Distributed Systems and Internet Technologies Lab
Distributed Data Intelligence and Technologies Lab
Department of Information Engineering (DINFO)
University of Florence
http://www.disit.dinfo.unifi.it

Km4City - The Knowledge Model 4 the City

Smart City Ontology (ENG)

Since 2018, the Km4City is maintained by Snap4City: https://www.snap4city.org also operated by DISIT lab and community.

Authors: Pierfrancesco Bellini, Paolo Nesi, Mirco Soderi

Referent coordinator: paolo.nesi@unifi.it

Knowledge base accessible as SPARQL entry point as shown from https://log.disit.org

OWL version accessible from: https://www.disit.org/6506 **[the download link is for version 1.6.7]**

https://www.disit.org/km4city/schema/ **[the download link is for version 1.6.7]**

https://www.disit.dinfo.unifi.it

Info from info@disit.org , snap4city@disit.org

Version 5.1, of this document

Referring to version 1.6.7 of the ontology

Date 26-08-2020

Questo documento in Italiano: https://www.disit.org/6461 **[the link is for version 5.1 of this document]**

This document in English: https://www.disit.org/5606 **[the link is for version 5.1 of this document]**

Aim

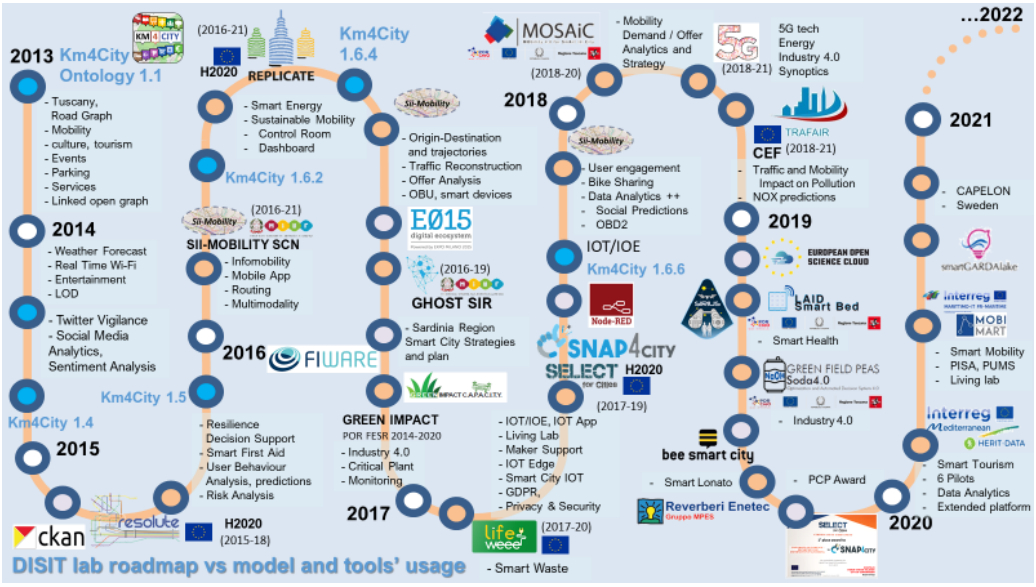
- **Provide a unique point of service** with integrated and aggregated data and tools for:
  - Qualified users: public administrations → developers
  - Operators: mobility, energy, SME, shops, … → developers
  - Final users → citizens, students, pendular, tourists
- **Problems**:
  - Support for semantic queries enabling the development of smart solutions:
    - Bike sharing
    - Smart parking
    - Resilience decision support system
    - What-if analysis
    - Routing
    - Pollutant prediction models
    - IOT integration
    - Anomaly detection
    - Etc.
  - Aggregated data are not available:
    - not semantically interoperable, heterogeneous for: format, vocabulary, structure, velocity, volume, ownership/control, access/license, …
    - As OD, LD, LOD, private data, …
  - Lack of Services and tools to make the adoption ***simple***

Km4City is in use for the Sii-Mobility SCN and for RESOLUTE H2020 projects. Km4City has been highly ranked by Ready4SmartCities FP7 CSA http://smartcity.linkeddata.es. On the other hand, all DISIT tools for smart city are agnostic with respect to the data model.

Km4City provides a collection of models and tools for smart city developers and administrators.

This work has been performed at DISIT lab for a number of smart city projects, see also for its use on:

- Snap4City https://www.snap4city.org
- SMART CITY at DISIT Lab: http://www.disit.org/6056
- Linked Open Graph: http://log.disit.org
- Service Map: http://servicemap.disit.org
- Slide on Km4City status, trends and tools http://www.disit.org/6669
- Smart city ingestion process document and process in place: http://www.disit.org/6058



A number of projects are using and/or contributing on Km4City technologies and solutions:

- Snap4city:
  - https://www.snap4city.org
- Sii-Mobility → mobility and transport, sustainability
  - http://www.sii-mobility.org/
- REPLICATE → ICT, smart City Control room, Energy, IOT
  - http://www.resolute-eu.org/
- RESOLUTE → Resilience, ICT, Big Data
  - http://replicate-project.eu/
- GHOST → Strategies, smart city
  - http://sites.unica.it/ghost/home/
- TRAFAIR → Environment & transport
  - https://trafair.eu/
- MOSAIC → mobility and transport
- WEEE Life → Smart waste, environment
  - https://www.lifeweee.eu/lifeWeee_it
- Smart Garda Lake → Castelnuovo del Garda
- 5G → Industry 4.0 vs SmartCity
  - https://www.corrierecomunicazioni.it/telco/5g/wind-tre-e-open-fiber-protagonisti-a-prato-del-debutto-italiano-del-5g/
- Green Impact → Industry 4.0, Chemical Plant
- SmartBed (laid) → smart health
- Green Field Peas (soda) → Industry 4.0, Chemical plant
- PISA MobiMart and Agreement → data aggregation, Living Lab
  - http://interreg-maritime.eu/web/mobimart
- Lonato del Garda → smart parking, environment
  - https://smartgardalake.it/progetto/
- Herit Data → tourism, culture and management
  - https://herit-data.interreg-med.eu/

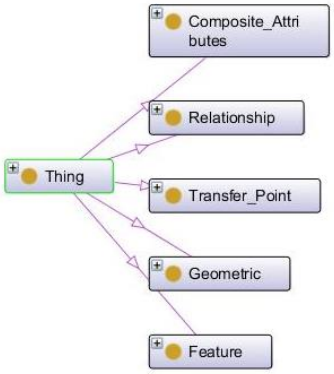
State of the Art

To interconnect the data provided by the Tuscany Region, the Open Data of the City of Florence, and the other Static and Real Time dataset, we started to develop a Knowledge Model, that allows to collect all the data coming from the city, related to mobility, statistics, street graph, etc.

