in recent years, its development has been carried out in close coordination with the development of the railML exchange data format. The second model is the RailSystemModel. In fact, it is an extension of the original UIC RailTopoModel specifications to include some substantive aspects of the description of railway infrastructure and its surroundings. It remains under the management of the International Union of Railways (UIC). The versions compared are RailTopoModel 1.5 (released together with railML 3.3) and RailSystemModel 1.2 (the only stable version of RailSystemModel so far). The focus is on aspects that appear in both data models simultaneously, expressed in each case in a different way, rather than on aspects that have been added to each of the data models.

Keywords: Data description, RailSystemModel, RailTopoModel, railway infrastructure.

1. Introduction

With the development of software applications using railway data, data models for describing railway infrastructure are also gaining importance. As a response to the emergence of diverse approaches to railway infrastructure modelling, various tendencies have arisen to standardise these processes.

One of the significant initiatives of this kind was the creation of the RailTopoModel, a generic data model of railway infrastructure. The beginnings of this initiative can be traced back to 2013, when the Feasibility Study of the UIC RailTopoModel and Data Exchange Format [1] was prepared. This study included, among other things, an analysis of existing data models of railway infrastructure and the definition of requirements for a new data model, as well as for the related exchange data format [1]. Based on these requirements, the development of the RailTopoModel was initiated. After several beta versions, the RailTopoModel version 1.0 was released in 2016 [2], followed by RailTopoModel 1.1 in 2017 [3]. These early versions of RailTopoModel were created under the patronage of UIC.

One of the subjects of interest of the aforementioned Feasibility Study was also the railML exchange format, developed by the independent consortium railML.org. As part of the study, railML version 2.2 was being evaluated and the need to redefine the railML format specifications was identified [1]. This triggered the development of railML 3 based on the RailTopoModel, being created, at the time. That meant a significant change in the structure of the railML format and brought a more efficient approach to modelling aspects related to railway infrastructure.

During the further development of railML 3, it became clear that not all modelling principles on which the RailTopoModel was built can be easily reflected in practice when developing the data format. Specific requirements were continuously emerging, which also required changes to the RailTopoModel specifications. This situation gradually resulted in the tendency to create a new RailTopoModel version together with each new version of railML, which concerns the RailTopoModel versions 1.2 from 2019 [4], 1.4 from 2022 [5] and 1.5 from 2024 [6] and their corresponding railML versions 3.1, 3.2 and 3.3, respectively. This process is now being managed by railML.org, which, together with new versions of the railML 3 series, also releases new underlying versions of RailTopoModel.

Despite the fact that the development of RailTopoModel had begun to be subordinated to the activities related to the development of the railML 3 format, development also continued on the UIC side, which focused on expanding the scope of the data model. This also led to its renaming to RailSystemModel. Since the RailTopoModel version 1.1 from 2017 [3] can be considered an imaginary point at the beginning of the switch where the development of RailTopoModel and RailSystemModel diverged from each other, the first and so far only version of RailSystemModel, published in 2021, is referred to as RailSystemModel 1.2 [7].

As part of the investigation described in this article, the aim was to find out how different the two diverging tracks, which can currently be considered the development of RailTopoModel and RailSystemModel, are, what features they still have in common, where they lead in parallel, and whether they have any hope of ever connecting again.

Version	Release date	Supported until	Comment
0.x	2013–2016	December 2016	beta versions; internal usage only
1.0	April 18, 2016	November 6, 2017	under patronage of UIC; also known as IRS 30100
1.1	November 6, 2017	February 19, 2019	under patronage of UIC; bugfixes only
1.2	February 19, 2019	not yet announced	used in railML 3.1
1.3	version not used to avoid confusion with forks of the same name from other projects		
1.4	April 26, 2022	not yet announced	used in railML 3.2
1.5	November 5, 2024	not yet announced	used in railML 3.3

Table 1. RailTopoModel version timeline [8].

2. Development of the models

2.1. Beginnings of RailTopoModel

The RailTopoModel is a logical object model to standardise the representation of railway infrastructure-related data [9]. Originally, RailTopoModel was an initiative of the International Union of Railways (UIC). Taking into account the conclusions of the Feasibility Study of the UIC RailTopoModel and Data Exchange Format [1] and following several beta versions, the RailTopoModel version 1.0 was published on April 18, 2016 [2]. On September 1, 2016, it was issued by UIC in the form of the document referred to as IRS 30100. The acronym IRS stands for International Railway Solution (formerly also interpreted as Standard) [10].

Regarding the general principles of data model architecture, RailTopoModel respects the principle of separation of concerns. This principle is used to break down a complex system into smaller, more manageable parts [11]. The result of this approach is the existence of several thematic packages focused on a specific function, which co-create the system architecture.

As for the RailTopoModel version 1.0, the model was divided into the *Base*, *Topology*, *Positioning Systems* and *Net Entity* packages [2]. Starting with the version 1.1, released on November 6, 2017, the *Base* package was split into the *Base* and *Network* packages, and the *Net Entity* package into the *Location* and *Net Entity* packages. This decomposition persists in subsequent versions, which are already being created together with the railML format.

The structure of RailTopoModel can be expressed using a Unified Modelling Language (UML) class diagram, which includes relationships between individual classes (both terminal, which can have concrete instances, and abstract) with the corresponding attributes. Each of these classes belongs to one of the packages. Since the data model was designed as a common basis for infrastructure modelling, the UML class diagram includes only relatively general classes. For practical use, the data model needs to be extended so that it is possible to describe specific objects appearing on the railway infrastructure in the context of the needs of a given use case. This is a matter of various specialized models, which includes, for example, the *Infrastructure* schema of the already mentioned railML exchange format.

2.2. Further develepment of RailTopoModel

Since the end of UIC's project, the further development of RailTopoModel in done by railML.org with the majority of the infrastructure managers of the initial project [9]. The main activity of the railML.org consortium is the development of the railML format designed for data exchange within the railway industry. The initiative was founded in 2002 in order to connect different IT applications using railway data. The data structure of the railML interface is based on the Extensible Markup Language (XML). It is described using the widely supported and standardised XML Schema Definition Language (XSD) [12].

While the railML versions of the series 1 and 2 used only a simplified infrastructure description, mainly for the needs of exchanging timetable data, with the advent of RailTopoModel, railML 3 began to emerge, in which the infrastructure description is more sophisticated, suitable for more demanding use cases, and based on the principles of RailTopoModel. The rtm4railml3.xsd file, whose structure reflects the RailTopoModel specifications, also became part of the XSD folder with individual railML 3 schemas.

During the gradual development of railML 3, it became clear that the RailTopoModel specifications need to be adapted to meet the requirements of the data format. Initially, when RailTopoModel was being developed under the patronage of the UIC, this was difficult to implement since the RailTopoModel specifications were considered inviolable. Later, after the development of RailTopoModel was taken over by the railML.org consortium, new versions of RailTopoModel began to be created together with new versions of the series railML 3. Among other things, RailTopoModel was on this occasion enriched and is being further enriched with some aspects that are typical for railML or that are newly incorporated into it.

After the version 1.2 from August 29, 2018 [4], and the version 1.4 from April 22, 2022 [5] (while the version under the designation 1.3 was not used), the latest version of RailTopoModel is the version 1.5, which was released on November 5, 2024 [6], together with railML 3.3. An overview of the individual versions of RailTopoModel, as they were gradually created, is presented in the Table 1.

2.3. RailSystemModel

The development alongside the railML data format is not the only way in which RailTopoModel has been expanded. Within the UIC, efforts continued to bring new aspects to the world of digital modelling of railway infrastructure so that the common data model could capture more than just issues related to the topology of the railway network and things closely related to it. As it became apparent that the model structure would become more broadly focused, it led to its renaming to RailSystemModel. Since the version of RailTopoModel that existed before the separation of the above two tendencies is referred to as version 1.1, RailSystemModel followed this numbering and on January 10, 2022, the RailSystemModel version 1.2 was released, which is its only release so far [7].

With the introduction of RailSystemModel, new possibilities for modelling railway infrastructure using the basis already known from RailTopoModel have emerged. While the RailTopoModel *Net Entity* package (after the *Location* package was separated) includes only very general classes, RailSystemModel brings more specialized classes derived from the *LocatedNetEntity* class, which allow covering the *Track*, *Signalling* and *Energy* domains of infrastructure at the top level. A separate new package, also closely related to railway infrastructure however within RailSystemModel considered independent of the *Net Entity* package, is the *Geometry* package. In addition, RailSystemModel offers scope for more sophisticated support of some aspects which is the subject of the packages *Observation and Measure*, *Net Property* and *Time Axis*. The fact that the model also includes realities outside the technical world of railways can be demonstrated in the *Legal Entities* and *Weather* packages, which are part of the *Enviroment* package.

Prospectively, this allows the data model to be used autonomously for general purpose tasks, such as describing railway assets, generating schemas and maps or simulating train runs. Nevertheless, RailSystemModel does not aim to become a data model that replaces all models, but a model that provides a lean, solid fundament for other, highly specialized models, itself based on solid industry standards. RailSystemModel cooperates with expert projects like Eulynx, IFC Rail and ERA RINF, which are specialists of their respective domains. At the same time, it remains a general, functional, project-independent implementable model [13].

Among other things, RailSystemModel has found its application in some projects of the European Shift2Rail programme and its follow-up programme Europe's Rail. It has become one of the source models (as well as, for example, the already mentioned EULYNX DP and IFC Rail) of the Conceptual Data Model (CDM) representing a federation of data models focused on various areas of data modelling in the railway sector which was being developed e.g. in the LinX4Rail and LinX4Rail-2 projects [14].

3. Comparison

This section compares both data models, focusing on the latest versions of each, i.e. RailTopoModel 1.5 [6] and RailSystemModel 1.2 [7]. Due to the need to view the issue from the perspective of historical development, previous versions will also be referenced, in particular RailTopoModel 1.0 [2], which was the original predecessor, and RailTopoModel 1.1 [3], which is the latest common version on which both current versions of the data models are based.

The assessment will be carried out taking into account the division of the data models into packages as introduced in RailTopoModel 1.1 and also valid for subsequent versions up to RailTopoModel 1.5. Domains that are contained exclusively in RailSystemModel 1.2 and do not appear in RailTopoModel at all will not be taken into account here, except for a few minor notes, in some cases. These are the packages *Geometry*, *Observation and Measure*, *Net Property*, *Time Axis*, *Legal Entities* and *Weather*. Nevertheless, we could find alternatives to some of them in the railML 3 format, which builds on RailTopoModel.

Let us be content with stating that here we could also look for differences, for example in the fact that the geometric entities defined in the railML 3 *Infrastructure* schema are considered specializations of the *RTM_LocatedNetEntity* class, while the corresponding alignment segments from RailSystemModel 1.2 are of the classes derived directly from the *BaseObject* class. Another difference is that RailSystemModel has defined classes for each specific type of geometric curve, while railML 3 only distinguishes more generally between horizontal and gradient curves [6, 7, 15].

3.1. Base

In RailTopoModel 1.0, the *Base* package included both the *BaseObject* class with *id*, *name*, *validFrom* and *validTo* attributes and classes describing the network. The classes describing the network were subsequently moved to the separate *Network* package, and the contents of the *Base* package were divided into several other classes. Nevertheless, its main subjects remain the expression of the identity of a object, its naming, and the expression of validity. The latter extends beyond the *Base* package, specifically into the *Positioning* package, but for practical reasons will be described here, uniformly.

The first aspect is the method of identifying objects. The basic principle of internal object identification, which was applied in early versions of RailTopoModel and is still maintained in both RailTopoModel 1.5 and RailSystemModel 1.2, is the use of the *id* attribute of the *BaseObject* class. Within RailTopoModel 1.0, it was originally assumed that the value of this attribute would always be generated as a Universally Unique Identifier (UUID) [16]. This may still be considered

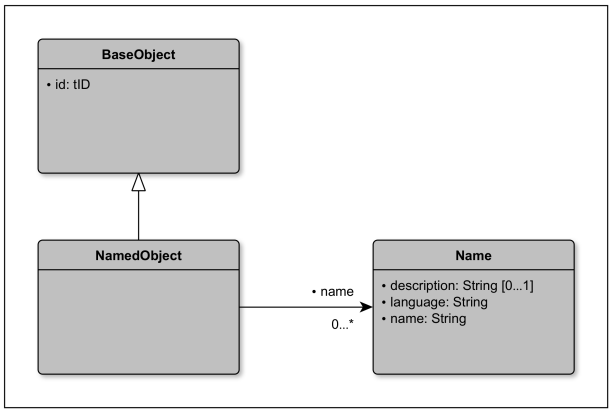


Figure 1. Identification and naming of an object in RailTopoModel 1.5 [6].

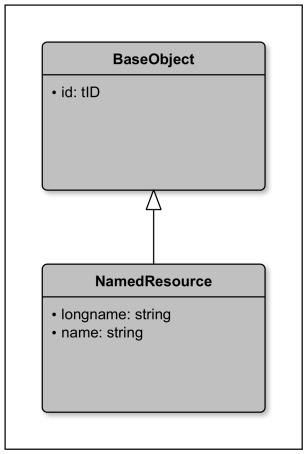


Figure 2. Identification and naming of an object in RailSystemModel 1.2 [7].

the preferred solution, however, due to the need to allow models to be used for existing systems, the legacy id have also been allowed as an alternative, starting already with the RailTopoModel version 1.1.

Regarding naming objects with a user-readable name, which may not be unique in all circumstances (and certainly not universally), the approach of the current version of RailTopoModel is more different from the RailSystemModel approach. In RailTopoModel 1.0, only the *name* attribute of the *BaseObject* class was used for these purposes. Starting with RailTopoModel 1.1, the *NamedResource* class was introduced, which is a specialization of the *BaseObject* class carrying the additional *name* and *longname* attributes, which allows instances of the classes that are specializations of the *NamedResource* class to use two user-defined names. While RailSystemModel 1.2 still reflects this approach, newer versions of RailTopoModel are already influenced by the approach that comes from the railML data format. For example, in the RailTopoModel 1.5 version, user-named objects are instances of the classes derived from the *NamedObject* class of the *Base* package to which any number of instances of the *Name* class with the attributes *name*, *language* and optionally *description* can be associated. This allows a user to name individual objects in different languages.

The Figures 1 and 2 demonstrate the difference between the expression of object naming in RailTopoModel 1.5 and RailSystemModel 1.2. In addition, railML 3 also allows us to use a designator for some objects, which is, nevertheless, not part of RailTopoModel [15].

Another aspect of the systemic character that can be captured using both data models, but in each case in a different way, is the temporal validity of objects. While in RailTopoModel 1.0, validity was expressed using the *validFrom* and *validTo* attributes of the *BaseObject* class, in the version 1.1, it moved to more specific classes (specifically *NetworkResource* of the *Network* package and *PositioningSystem* and *AssociatedPositioningSystem* of the *PositioningSystem* package). Nevertheless, it still only allowed the expression of one continuous period of time. In current versions of the data models, this issue is addressed in a more sophisticated way. In RailTopoModel 1.5, each instance of the *NetworkResource* class, which is not a specialization of the *NamedObject* class in this case, can be associated with any number of instances of the *Validity* class with *from* and *to* attributes indicating the bounding time positions.

In the case of the *PositioningSystem* and *AssociatedPositioningSystem* classes, which are specializations of the *NamedObject* and *BaseObject* classes, respectively, the possibility of expression remains only using one continuous time interval bounded by the values of the *validFrom* and *validTo* attributes. RailSystemModel 1.2 takes advantage of introducing quantities directly into the model classes. It uses instances of the *TemporalPosition* class, which is a specialization of the general *Quantity* class, to define time intervals. Instances of the *SimpleTimeInterval* class, which belongs to temporal entities, can have any number of beginnings and ends expressed by instances of the *TemporalPosition* class. The validity itself is then expressed using the *ValidityPeriod* class, which is a specialization of the *SimpleTimeInterval* class. An instance of the *ValidityPeriod* class can be part of any instance of the *NetworkResource* class or can be associated with instances of the *PositioningSystem* and *AssociatedPositioning* classes. In RailSystemModel 1.2, the *PositioningSystem* class is a specialization of the *NamedResource* class, and *AssociatedPositioning* is not a specialization of any more general class.

The differences in aspects related to validity issues in RailTopoModel 1.5 and RailSystemModel 1.2 are demonstrated in the Figures 3 and 4.

3.2. Network

As can be seen in the Figures 5 and 6, the *Network* package consists of three basic building blocks for defining a railway infrastructure network. This is

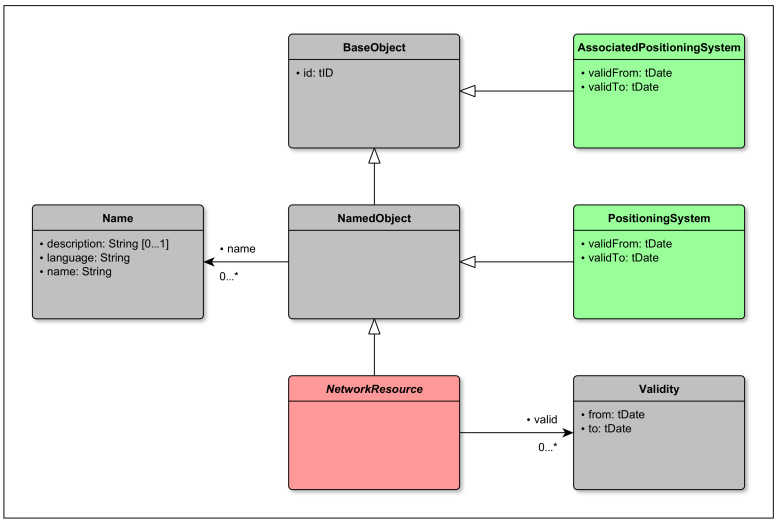


FIGURE 3. Validity in RailTopoModel 1.5 [6].

the *Network* class, which, in the case of the current versions of both compared data models, is a specialization of the *NamedResource* class. Its instances represent individual railway infrastructure networks. Each network can be expressed at one or more levels or detail which are represented by instances of the *LevelNetwork* class with the *descriptionLevel* attribute in the case of RailTopoModel 1.5 or by instances of the *NetworkLevel* class with the *hasDescriptionLevel* attribute in the case of RailSystemModel 1.2. The attribute expressing the level of detail can take on the values *Macro*, *Meso* and *Micro* and in the case of RailSystemModel 1.2 also the value *Nano*.