A state of the art study on ontology related to Smart Cities, was carried out.

The only ontology presented as a "help to transform cities into Smart Cities" is SCRIBE, made by IBM, on which not much information is available online free of charge.

Among other ontologies that may be related to Smart Cities, we mention the OTN, Ontology of Transportation Networks.



This ontology defines the entire transport system in all its parts, from the single road/rail, to the type of maneuvering that can be performed on a segment of road or public transport routes. As you can see from the figure, the OTN includes the concepts expressed in the 5 main macro-classes, Composite Attributes (where classes like TimeTable, Accident, House_Number_Range, Validity_Period, Maximum_Height_Allowed can be found), relationships (in which we find the Manoeuvre), transfer points (macroclass which includes classes such as Road, Road_Element, Building, and others), geometry (i.e. classes Edge and Face Node), and features (which contains classes such as Railways, Service, Road_and_Ferry_Feature, Public_Transport).

On the net there are also many other ontologies related to sensor networks, such as the most recent version of the Semantic Sensor Network Ontology (see https://www.w3.org/TR/vocab-ssn/), which provides elements for the description of sensors and their observations, and FIPA Ontology which is more focused on the description of the devices and their properties both hardware and software (see http://www.fipa.org/specs/fipa00091/PC00091A.html#_Toc511707116). At today, the SSN ontology is reused in the Km4City Ontology for modelling the Internet-of-Things devices, their organization, and their detections.

Analyzing data made available by the PA (mainly by the Tuscany Region) it was found that, in relation to the roads graph, data could be easily mapped into large parts of the OTN, concerning to Street Guide; taking into account these observed similarities, we think it could be useful, in order to associate a more precise semantic to the various entities of our ontology, to make explicit reference to the OTN, to make the concepts clearer and more easily linkable to other, wanting to follow guidelines for the implementation of Linked Open Data (see http://www.w3.org/DesignIssues/LinkedData.html).

Due to the many similarities with the data model at our disposal, the OTN ontology has become one of the vocabularies used during the development of the Km4City model.

However, another interesting ontology more e-commerce oriented, is the Good Relations Ontology, a standardized vocabulary that allows to describe data related to products, prices, stores and businesses so that they can be included into already existing web pages and they can be understood by other computers. This way, products and services offered can increase their visibility into latest generation search engines, or recommendations systems and similar applications.

The possible integration between our Knowledge Model and the Good Relations ontology could be achieved at class level: the classes km4c:Entry and GoodRelations:Locality can be connected to interconnect the represented service to the geolocated couple address-housenumber through the object property km4c:hasAccess.

The schema.org ontology, which already incorporates the Good Relation project, has been central to the definition of events: the created class inherits the structure of the schema:Event class and it was then expanded based on information in our possession; the same ontology has been exploited for the definition of companies contact card.

Another ontology that has been used for the definition of the geometric forms is the geoSPARQL, whose integration put the Km4City Ontology on the route toward the use of geoSPARQL query language.

The Knowledge Model

The Km4City knowledge model enables and will enable interconnection, storage and interrogation of data from many different sources, such as various portals of the Tuscan region (MIIC, Muoversi in Toscana, Osservatorio dei Trasporti), Open Data and Linked Data, provided by individual municipalities (mainly Florence) or other organizations and projects (such as the Open Street Map). It is therefore evident that the resulting ontology is rather complex, and so it may be helpful to view it as consisting of a set of macro-classes (also called sections, or domains, within this document):

1. Administration: the first macroclass that is possible to discover, whose main classes are PA, Municipality, Province, Region, Resolution, District, Hamlet.
2. Street Guide: formed by classes like Road, Node, RoadElement, AdminidtrativeRoad, Milestone, StreetNumber, RoadLink, Junction, Entry, EntryRule, Maneuver, Lanes, and Restriction.
3. Points of Interest: includes all services and activities that may be useful to the citizen, and that one may have the need to reach. The classification of individual services and activities will be based on classification previously adopted by the Tuscany Region. Digital Location and scheduled Events (real time data), from the municipality of Florence, are also included in this macroclass.
4. Local Public Transport: currently we have access to data relating to scheduled times of the leading Local Public Transport, the graph rail, and real-time data relating to ATAF services. This macroclass is then formed by many classes like TPLLine, Ride, Route, AVMRecord, RouteSection, BusStopForeast, Lot, BusStop, RouteLink, TPLJunction.
5. Sensors: the Km4City Ontology is suitable to represent the traffic sensors, the traffic detections, the free places in the car parks, the real-time detections of critical events and emergencies, the weather conditions and forecasts, and so on.
6. Temporal: macroclass pointing to include concepts related to time (time instants and time intervals) in the ontology, so that you can associate a timeline to the recorded events and can be able to make predictions.
7. Metadata: set of triples associated with the context of each dataset; such triples collect information related to the license of the dataset, to the ingestion process, if it is fully automated, to the size of the resource, a brief description of the resource and other info always linked to the resource itself and its ingestion process.
8. Internet of Things: a new section has been included in the Km4City Ontology since version 1.6.5, that is aimed at modelling generic sensors, actuators, their brokers, and the types of measurements that they are capable to perform. In this 1.6.7 a strong improvement has been done in this area to cope with SmartHome and Smart Industry, Industry 4.0 solutions:

a. Industry 4.0: https://www.snap4city.org/drupal/node/369
b. Smart Home: https://www.snap4city.org/drupal/node/617

Let us now analyze one by one the different macro-classes identified.

The first macro-class, namely the Administration macro-class, is composed as shown in Figure 1.