Based on the level of detail, it is possible to infer what types of objects the network consists of. Within the *Network* package, it is not yet possible to define specific network objects, however the abstract class *NetworkResource* is introduced, whose specializations are classes of topological elements and network entities that can be part of a network. Any number of network resources can be associated with each network level. In RailSystemModel 1.2, the possibility of aggregating individual network resources directly into the network still remains, while in RailTopoModel 1.5, apparently with regard to the structure of the railML data format, this aggregation is now handled only via the *LevelNetwork* class. In any case, this is a weaker form of ownership, where each network resource can belong to multiple networks, which is expressed by aggregation (not composition).

3.3. TOPOLOGY

At a general level, it can be stated that the *Topology* package of the considered data models works with net elements and relations, which are used to build the skeleton of a network. These can be represented by classes with different names at different levels of abstraction and specialization, and also with respect to the respective model. Nevertheless, these are always classes that are somehow based on the *NetworkResource* class. Within RailTopoModel 1.5, the most specialized (terminal) classes are the *PositioningNetElement* and *NetRelation* classes. Each instance of the *NetRelation* class is described by the *navigability*, *positionOnA*, and *positionOnB* attributes, which define the method of connecting the net relation to the respective two net elements and the possibility of traversing this relation. At the same time, it remains required to associate net relations from net elements.

There is also a possibility to express the fact that a net element is a collection of several element parts. It can be an ordered or unordered collection. While this feature reflects the original RailTopoModel approach, RailTopoModel 1.2 also includes a connector mechanism in the *Topology* package. This is a concept based on research by the author of this article during his internship at railML.org. Its purpose is to allow the expression of connecting and splitting of network descriptions within the railML data format [17]. Therefore, its equivalent cannot be found in RailSystemModel 1.2.

Unlike RailTopoModel 1.5, RailSystemModel 1.2 uses specializations of the *NonLinearElement* and *LinearElement* classes, which are the specializations of the *PositioningNetElement* class and allow for distinguishing between non-linear and linear elements of the topology description. Originally, RailTopoModel 1.0 and RaiTopoModel 1.1 also included them, nevertheless with the advent of RailTopoModel 1.5, they disappeared as there was no use for them in railML. RailSystemModel 1.2, on the other hand, introduced a specialization of the *LinearElement* class called *LinearElementWithLength*, which addresses the difficulties associated with assigning lengths to linear network elements. It is often required for practical reasons but it also contradicts the concept of pure dimensionless topology, as was already addressed at the time when railML 3 began to be created.

In RailSystemModel 1.2, as well as in the RailTopo-

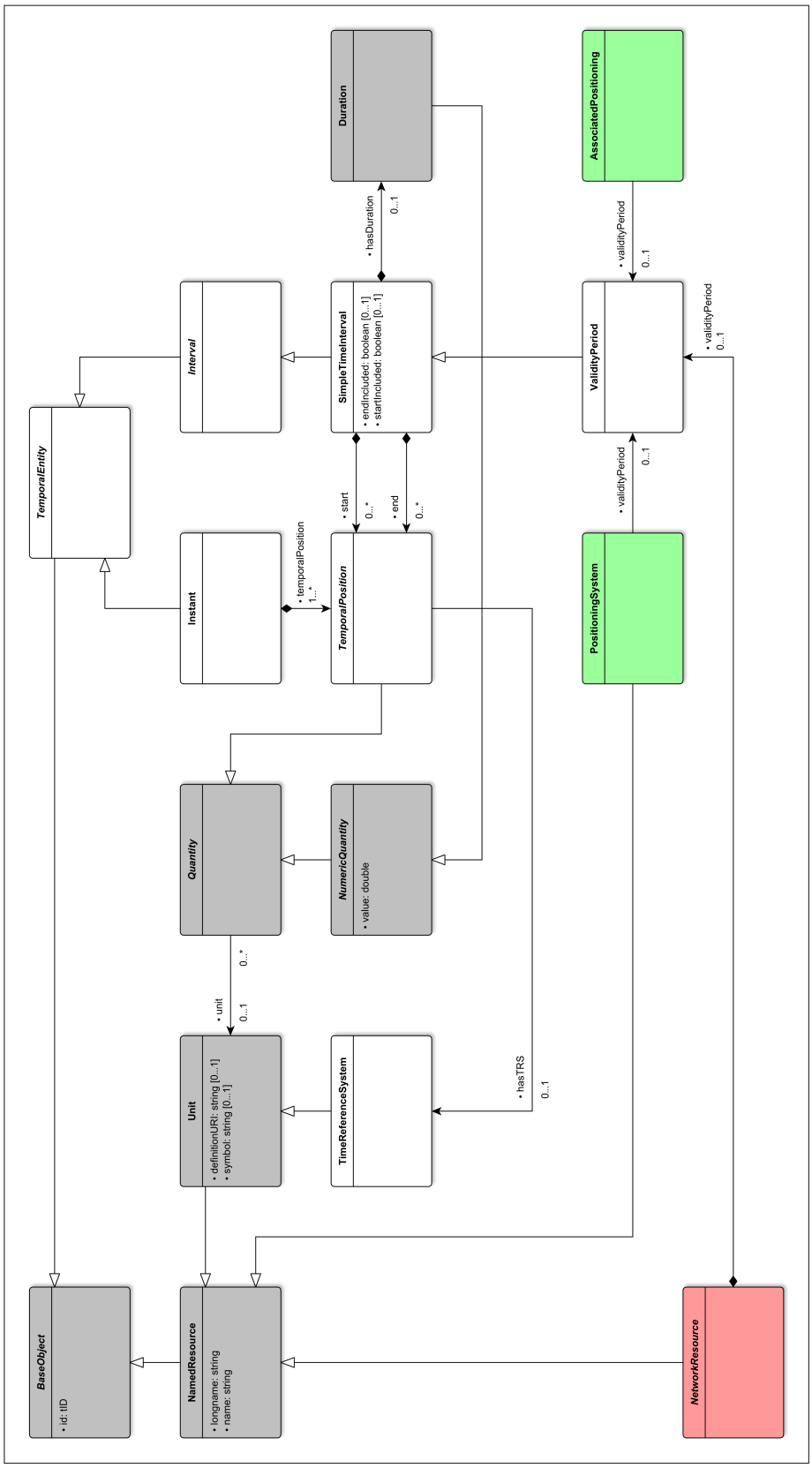


Figure 4. Validity in RailSystemModel 1.2 [7].

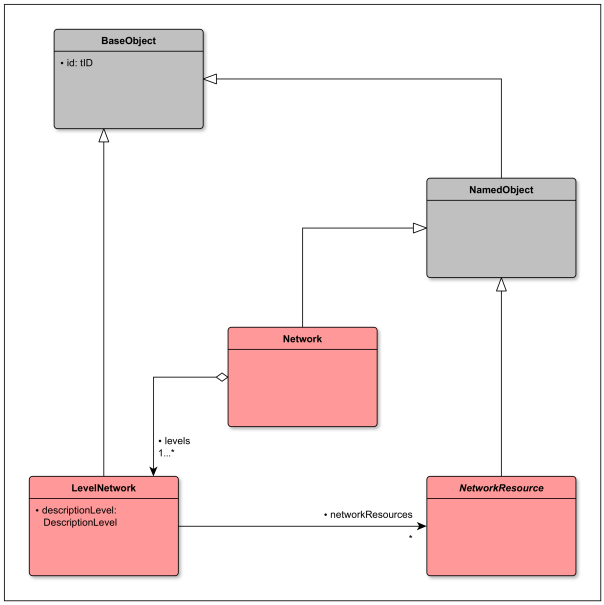


FIGURE 5. Network in RailTopoModel 1.5 [6].

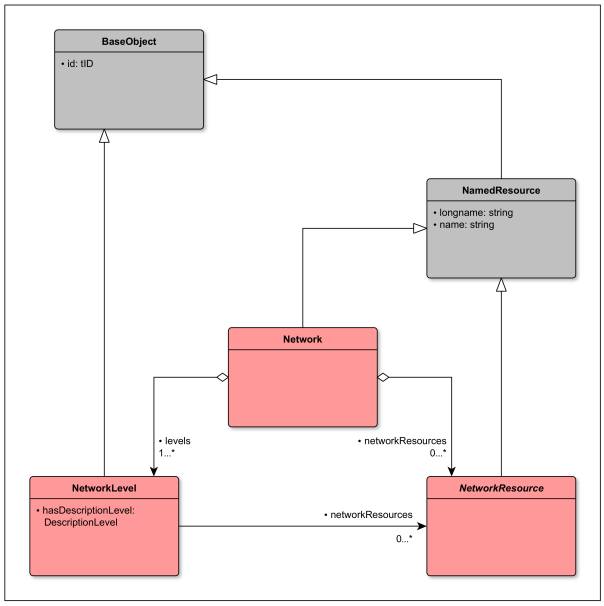


FIGURE 6. Network in RailSystemModel 1.2 [7].

Model versions 1.0 to 1.2, the class whose instances connect the instances of the *PositioningNetElement* class (or its specializations) is called *PositionedRelation*. The names and functions of the related attributes remain the same across different versions of the data models, i.e. as listed for the *NetRelation* class of RailTopoModel 1.5. The only difference is that in RailSystemModel 1.2, to an instance of the *PositionedRelation* class, the attribute *leadTowards*, which allows the direction of the relations underpinning turnouts to be expressed, can be assigned and that there is no (apparently redundant) association from net elements to relations. Indeed, there is still a mechanism for creating ordered and unordered element part collections.

The differences between the data models within the *Topology* package can be seen in the Figures 7 and 8.

3.4. POSITIONING

Positioning within RailTopoModel and RailSystemModel can be viewed from different perspectives. First of all, it may be about expressing a position within a net element. The original UIC versions of RailTopoModel already considered that an instance of the *PositioningNetElement* class is composed of one or more instances of the *AssociatedPositioningSystem* class. In this case, it is a strong ownership. No associated coordinate system can exist independently and be shared by multiple positioning net elements. Similarly, each associated positioning system is composed of several instances of the *IntrisicCoordinate* class. The intrinsic coordinate value within a positioning net element is specified by the *intrinsicCoord* attribute.

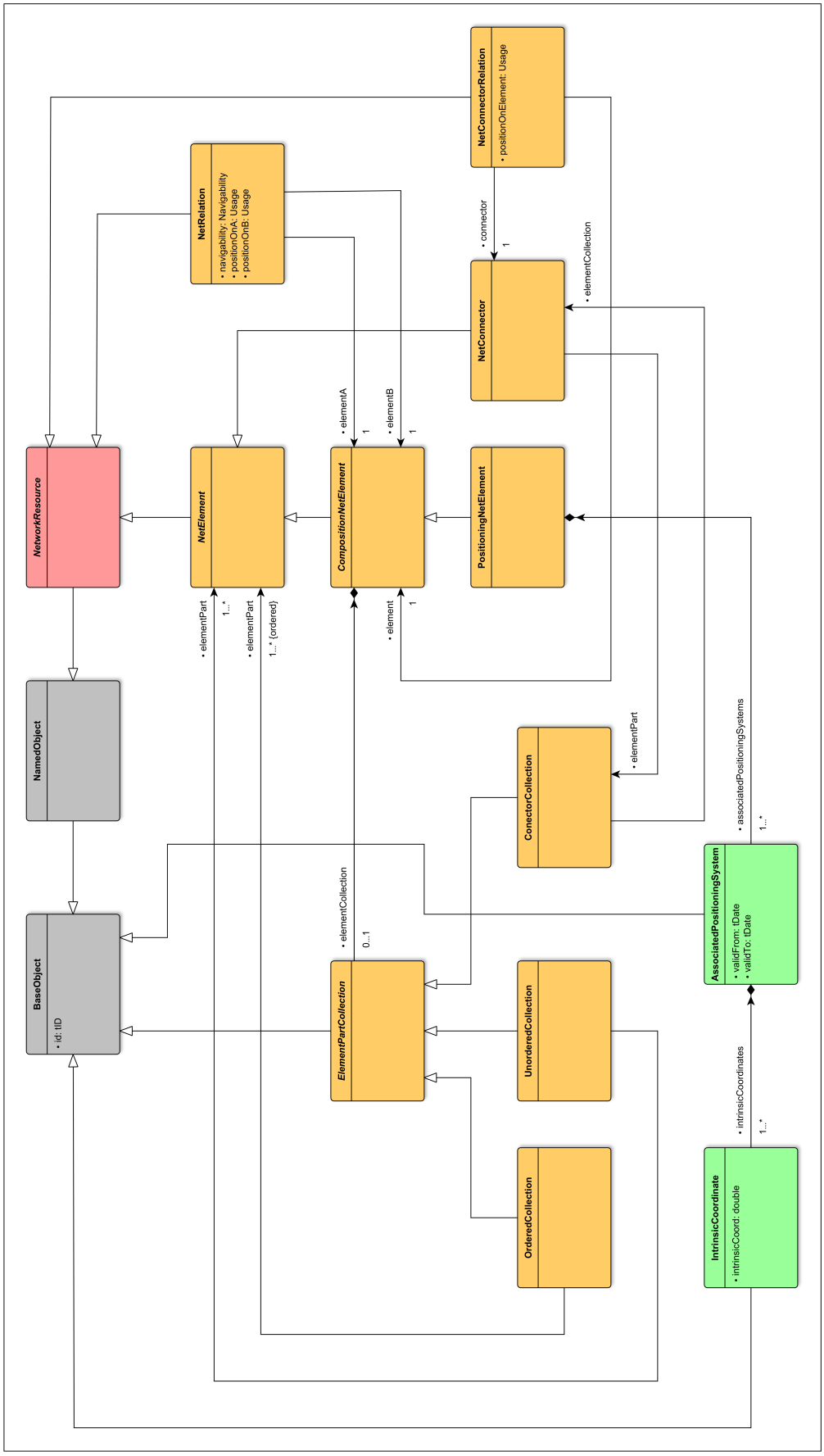


Figure 7. Topology in RailTopoModel 1.5 [6].

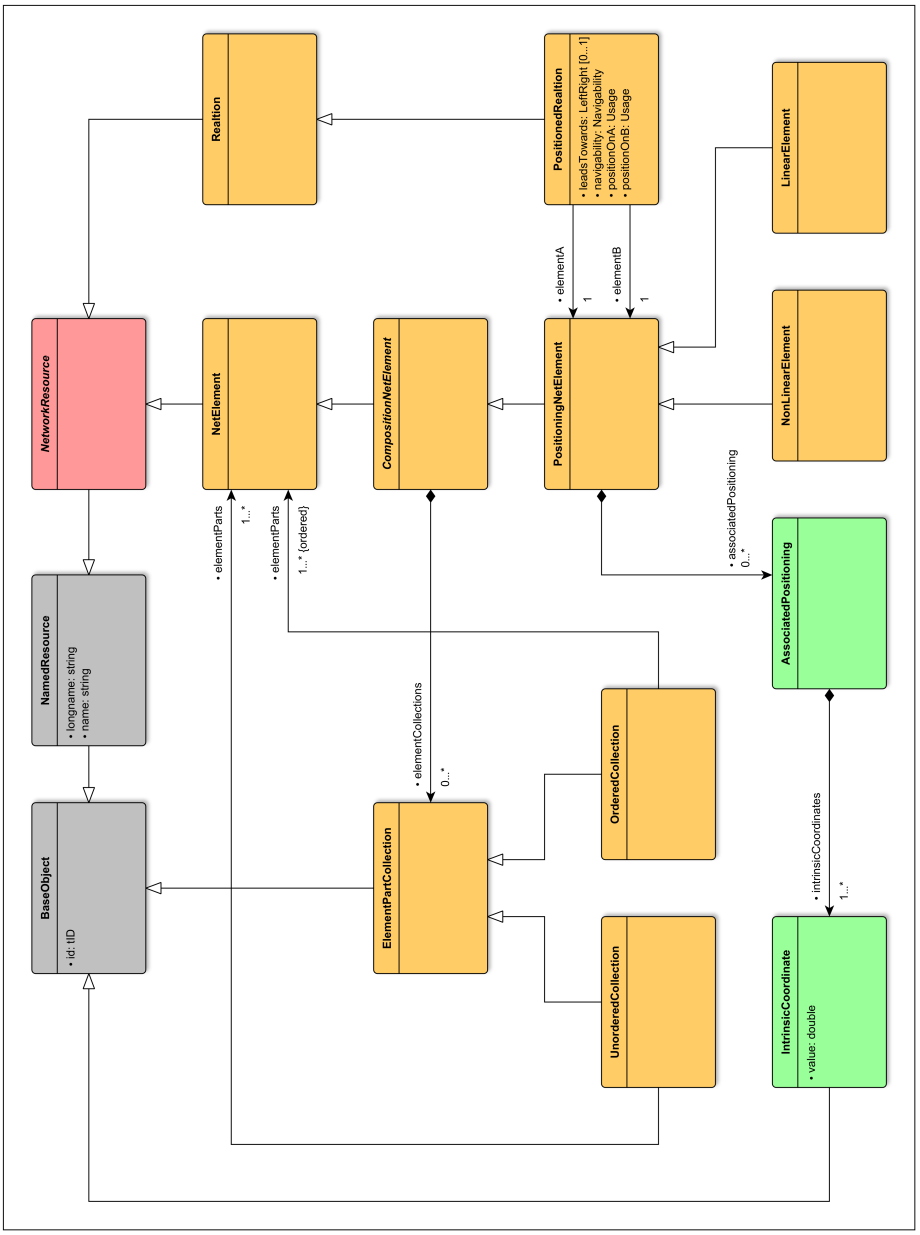


FIGURE 8. Topology in RailSystemModel 1.2 [7].