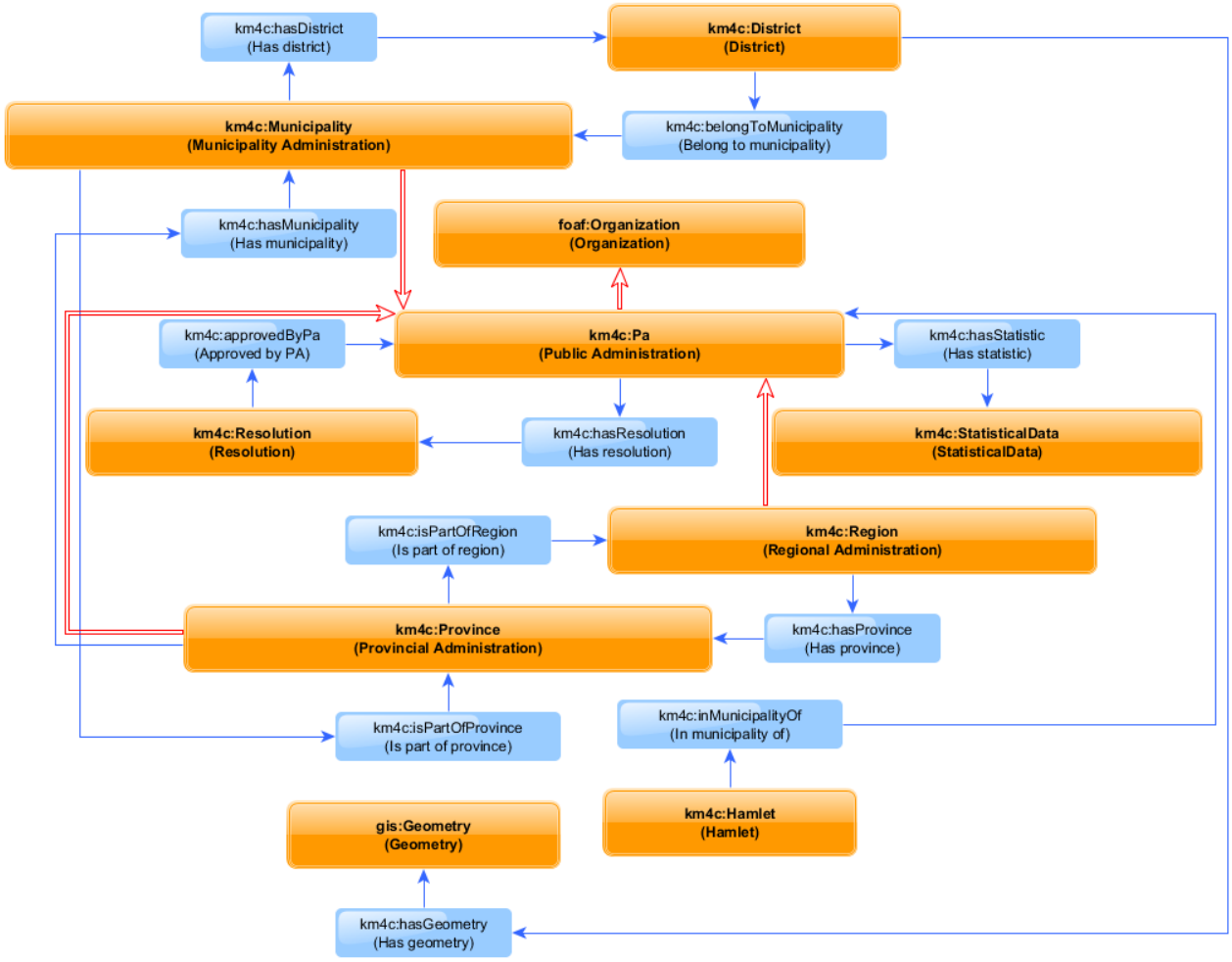


Figure 1 Public Administrations

The main class is km4c:PA, which has been defined as a subclass of foaf:Organization, link that helps us to assign a clear meaning to our class. The 3 subclasses of km4c:PA are automatically defined according to the restriction on their object properties (represented in blue in Figure 1). For example, the Region class is defined as a restriction of the class PA on the object property km4c:hasProvince, so that only the PAs that possess provinces, can be classified as a regions. Another example: to define the PA elements that make up the Municipality class was instead used a restriction on the object property km4c:isPartOfProvince, so if a PA is not assigned to a province, it cannot be considered a municipality.

During the establishment of the hierarchy within the class PA, for each step were defined pairs of inverse object properties: km4c:hasProvince and its inverse km4c:isPartOfRegion, km4c:hasMunicipality and its inverse km4c:isPartOfProvince.

Connected to the class km4c:PA through the object property km4c:hasResolution, we can find the km4c:Resolution class, whose instances are represented by the resolutions passed by the various PA note. The object property km4c:hasResolution has an inverse property, namely the km4c:approvedByPa.

The neighborhoods in which a city can be administratively divided are represented by the km4c:District class; this class is directly connected to the class km4c:Municipality through a pair of inverse object properties: the km4c:belongToMunicipality, and the km4c:hasDistrict. Semantically similar to the km4c:District is the km4c:Hamlet class, that represents any named subpart of a Municipality, not only the districts that are officially instituted in the large towns. The Hamlet is connected to the Municipality where it locates, through the object property km4c:inMunicipalityOf.

The last class in this macroclass is km4c:StatisticalData: given the large amount of statistical data related both to the various municipalities in the region and to each street, that class is shared by both the Administration and Street Guide macro-classes. As we will see in the next macro-class description, the class km4c:StatisticalData is connected to both km4c:Pa and km4c:Road through the object property km4c:hasStatistic.

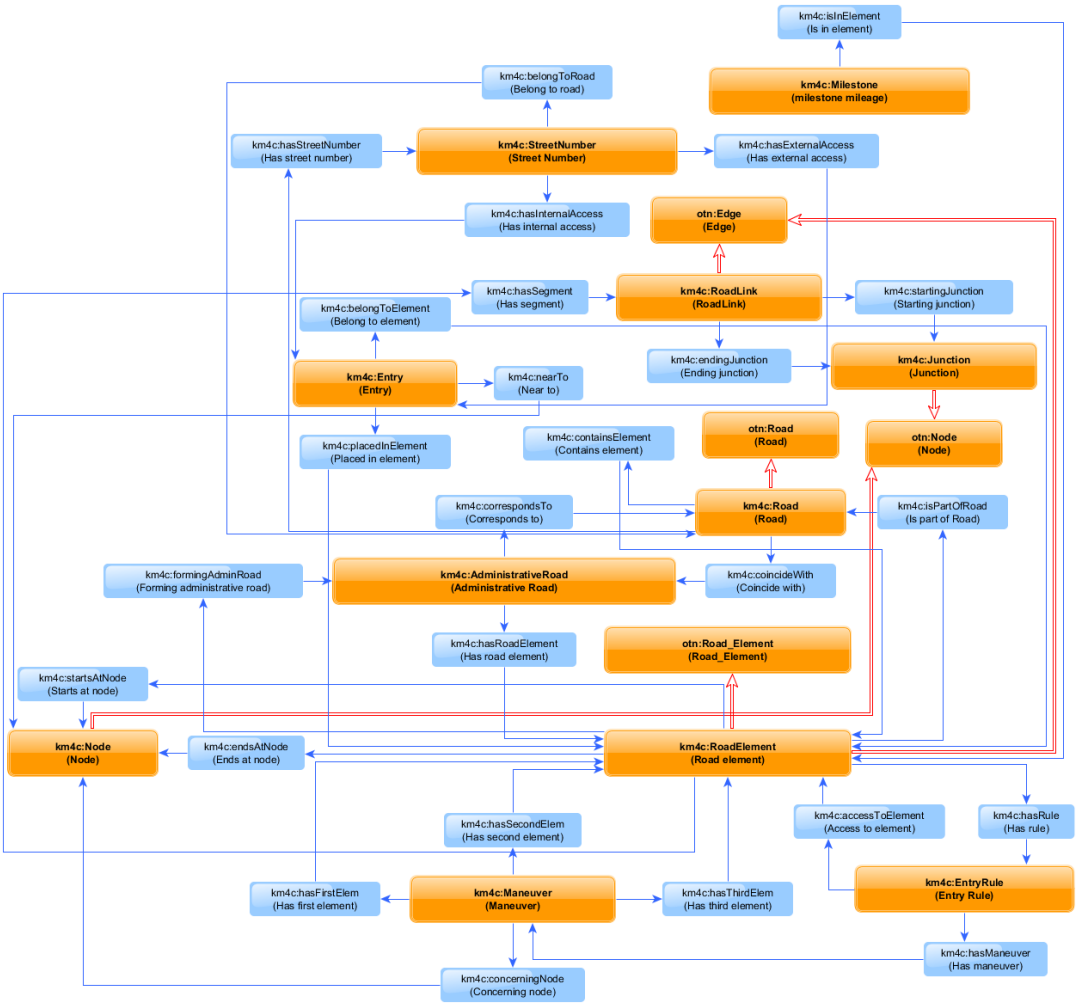


Figure 2 Street Guide

The macro-class Street Guide is summarized in Figure 2.

Clearly this macro-class is the most complex of the Ontology, since it represents the entire street graph, various house numbers, their accesses, but also rules of access to various roads, and finally permitted maneuvers.

The main class, in the middle of the Street Guide macro-class, is km4c:RoadElement, which is defined as a subclass of the corresponding element in the ontology OTN, i.e. Road_Element. Each RoadElement is delimited by a start node and an end node, detectable by the object properties km4c:startsAtNode and km4c:endsAtNode, which connect the class object of the class km4c:Node. Some restrictions have been specified in the km4c:RoadElement class definition, related to the km4c:Node class: a road element must have both the km4c:startsAtNode and the km4c:endsAtNode object properties, both with a cardinality exactly equal to 1. One or more road elements make up a road: the class km4c:Road is in fact defined as a subclass of the corresponding class in the OTN, i.e. the homonymous Road, and with a cardinality restriction on km4c:containsElement object property (whose inverse property is km4c:isPartOfRoad), which must be at minimum equal to 1, i.e. a road cannot exist that does not contain at least one road element. Also the class km4c:AdministrativeRoad, which represents the administrative division of the roads, is connected to the class km4c:RoadElement through two inverse object properties, namely the km4c:hasRoadElement and the km4c:formAdminRoad, while it is connected with only one object property to the km4c:Road class, namely the km4c:coincideWith. Some further details about the relationship that exists among km4c:Road, km4c:AdministrativeRoad and km4c:RoadElement: a Road's instance can be connected to multiple instances of km4c:AdministrativeRoad (e.g. if a road crosses the border between the two provinces), but the opposite is also true (e.g. when a road crosses a provincial town center and assumes different names), i.e. there is a N:M relationship between the two classes.

On each road element it is possible to define access restrictions identified by the class km4c:EntryRule, which is connected to the class km4c:RoadElement through two inverse object properties, i.e. km4c:hasRule and km4c:accessToElement. The km4c:EntryRule class is defined with a restriction on the minimum cardinality of the object property km4c:accessToElement (set equal to 0), since in most cases one element has an associated road, but in some exceptional cases, there could be no association. Access rules allow to define uniquely a permit or limitation access, both on road elements (for example due to the presence of a ZTL) as just seen, but also on maneuvers; for this reason, the class km4c:Maneuver and the class km4c:EntryRule are connected by the km4c:hasManeuver object property. The term maneuver refers primarily to mandatory turning maneuvers, priority or forbidden, which are described by indicating the involved road elements. By analyzing the data from the roads graph we have verified that in rare cases maneuvers involve three different road elements. Then, to represent the relationship between the classes km4c:Maneuver and km4c:RoadElement, three object properties were defined: km4c:hasFirstElem, km4c:hasSecondElem and km4c:hasThirdElem, in addition to the object property that binds a maneuver to the junction that is interested, namely the km4c:concerningNode property, named this way because a maneuver always takes place in the proximity of a node. In the km4c:Maneuver class definition there are cardinality restrictions: km4c:hasFirstElem and km4c:hasSecondElem have their minimum cardinality set to 1, while km4c:hasThirdElem has its maximum cardinality set to 1, since for the maneuvers that affect only two road elements, this last object property is not defined.

As previously mentioned, each road element is delimited by two nodes (or junctions), the starting one and the ending one. It was then defined the km4c:Node class, subclass of the OTN Node class. The km4c:Node class has been defined with a restriction on the data properties geo:lat and geo:long, both coming from the class SpatialThing that can be found in the Geo WGS84 Ontology: indeed, each node must have one and only one pair of geospatial coordinates.

The km4c:Milestone class represents the kilometer stones that are placed along the administrative roads, i.e. the elements that identify the precise value of the mileage at that point, or if you prefer, the distance from the beginning of the road. A milestone is expected to be associated with a single AdministrativeRoad, and it is why the object property km4c:placedInElement has its maximum cardinality set to 1. Also the km4c:Milestone class is defined as a subclass of geo:SpatialThing, but this time the presence of the coordinates is not mandatory (the maximum cardinality only is set).

The StreetNumber is used to define an address, and it always is logically related to at least one access (Entry), in fact every street number always corresponds to a single external access, which can be direct or indirect, and possibly to an internal access. Looking at this relationship from the point of view of the access, you find that each access is logically connected to at least one street number.

The StreetNumber can be related to the Road and RoadElement simply leveraging the object properties that we have already presented. Nevertheless, the shortcut properties km4c:placedInElement and km4c:belongToRoad can be found in the Km4City Ontology. This information is actually redundant and if it is deemed to be appropriate, one may delete the connection to the km4c:Road, only keeping the connection to the km4c:RoadElement. For this reason, the object property km4c:belongToRoad has an inverse, namely the km4c:hasStreetNumber property. Nevertheless, the StreetNumber class has a cardinality restriction on the object property km4c:belongToRoad, that is required to be present. Therefore, making the above outlined choice would imply the need of modifying the Ontology.