The identification of intrinsic coordinates with coordinates of other coordinate systems can be implemented based on the association from the *IntrinsicCoordinate* class to the *PositioningSystemCoordinate* class. This principle remained the same up to and including the RailTopoModel version 1.5, with the only change compared to RailTopoModel 1.0 regarding the classes from which the *AssociatedPositioningSystem* and *IntrinsicCoordinate* classes inherit. Currently, both of these classes are specializations of the *BaseObject* class.

In RailSystemModel 1.2, this is addressed similarly, with the difference that the instances of the *PositioningNetElement* class are composed of any number of instances of the *AssociatedPositioning* class (it is also permissible that a positioning net element has no associated positioning). Each instance of the *AssociaedPositioning* class is then a composition of at least one instance of the *IntrinsicCoordinate* class. The intrinsic coordinate is expressed by an attribute of this class called *value*, although the meaning is the same as in RailTopoModel 1.5 in the case of the *intrinsicCoord* attribute. The association to the *PositioningSystemCoordinate* class is implemented in the same way as in RailTopoModel 1.5. The *AssociatedPositioning* class is not a specialization of any more general class, while the *IntrinsicCoordinate* class is again a specialization of the *BaseObject* class.

The second area covered within the *Positioning* package refers to external coordinate systems. This concept is expressed by the general *PositioningSystem* class, whose specializations in the case of RailTopoModel 1.5 are the *LinearPositioningSystem* and *GeometricPositioningSystem* classes. This is a categorization that has survived from earlier versions of Rail-

TopoModel, as well as the corresponding attributes of the classes. RailSystemModel 1.2 started using a slightly different structure. While the *LinearPositioningSystem* was retained, the *GeometricPositioningSystem* class was replaced by two specializations. These are the classes *Canvas* (a 2D or 3D physical or virtual bounded space, such a screen or a sheet of paper on which the positioning takes place) and *GeographicCoordinateSystem*. Furthermore, the *GeographicCoordinateSystem* class has the *GridReferenceSystem* and *GeodeticReferenceSystem* specializations defined. Some changes have also affected the class representing the anchor points of the linear coordinate system. While in RailTopoModel it is the *LinearAchorPoint* class whose instances are components of a linear coordinate system, in RailSystemModel it is called *AnchorPoint*, the dependency mentioned is a general association and some other relationships have been added.

In addition to defining positioning systems, it is also necessary to express coordinate values. The general concept of coordinates is the *PositioningSystemCoordinate* class. Its instances, or rather instances of its specializations, express individual coordinates, or rather their tuples. The assignment of a tuple of coordinates to a positioning system is handled at a general level using the association between the *PositioningSystemCoordinate* class and the *PositioningSystem* class. Nevertheless, it is obvious that the type of coordinates matters. RailTopoModel 1.5 still uses the previously introduced specializations of the *PositioningSystemCoordinate* class, which are the *LinearCoordinate* and *GeometricCoordinate* classes, which accurately reflect the names of the respective positioning systems.

What is new is that the components of instances of these classes are instances of the *AssociatedLinearCoordinate* or *AssociatedGeometricCoordinate* classes, respectively, which are the specializations of the *AssociatedPositioningSystemCoordinate* class, also associated with one of the coordinate systems. In this version of RailTopoModel, they form the interface between the *Positioning* and *Location* packages. In RailSystemModel 1.2, the *LinearCoordinate* and *GeometricCoordinate* classes still exist, but the *GeometricCoordinate* class is now an abstract class. Instances can only be created based on their specializations, which are the classes *CartesianCoordinate* with coordinates expressible by the attributes *x*, *y* and optionally *z*, *GeographicCoordinate* with coordinates expressible by the attributes *longitude*, *latitude* and optionally *elevation*, or the class *Elevation* with the only attribute *value*. For these positioning system coordinate types, the affiliation to the type of positioning system is no longer so clear. In RailSystemModel 1.2, positioning system coordinates are now considered observable properties, generally defined in the *Observation and Measure* package.

Other newly introduced classes of the positioning package in RailSystemModel 1.2 are the *PolyLine* class and its specialization *Polygon*, whose instances are shapes defined by at least two coordinates always of the same positioning system.

An overall comparison of the Positioning package in RailTopoModel 1.5 and RailSystemModel 1.2 can be made based on the Figures 9 and 10.

3.5. Location

The *Location* package is used to express how net entities are localized to the topology. It includes classes that represent individual types of locations. Originally, in the version 1.0, RailTopoModel also included the *NetEntity* class, which is a specialization of the *NetworkResource* class expressing the general concept of a functional object (in some cases also a property) of railway infrastructure, and its specialization *LocatedNetEntity*, which extends the concept to include one or more locations. It was then solved as an aggregation. With the advent of RailTopoModel 1.1, these two net entity classes were separated into a standalone package and the respective association of the *LocatedNetEntity* class to the *EntityLocation* class was changed to a composition. The *EntityLocation* class is then the general concept of location. Originally, the class was considered a specialization of the *NetworkResource* class. Starting with RailTopoModel 1.1, it is now a specialization of the *BaseObject* class. Its specializations, since the beginning of RailTopoModel, have been the *SpotLocation*, *LinearLocation* and *AreaLocation* classes, which express localization either at a specific point on a net element, a section (even across multiple consecutive net elements) or an area (which typically includes multiple net elements or sections on them), respectively.

The way these classes are linked to the topology and positioning packages has changed during the development of RailTopoModel. While the *SpotLocation* class can be associated directly with the *PositioningNetElement* class or with the more general *NetElement* class (as is the case in RailTopoModel 1.5), in the case of linear and area locations, this is mediated by the *AssociatedNetElement* class (or for linear locations previously typically by its specialization *OrderedAssociatedNetElement*), through which a section is essentially defined on a specific net element, which sections are then composed in the relevant location.

Since the way the position is expressed on a net element matters, specializations of the *SpotLocation* class were defined, specifically the *SpotLocationIntrinsic* class, where the position is defined using the *intrinsicCoord* attribute, and the *SpotLocationCoordinate* class, ensuring association to the *PositioningSystemCoordinate* class, or, in the RailTopoModel 1.5 version, *AssociatedPositioningSystemCoordinate*. In RailTopoModel 1.5, even the *SpotLocation* class is considered abstract and instantiation is the subject of its listed specializations. Starting with the RailTopoModel version 1.1, specializations of the *AssociatedNetElement* class, namely *AssociatedNetElementIntrinsic* and *As-*