The km4c:Entry class also is connected to both the street number and road element where is located. The relationship between km4c:Entry and km4c:StreetNumber, is defined by two object properties, namely the km4c:hasInternalAccess and the km4c:hasExternalAccess. A street number must have one and only one external access, and it can have an internal access. Also the km4c:Entry class is defined as a subclass of geo:SpatialThing, and it is possible to associate a maximum of one pair of coordinates geo:lat and geo:long to each instance (restriction on the maximum cardinality of the two data properties of the Geo ontology set to 1).

The road elements are connected to two different Public Administrations through the object properties km4c:ownerAuthority and km4c:managingAuthority, which as the names suggest, respectively represent the Public Administration that has extensive administrative, and the Public Administration that manages the road element.

From a cartographic point of view, however, each road element is not a straight line, but a broken line, which will follow the actual course of the road. To represent this situation, the classes km4c:RoadLink and km4c:Junction have been added: we retrieve the set of the points that outline each road element, and each of these points is modelled as a Junction (defined as a subclass of geo:SpatialThing, with a compulsory single pair of coordinates). Each small segment between two Junction is instead an instance of the class km4c:RoadLink, which is defined by a restriction on the object properties km4c:endJunction and km4c:startJunction (both must have cardinality equal to 1), which connect the two classes.

The modelling of the road infrastructure has been improved in release 1.6.5 with the inclusion of concepts aimed at representing the lanes and the traffic restrictions. An overview of the improvements is provided in Figure 3.

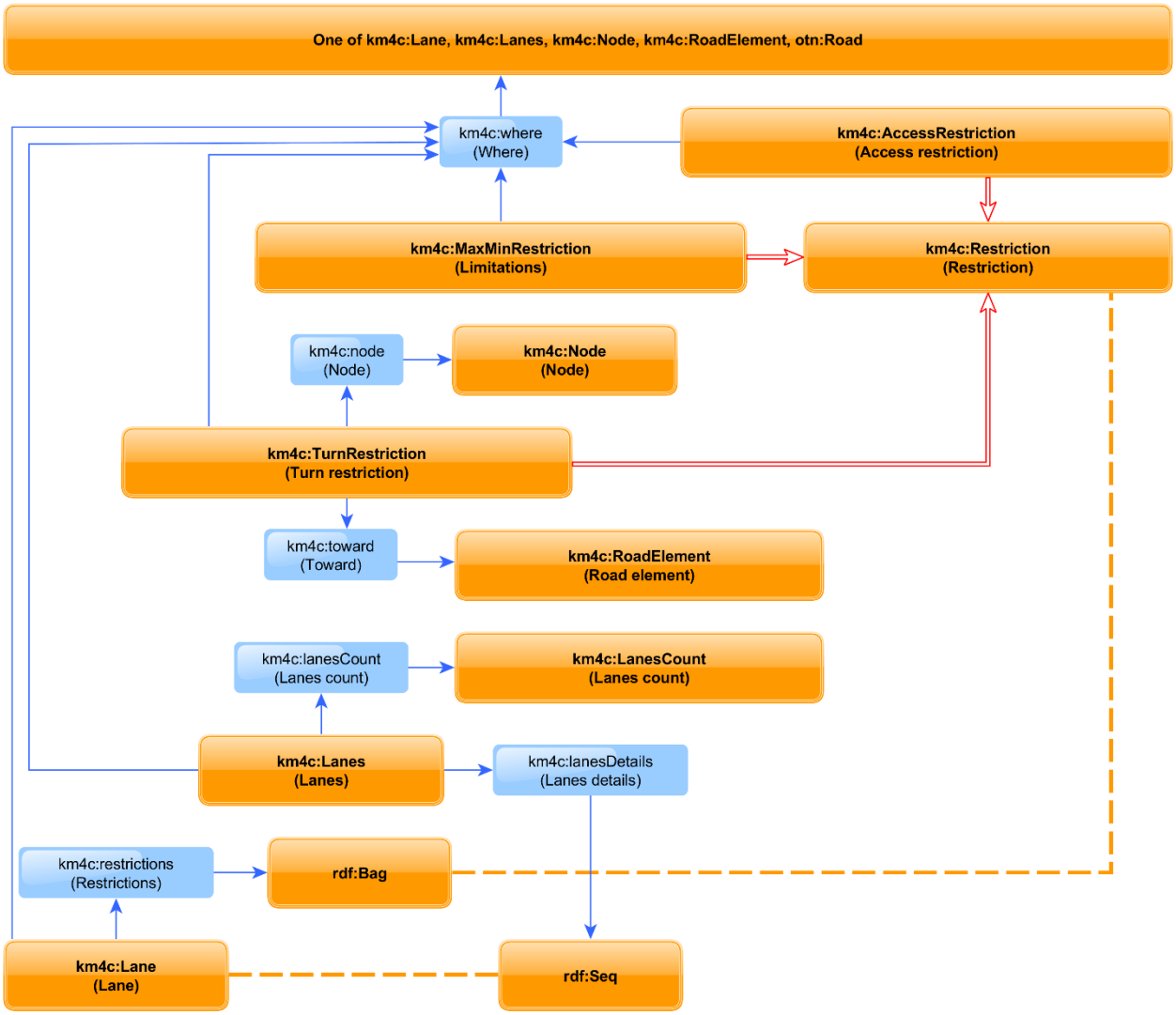


Figure 3 Recent improvements to the street graph modelling

As for the modelling of the lanes, the central concept is the km4c:Lanes. One single instance of the concept km4c:Lanes represents the lanes that can be found marked on a road or a segment of road, in a specific traffic direction that is expressed through the property km4c:direction. The property km4c:where indicates where the lanes are marked, and its range can be a km4c:Road, or a km4c:RoadElement. Its inverse is the property km4c:lanes. Since some of the lanes could be reserved to specific typologies of vehicles, and we need to represent this information, each instance of km4c:Lanes is expected to have a property km4c:lanesCount whose range is an instance of the concept km4c:LanesCount whose properties borrow their names from the different typologies of vehicles that could have dedicated lanes, and take each a numeric (integer) value that indicates how many of the lanes are dedicated to the specific typology of vehicles. A km4c:undesignated property is also typically set for a km4c:LanesCount instance, and it indicates how many of the lanes are not reserved to any specific typology of vehicles. It could sometimes happen that one or more of the lanes have peculiar restrictions, apart from the possible reservation to specific typologies of vehicles. As an example, some lanes could have a speed limit, while others could have a different speed limit, or not to have a speed limit at all. Since we need to represent these restrictions, the property km4c:lanesDetails has been introduced and can be found set for a km4c:Lanes instance. Its range is a rdf:Seq of km4c:Lane instance, each of which represents one of the lanes. The rdf:Seq is a sorted set.