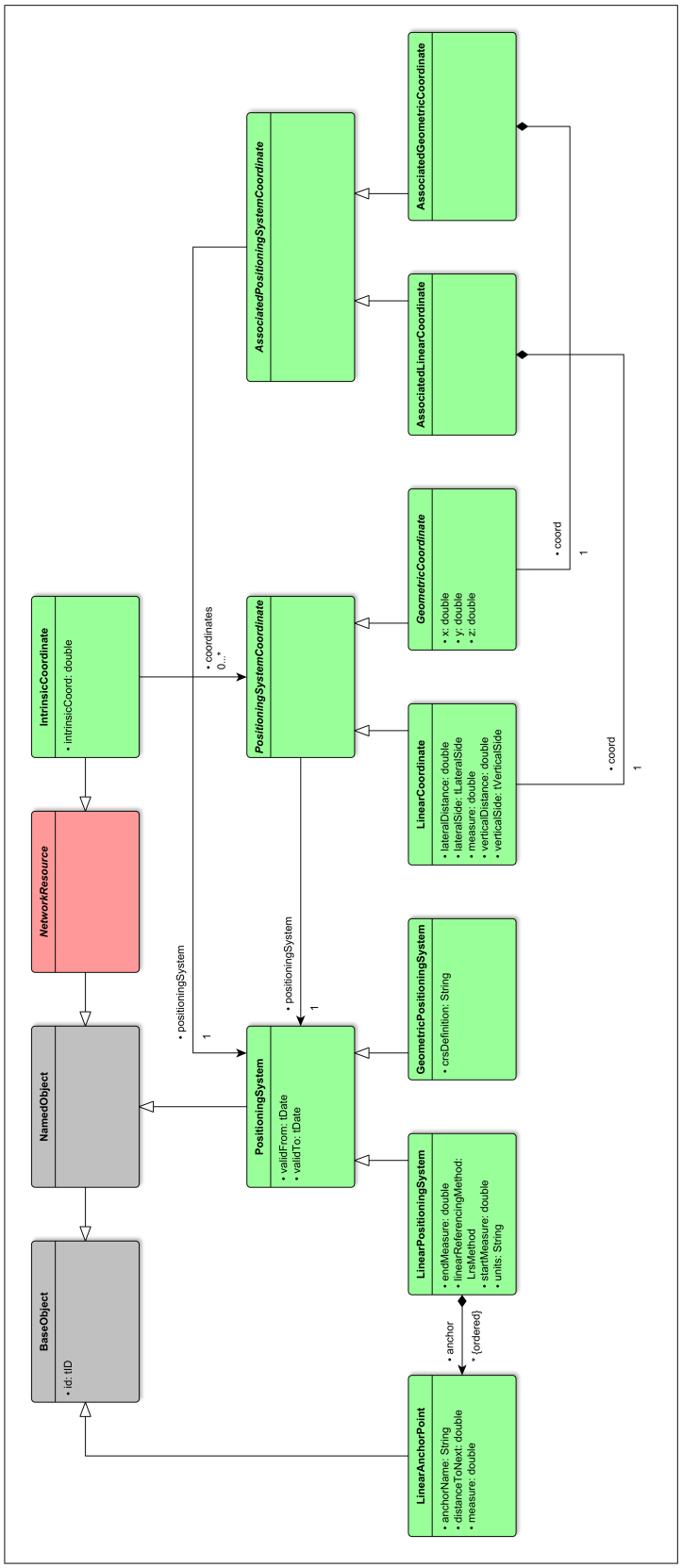


FIGURE 9. Positioning in RailTopoModel 1.5 [6].

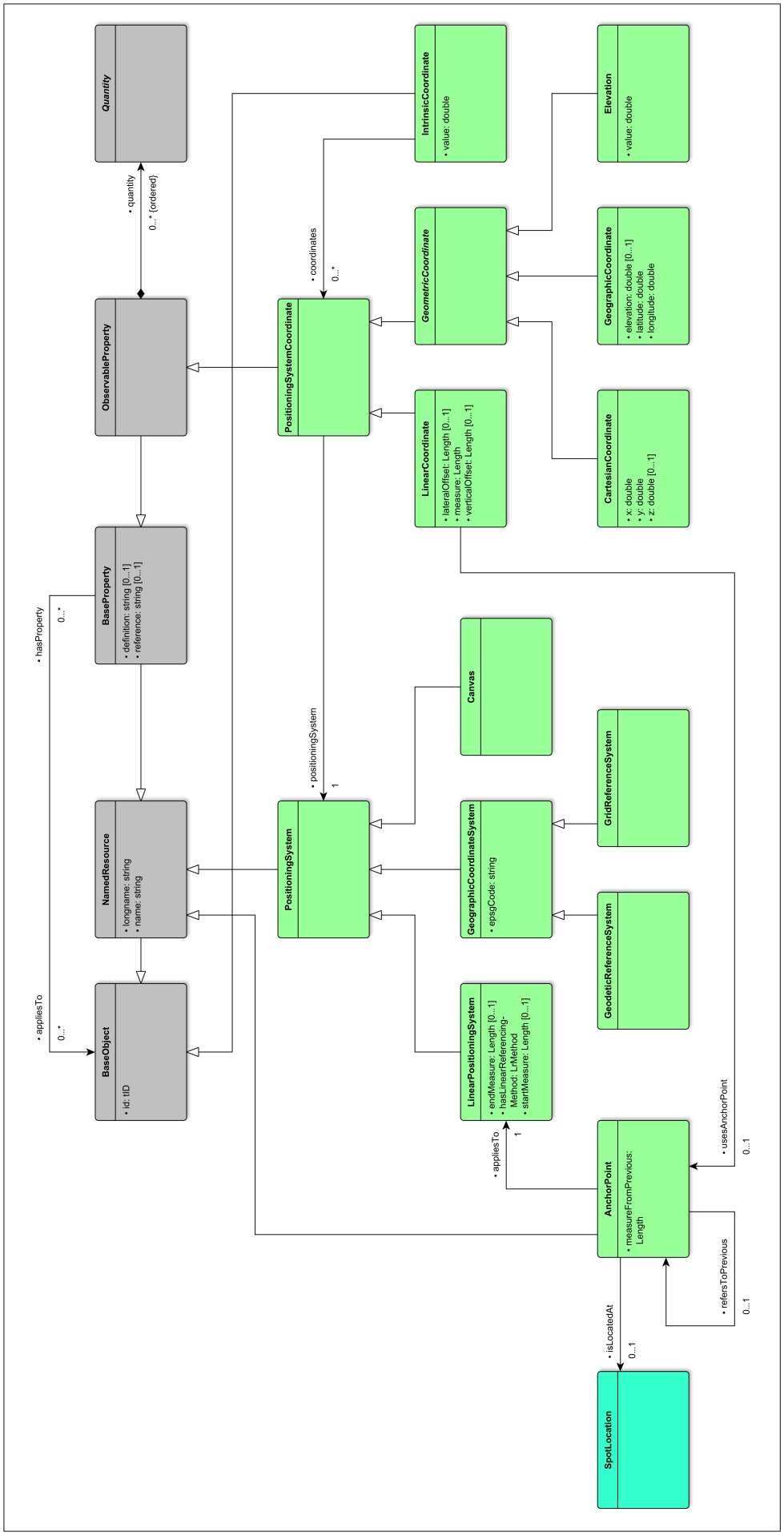


Figure 10. Positioning in RailSystemModel 1.2 [7].

sociatedElementCoordinate, are defined to perform a similar function.

Since in some cases the function of an object connected to the topological layer can only be applied in a certain direction, the *SpotLocation* and *LinearLocation* classes have the *applicationDirection* attribute, by which this can be expressed. In older versions of RailTopoModel, the *AssociatedNetElement* class also included the *keepsOrientation* attribute, which took into account the direction of the relevant net element participating in defining the location, however starting with RailTopoModel 1.4, this attribute disappeared.

As for RailTopoModel 1.2, the name of the general location class has been changed to *BaseLocation*. This class has reverted to a weaker relationship to located net entities, which is again an aggregation, with each located net entity being able to have any number of locations (newly including none). Although RailSystemModel 1.2 still uses the *SpotLocation*, *LinearLocation*, and *AreaLocation* classes (which are here specializations of the *BaseLocation* class), it has eliminated their specializations, as well as the specialization of the *AssociatedNetElemnet* class. It introduced the *TimedLocation* class as a specialization of the *SpotLocation* class, that is true, nevertheless, it serves a completely different purpose. It standardized the function of the *AssociatedPositioningElement* class to provide an association to the *PositioningNetElement* class for all types of locations. That means that this is newly used also by spot location. Each individual location is a composition of any number of instances of the *AssociatedNetElement* class. Each instance of the *AssociatedNetElement* class is further associated with two bounding instances of the *IntrinsicCoordinate* class, already known from the *Positioning* package.

The class *AssociatedNetElement* also carries an attribute expressing the application direction, this time named *appliesInDirection*, and also the semantically new attribute *isLocatedToSide* (nevertheless, we can also find an equivalent of it in railML 3 at the level of some specific net entity classes).

The linking to external coordinate systems then takes place at the level of individual location types. Specifically, an instance of the *SpotLocation* class can have any number of positioning system coordinates associated with it, linear location refers to any number of polylines, and area location refers to any number of polygons.

The differences in the concept of the *Location* package regarding its expression in RailTopoModel 1.5 and RailSystemModel 1.2 can be seen again in Figures, this time in the Figures 11 and 12.

3.6. NetEntity

As already stated, net entities represent objects and properties of railway infrastructure. After this package was separated from the *Location* package in RailTopoModel 1.1, it remains unchanged in other versions of RailTopoModel, including the current version 1.5, containing only the generic classes *NetEntity* and *LocatedNetEntity*. Their specializations, which are mentioned in some schemes, are given for illustrative purposes only. It is up to the specific domain model to implement specific types of net entities on its own. This is also the case with the railML 3 exchange format, which derives its infrastructure object classes, mainly from the *Infrastructure* schema, from the *RTM_LocatedNetEntity* class. The *NetEntity* domain itself remains narrow like this even within RailSystemModel 1.2. Specializations of the *LocatedNetEntity* class are included in the *Track*, *Signalling* and *Energy* domains.