The lanes appear within it from left to right respect to the driving direction. Each instance of the concept km4c:Lane is expected to have a km4c:where property that indicates the km4c:Lanes instance of which the km4c:Lane is part. Also, each km4c:Lane instance is expected to have a km4c:position property, a numeric (integer) property that indicates the position of the lane from left to right with respect to the driving direction. Also, an instance of km4c:Lane could have a km4c:turn property, a plain text that indicates the mandatory maneuver that drivers will be required to perform once they will reach the end of the lane. Finally, a km4c:Lane instance could have a km4c:restrictions property, that is filled by a rdf:Bag of km4c:Restriction instances. An rdf:Bag is an unsorted set of instances. A km4c:Restriction represents a traffic restriction. Traffic restrictions can also be found applied to km4c:Road and km4c:RoadElement instances. In those cases, the km4c:where property set on the km4c:Restriction indicates the km4c:Road or the km4c:RoadElement that is subject to the restriction. Three types of restrictions are modelled, i.e. the turn restrictions, the access restrictions, and the *range* restrictions. The turn restrictions are modelled through the concept km4c:TurnRestriction, and they represent the mandatory and the forbidden maneuvers. For each km4c:TurnRestriction instance, a property km4c:where is expected to be set that indicates the origin km4c:Road or km4c:RoadElement, a property km4c:toward is expected to be set that indicates the destination km4c:Road or km4c:RoadElement, and a property km4c:restriction is expected to be set that indicates whether the maneuver is mandatory or forbidden. Other properties can be found if the restriction only applies in specific days and times, or if it only applies to specific categories of vehicles, and so on. The access restrictions are represented through the concept km4c:AccessRestriction and they represent the prohibition of accessing a road or a segment of road. They often apply to one or more categories of vehicle only, in which case the property km4c:who is set for indicating the categories of vehicle to which the restriction applies. Also, they often apply to a specific traffic direction, in which case the property km4c:direction is set for indicating the traffic direction to which the restriction applies. All instances of km4c:AccessRestriction are expected to have a km4c:access property that provides the correct interpretation of the restriction, indicating if the targeted typologies of vehicles are granted an exclusive permission of accessing the road or segment of road, or it the access is forbidden for them. For those access restrictions that only apply under peculiar conditions, such as in some days and/or times only, the property km4c:condition is also set that describes the condition. Finally, the range restrictions could be related to the weight or size of the vehicles, but they also could prescribe a maximum and/or minimum speed, and other. They are modelled through the concept km4c:MaxMinRestriction. The instances of this concept are expected to have a km4c:where property set that indicates the road, segment of road or lane to which the restriction applies. They also are expected to have a km4c:what property set that describes to what the range restriction refers to (speed, width, height, weight, or what other), and a km4c:limit property where the range itself is indicated. If the restriction applies to only one of the traffic directions, the property km4c:direction is set that indicates the traffic direction to which the restriction applies. Finally, if the restriction only applies under peculiar conditions, such conditions are represented through the property km4c:condition.

For the third macro-class, the Points of Interest, depicted in Figure 4, a generic class Service has been defined, and a set of classes and subclasses has been defined, getting inspiration from the ATECO classification (the ISTAT code classification of economic activities). In an early version of the Ontology, the classes and subclasses had instead been defined based on a regional classification of the business activities. Those services that had been imported in the Knowledge Base at that time, have been reclassified.