4. Way to reconnection

As could be seen in the previous analysis, both data models within the assessed most recent versions, i.e. RailTopoModel 1.5 and RailSystemModel 1.2, compared to their common predecessor, RailTopoModel 1.1, which follows on from RailTopoModel 1.0, have moved a significant step forward and offer useful concepts for data modelling of railway infrastructure. Although from the perspective of each of these individual models, this progress can be perceived as very positive and beneficial to the railway sector, where data communication plays an increasingly strong role, when looking at this situation comprehensively, we can also notice its downsides. The original idea of creating one general data model that would become a common solid basis for individual specialized data models, with which a data format ensuring data exchange would be compatible, was considerably disrupted by the described development.

The current situation is that there are now two of these data models without any mutual coordination of further development, with one of them being designed practically exclusively for forming the basis for an exchangeable data format that is not compatible with the other. There are cases where the same reality is modelled differently in each of the data models. There are some classes and attributes with the same or similar names in both data models, but they may have slightly different meanings in the context of each data model, which can be particularly confusing.

Nevertheless, it is also possible to find cases where the reality described can be modelled only using one of the data models. It can be expected that the user may be interested in using concepts from both models simultaneously. An example can be the use of advanced object naming occurring in RailTopoModel 1.5 also for objects that are instances of specializations of the *LocatedNetEntity* class defined within RailSystemModel 1.2.

Possible solution for mapping data structured according to different data models is the use of ontologies, which emphasize the semantics of data, regardless of their representation, and have recently gained importance in the area of semantic linking

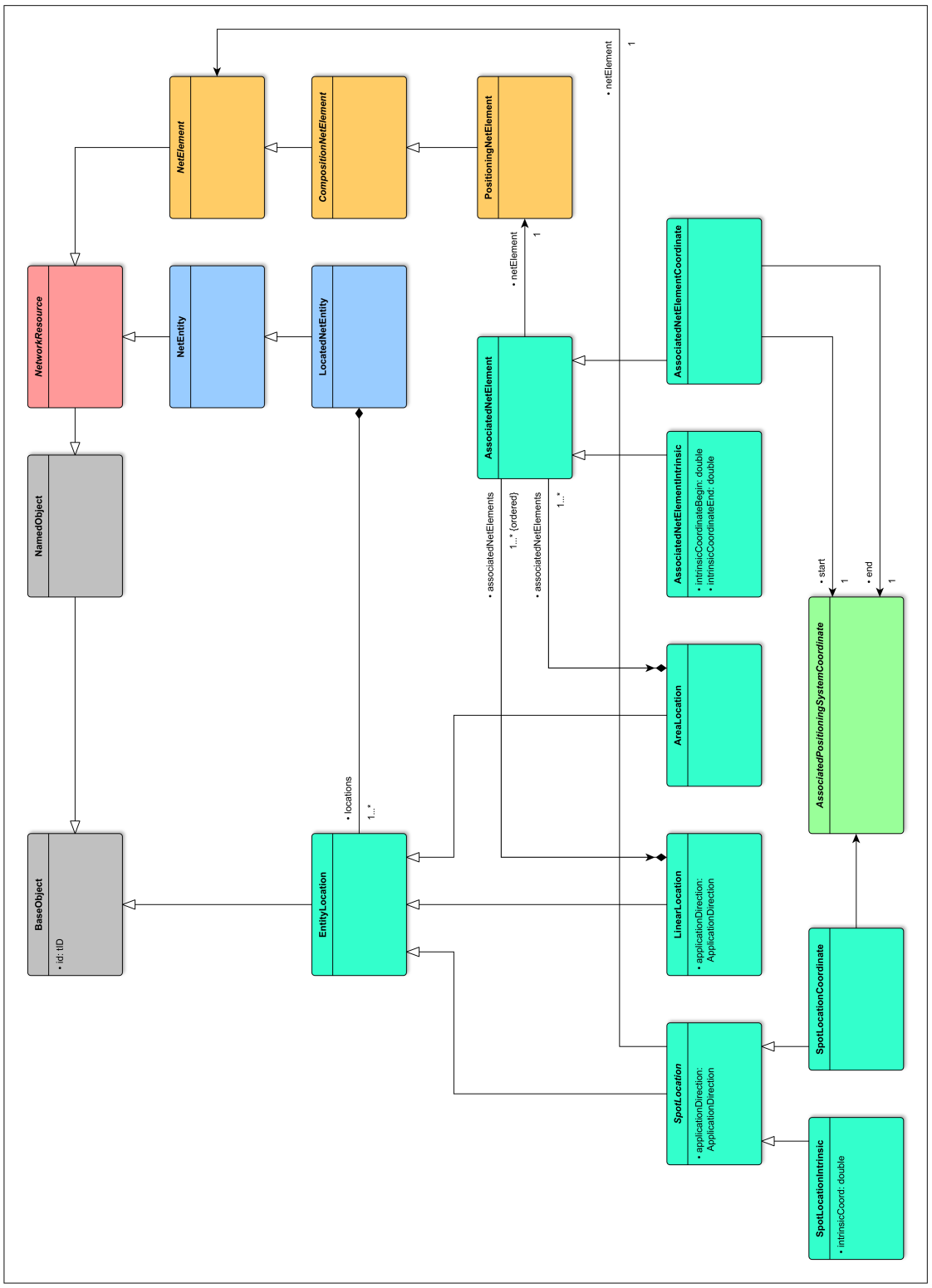


Figure 11. Location in RailTopoModel 1.5 [6].

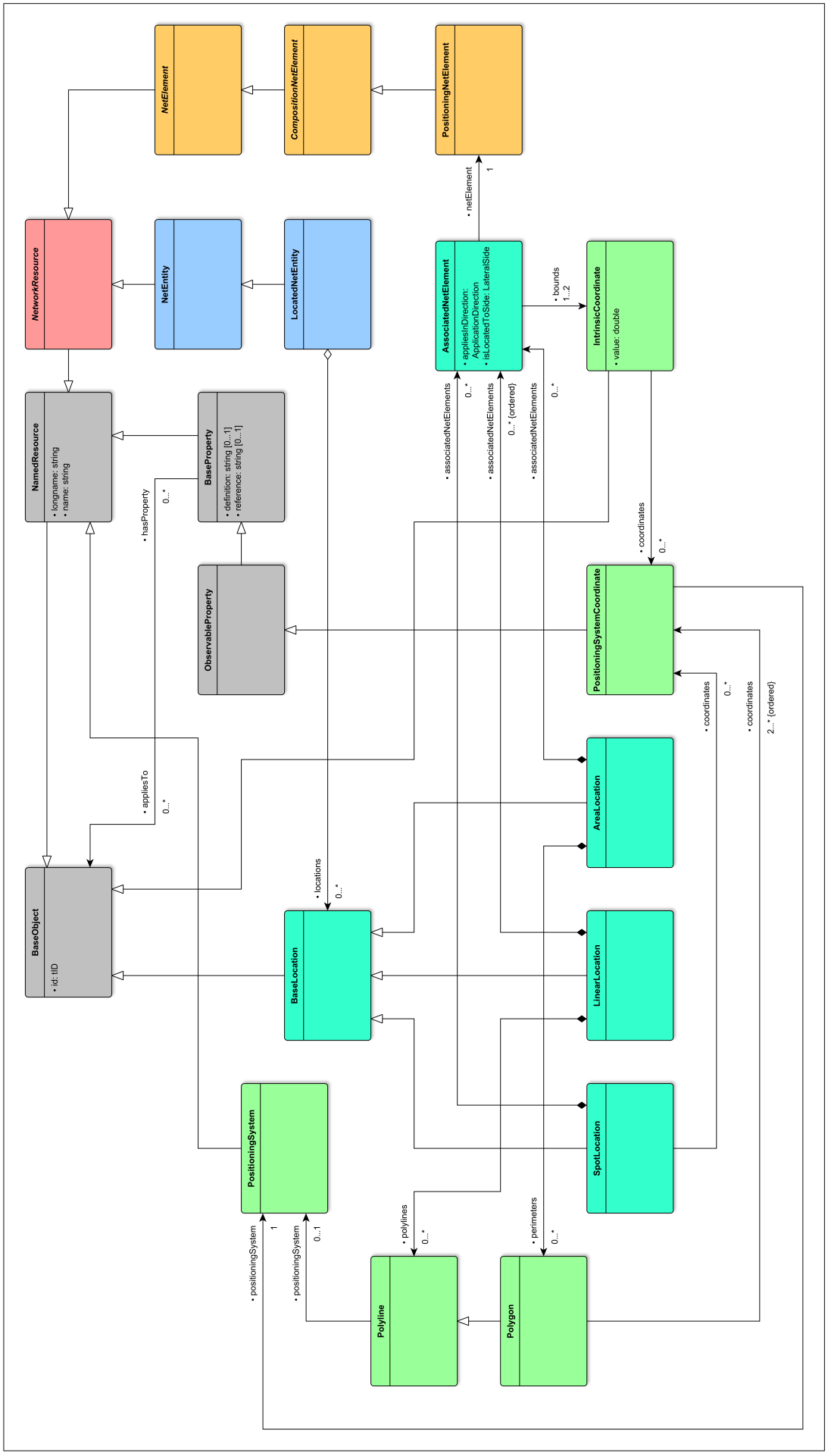


FIGURE 12. Location in RailSystemModel 1.2 [7].

of data models. In software engineering, ontology is understood as a representation of domain knowledge in the form of a set of concepts within the domain, the properties of the concepts and interrelations of those concepts [18].