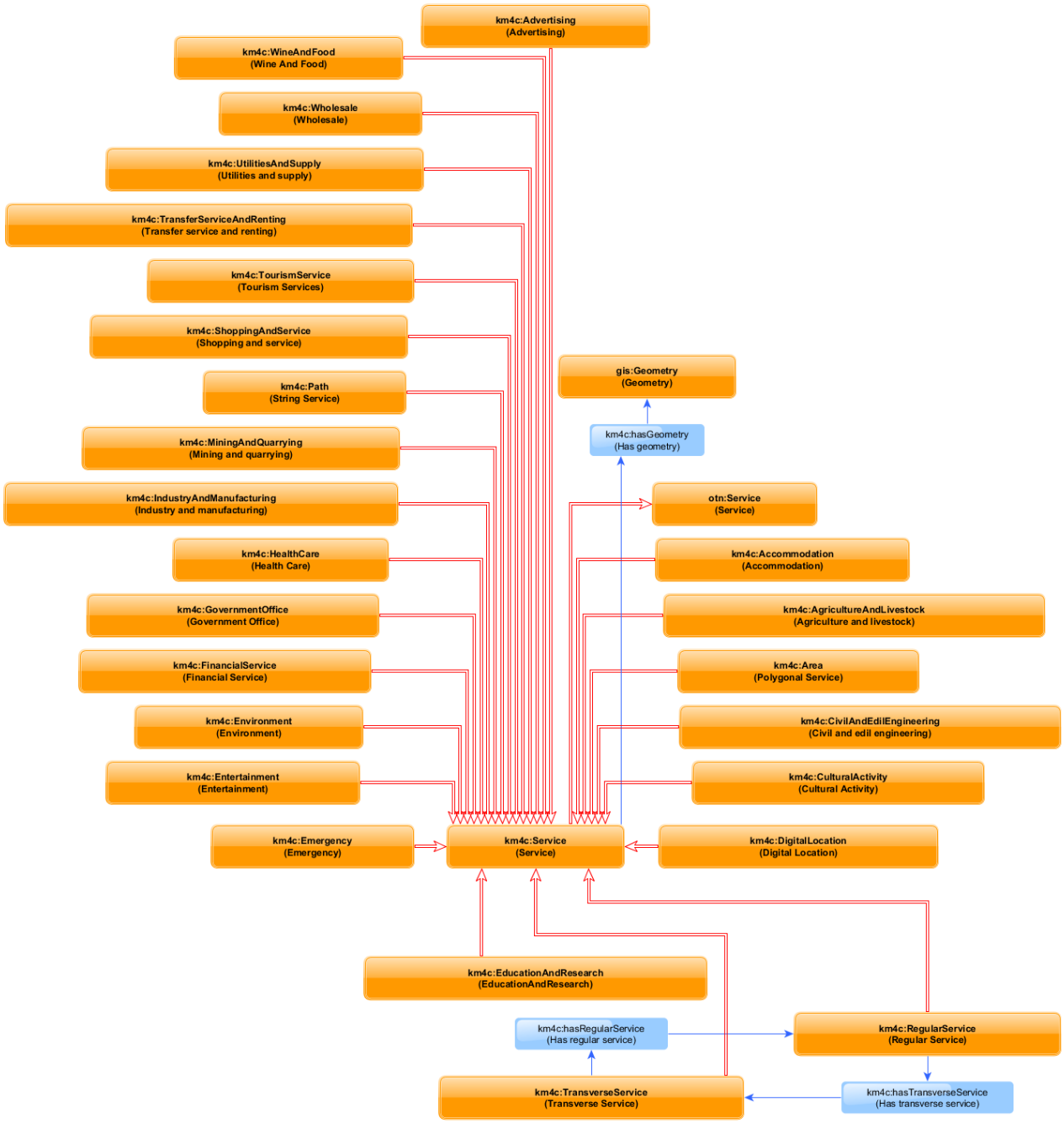


Figure 4 Points of Interest

Nevertheless, the new categorization has not been created with a total correspondence with the entire list of the ATECO codes, but it has been created with the purpose of providing more details for those subclasses that are of major interest for our purposes (service providers, retail sales, etc.), staying more general for those categories that are less relevant for our purposes, such as wholesale, industry, and so on).

Currently, the following classes have been identified: Accommodation, GovernmentOffice, TourismService, TransferServiceAndRenting, CulturalActivity, FinantialService, ShoppingAndService, Healthcare, EducationAndResearch, Entertainment, Emergency, WineAndFood, IndustryAndManufacturing, AgricultureAndLivestock, UtilitiesAndSupply, CivilAndEdilEngineering, Wholesale, Advertising, MiningAndQuarrying and Environment. Each of them has a number of subclasses that define which types of services belong to this class: for example, for the class Accommodation the following subclasses have been

defined: Holiday_village, Hotel, Summer_residence, Rest_home, Hostel, Farm_house, Beach_resort, Agritourism, Vacation_resort, Day_care_centre, Camping, Boarding_house, Other_Accommodation, Mountain_shelter, Religiuos_guest_house, Bed_and_breakfast, Historic_residence and Summer_camp. Notable in the Environment class are the Noise_level_sensor class, introduced in late 2018 and immediately used for representing noise level sensors deployed in Helsinki, and the People_counter class, introduced in 2019 for representing for example pax counters (ESP32, …). Also notable in the TourismService class is the Digital_signage class (see also https://it.wikipedia.org/wiki/Digital_signage) introduced in late 2019 to represent a set of informative digital displays deployed in the city of Florence and its surrounding.

Currently, 512 subclasses can be found in the Km4City Ontology, divided as follows:

Accommodation	18
FinancialService	10
Environment	14
MiningAndQuarrying	5
Advertising	2
Wholesale	10
CivilAndEdilEngineering	9
UtilitiesAndSupply	30
AgricultureAndLivestock	7
IndustryAndManufacturing	54
EducationAndResearch	33
Entertainment	27
Emergency	14
TourismService	15
HealthCare	25
WineAndFood	21
CulturalActivity	26
ShoppingAndService	140
GovernmentOffice	15
TransferServiceAndRenting	39

The Service class also has a pair of sub-classes that allow to identify more quickly non-point services. These classes, namely the km4c:Path and the km4c:Area, represent respectively those services that can be represented through a broken line on a map, and those services that can be represented as a polygon. Therefore, each not-punctual service necessarily falls into one of these two classes.

It was also defined a further services categorization in regular services (km4c:RegularService) and transversal services (km4c:TransversalService). This categorization was proved to be necessary since with the inclusion of new services, some services were discovered to be related to other services. As an example, a restaurant (regular service) can offer a free WiFi service (transversal service) to its customers. To handle this situation, it is therefore necessary to connect the two services to each other, but it should also be possible to identify only the Wifi, as a service. It is therefore clear that there are two types of services, primary (or regular) services, and secondary (or trasversal) services. Transversal services are connected to regular services through the object property km4c:hasRegularService, and through its inverse km4c:hasTransversalService.

A further part of the Point of Interest macroclass is represented by the integration of Digital Locations provided by the Municipality of Florence. The Digital Location data are collected and published by the municipality of Florence as Linked Data, representing locations deemed attractive for the city itself, such as jogging paths, green areas and gardens, monuments and historic buildings, and so on, for a total of approximately 3200 different elements. These Digital Locations have been modelled as km4c:Service instances. For those Digital Locations that already existed in the Knowledge Base since they had been already imported from sources other than the collection of the Digital Locations, two instances of km4c:Service can be found that represent the same service in the real world. The two instances are related each other through the property owl:sameAs.

The introduction of geoSPARQL ontology, has also allowed to associate to each service a geometric shape: indeed, a service can be identified as a point (in case of a store, a monument, etc.), or as a path (a broken line that connects various points of interest, such as a tourist route) or as an area (for example, a polygon that corresponds to a garden or a park).

Another class that was added to this macroclass is the km4c:Event class, which inherits most of its properties from the schema:Event class, but not only: this class has indeed many additional properties and it is also connected through the scheme:location object property to the km4c:Service instance that represents the place where the event takes place.

The fourth macro-class, the Local Public Transport, depicted in Figure 5, presents data and classes inspired to the metropolitan bus lines of Florence tramway and rail network. Nevertheless, the model is proved to be applicable to the urban bus lines distribution in territories other than the Municipality of Florence, and the suburban bus lines too.

We analyze first the part related to the metropolitan transport in Florence which includes bus transportation and tramway.

Each TPL lot, represented by the km4c:Lot class, is composed of a number of bus or tram lines (class km4c:PublicTransportLine), and this relationship is represented by the object property km4c:isPartOfLot, which connects each instance of km4c:PublicTransportLine to the corresponding instance of km4c:Lot. The km4c:PublicTransportLine class is defined as a subclass of OTN:Line. Each line includes at least one race, identified through a code provided by the TPL company; the km4c:PublicTransportLine class is indeed connected to the class km4c:Ride through the object property km4c:scheduledOnLine, on which is also defined as a limitation of cardinality exactly equal to 1 (a race must be connected to one and only one line).