In order to express knowledge in a machine-readable form, Web Ontology Language (OWL) is often used. It is based on the Resource Description Framework (RDF), which is a general-purpose language for representing information in the Web. Its data can be expressed using a graph with directed edges, which can be represented as a set of triples. It can also be completely written in a compact and natural text form using textual syntax called Terse RDF Triple Language (Turtle). This is data stored in files with the ttl extension [19–21].

While UML class diagrams, which are used to express the structure of RailTopoModel and RailSystemModel, are essentially descriptive, they can also be transformed into other languages that are more adapted to specific tasks, which in the case of semantics could be RDF/OWL [22]. What's more, in addition to the xmi format, RailSystemModel 1.2 description is also available directly in the format ttl. A transformation process is also defined to enable RDF serialization [23].

Activities related to the development of ontologies are taking place both within railML.org, where the Ontology working group was also established [24], and within the UIC Digital Modelling Group, which ensures the governance of both RSM and the OntoRail project integrating multiple source ontologies [25]. Efforts are underway to harmonize individual railway data models, including newly emerging ones, e.g. within the framework of the Europe's Rail programme projects, with the ERA Ontology. This is the human and machine readable vocabulary governed by the European Union Agency for Railways (ERA). It represents the concepts and relationships linked to the sectorial legal framework and related use cases and is still being expanded [26]. The ERA Ontology therefore appears to be a suitable candidate also in the case of efforts to harmonize concepts between the existing versions of RailTopoModel and RailSystemModel. Nevertheless, this is essentially another data model, which again may not be certain to maintain its position as a general reference model in the long term.

Even though ontologies offer the possibility of interconnecting concepts (representing, for example, the names of object classes in different data models), they cannot help if there is no possibility to express a certain fact in any of the source models. Possible complications can also arise if instances of the concepts appear in different data models whose semantics are very similar, but still slightly different. This may be the case with RailTopoModel and RailSystemModel, where the semantics may not be completely obvious in some cases and may evolve over time, or across individual versions.

An example is the *LocatedNetEntity* class, which appears in both data models, but judging by its specializations, it is somewhat more broadly understood in the context of RailTopoModel 1.5 or railML 3.3 than in the context of RailSystemModel 1.2 (where, for example, it does not include alignment curves and net properties), which may not be obvious at first glance. At the same time, the generic *EntityLocation* class of RailTopoModel 1.5 seems to be semantically identical to the *BaseLocation* class of RailSystemModel 1.2. Is this really the case or is there some minor difference that makes it not a pure 1:1 relationship? As can be seen from these examples, the partially separated worlds of RailTopoModel and RailSystemModel may bring inconveniences that may not be appropriate and effective to try to deal with using the tools offered by ontologies. At the very least, it would be a challenging task.

From a technical point of view, it would therefore be advisable to start work on integrating both data models in such a way as to combine the strengths of each of them. In practice, this would mean using the more general approach from each model used to express some reality that is the subject of data modelling. Only if it turns out that the connection of RailTopoModel 1.5 and RailSystemModel 1.2 or their successors into one common data model is not feasible, which may be mainly due to non-technical factors, a partial solution would be to try to use ontology to identify similar concepts, but this may be accompanied by the pitfalls outlined above.

5. Conclusions

The presented article aimed to analyse the latest versions within both development initiatives, i.e. the RailTopoModel version 1.5 and RailSystemModel 1.2, compare them with each other and find out what development they have undergone since their common predecessors, i.e. the RailTopoModel versions 1.0 and 1.1. It was found that although many basic principles remained, within each of the above initiatives, relatively fundamental changes were made to the respective data model, making the models more sophisticated, comprehensive and usable for a wider range of use cases, possibly offering new possibilities and greater flexibility in some areas of modelling.

Although these are generally beneficial matters, the fact remains that the original idea of defining one standardized common approach to data modelling of railway infrastructure was to certain extent disrupted by each of the development trends going their own way. This involves not only adding new extensions to the original data model, but in some cases also making adjustments that in practice mean that some facts are expressed in a different way in each of the models. Each of the data models has developed in a different direction and it is therefore regrettable that the concepts that were extended to the original RailTopoModel are no longer shared in one common

model, which would be a user-friendly state free from possible misunderstandings potentially arising from the confusion of similar terms of slightly different meaning across the data models. It would be desirable to make an effort to reunify the data models newly expanded with extensions that arose during their separate development. If this path is not feasible, it is possible to attempt to connect the models at the semantic level using ontologies that both data models already deal with in some way.

References

[1] UIC. Feasibility study: UIC RailTopoModel and data exchange format, 2013. [2026-03-24]. https://www.railtopomodel.org/files/download/RailTopoModel/270913_trafIT_FinalReportFeasibilityStudyRailTopoModel.pdf

[2] UIC. RailTopoModel v1.0, 2016. [2026-03-24]. https://www.railtopomodel.org/files/download/RailTopoModel/270415_uic_railtopomodel_v1-0_superseded.pdf

[3] UIC. RailTopoModel v1.1, 2017. [2026-03-24]. https://uic.org/rail-system/railtopomodel

[4] railML.org. RailTopoModel v1.2, 2019. [2026-03-24]. https://development.railml.org/railml/railtopomodel/-/blob/v1.2/pdf/RTM.pdf?ref_type=tags

[5] railML.org. RailTopoModel v1.4, 2022. [2026-03-24]. https://development.railml.org/railml/railtopomodel/-/blob/v1.4/pdf/RTM.pdf?ref_type=tags

[6] railML.org. RailTopoModel v1.5, 2024. [2026-03-24]. https://development.railml.org/railml/railtopomodel/-/blob/1.5/pdf/RTM.pdf?ref_type=tags

[7] UIC. RailSystemModel 1.2, 2022. [2026-03-24]. https://rsm.uic.org/doc/rsm/rsm-1-2

[8] railML.org. RailTopoModel state of development. [2026-03-24]. https://www.railtopomodel.org/state-of-development

[9] railML.org. RailTopoModel. [2026-03-24]. https://www.railtopomodel.org

[10] UIC. IRS 30100, 2016. [2026-03-24]. https://www.railtopomodel.org/files/download/RailTopoModel/180416_uic_irs30100.pdf

[11] GeeksForGeeks. Separation of Concerns (SoC), 2024. [2026-03-24]. https://www.geeksforgeeks.org/software-engineering/separation-of-concerns-soc

[12] railML.org. About railML. [2026-03-24]. https://www.railml.org/en/about-railml

[13] railML.org. RailSystemModel. [2026-03-24]. https://rsm.uic.org

[14] Shift2Rail. Shift2Rail projects. [2026-03-24]. https://rail-research.europa.eu/shift2rail-projects

[15] railML.org. railML 3.3, 2024. [2026-03-24]. https://development.railml.org/railml/version3/-/blob/3.3-final/documentation/railml3.html?ref_type=tags

[16] P. Leach, M. Mealling, R. Salz. A Universally Unique IDentifier (UUID) URN Namespace, 2005. [2026-03-24]. https://datatracker.ietf.org/doc/html/rfc4122

[17] C. Rahmig, A. Hlubuček, L. Zhuchyi, V.-P. Kolmorgen. Splitting and connecting of railway infrastructure network representations: A railml case study. In *5th SmartRaCon Scientific Seminar – Proceedings*, pp. 167–184. 2023.

[18] I. Dubielewicz, B. Hnatowska, Z. Huzar, L. Tuzinkiewicz. Domain modeling in the context of ontology. *Foundation of Computing and Decision Sciences* **40**(1):3–15, 2015. https://doi.org/10.1515/fcds-2015-0001

[19] W3.org. OWL. [2026-03-24]. https://www.w3.org/OWL

[20] W3.org. RDF. [2026-03-24]. https://www.w3.org/RDF

[21] W3.org. Turtle. [2026-03-24]. https://www.w3.org/TR/turtle

[22] A. Magnien, B. Janssen, G. Dessagne, P. Tane. The case for a federated digital model of the rail system *Global Railway Review*, 2022. [2026-03-24]. https://www.globalrailwayreview.com/article/138386/the-case-for-a-federated-digital-model-of-the-rail-system

[23] A. Magnien. RailSystemModel evolution. [2026-03-24]. https://rsm-evolution.org

[24] railMl.org. railML ontology. [2026-03-24]. https://www.railml.org/en/ontology

[25] UIC. Ontorail aggregated ontology. [2026-03-24]. https://app.ontorail.org/ontorail

[26] ERA. ERA vocabulary. [2026-03-24]. https://data-interop.era.europa.eu/era-vocabulary