Each race follows at least one path (the 1:1 match has not been tested on all data, as soon as it occurred, eventually will include a restriction on the cardinality of the object property that connects the two classes, namely km4c:OnRoute), and the paths can be in a variable number even if referring to a same line: in most cases are 2, i.e. the forward path and backward path, but sometimes also come to be 3 or 4, according to possible extensions of paths deviations or maybe performed only in specific times. The object property km4c:onRoute is also used to connect the class km4c:Ride, representing the individual races on a specific path defined by the TPL operator. Through the object property gis:hasGeometry, instances of the class km4c:Route, can instead be connected to the instance of gis:Geometry class, that contains the line string representing the actual path of the route.

Each path is considered as consisting of a series of road segments delimited by subsequent stops: to model this situation, it was decided to define two object properties linking km4c:Route and km4c:RouteSection

classes, km4c:hasFirstSection and km4c:hasSection, since, from a cartographic point of view, wishing to represent the path that follows a certain bus, knowing the first segment and the stop of departure, you can obtain all the other segments that make up the complete path and starting from the second bus stop, identified as a different stop from the first stop, but that it is also contained in the first segment, we are able to reconstruct the exact sequence of the stops, and then the segments, which constitute the entire path. For this purpose has been defined also the object property km4c:hasFirstStop, which connects the classes km4c:Route and km4c:BusStop .

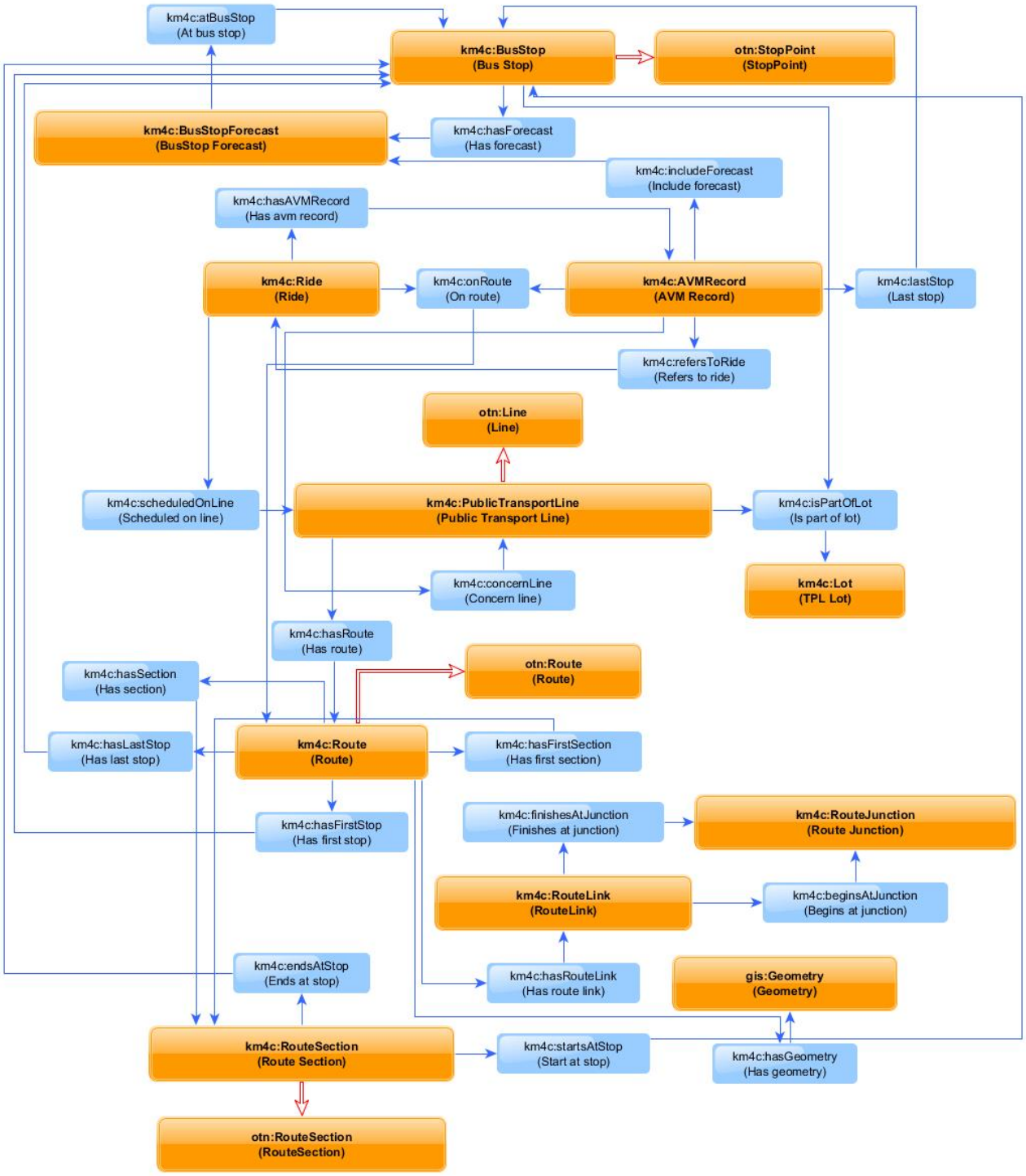


Figure 5 Local Public Transport

Applying the same type of modeling used for road elements, two object properties have been defined, km4c:endsAtStop and km4c:startsAtStop, to connect each instance of km4c:RouteSection to two instances of the class km4c:BusStop, class in turn defined as a subclass of OTN:StopPoint. Each stop is also connected to the class km4c:Lot, through the object property km4c:isPartOfLot, with a 1:N relation because there are stops shared by urban and suburban lines so they belong to two different lots.

Possessing also the coordinates of each stop, the class km4c:BusStop was defined as a subclass of geo:SpatialThing, having one and only one pair of geospatial coordinates.

Wishing then to represent to a cartographic point of view the path of a bus, i.e. a km4c:Route instance, we need to represent the broken line that composes each stretch of road crossed by the means of transport itself and to do so, the previously used modeling has been reused to the road elements: we can see each path as a set of small segments, each of which delimited by two junctions: were then defined km4c:RouteLink and km4c:RouteJunction classes, and the object properties km4c:beginsAtJunction and km4c:finischesAtJunction. The km4c:RouteLink class was defined as a cardinality restriction of both the mentioned object properties, imposing that it is always equal to 1. The km4c:Route class is instead connected to the km4c:RouteLink class through km4c:hasRouteLink object property.

The km4c:BusStop class is also connected to the km4c:Road class belonging to the Street Graph macro-class, through the object property km4c:isInRoad, thanks to which it is more simple to localize at municipality level each individual stops.

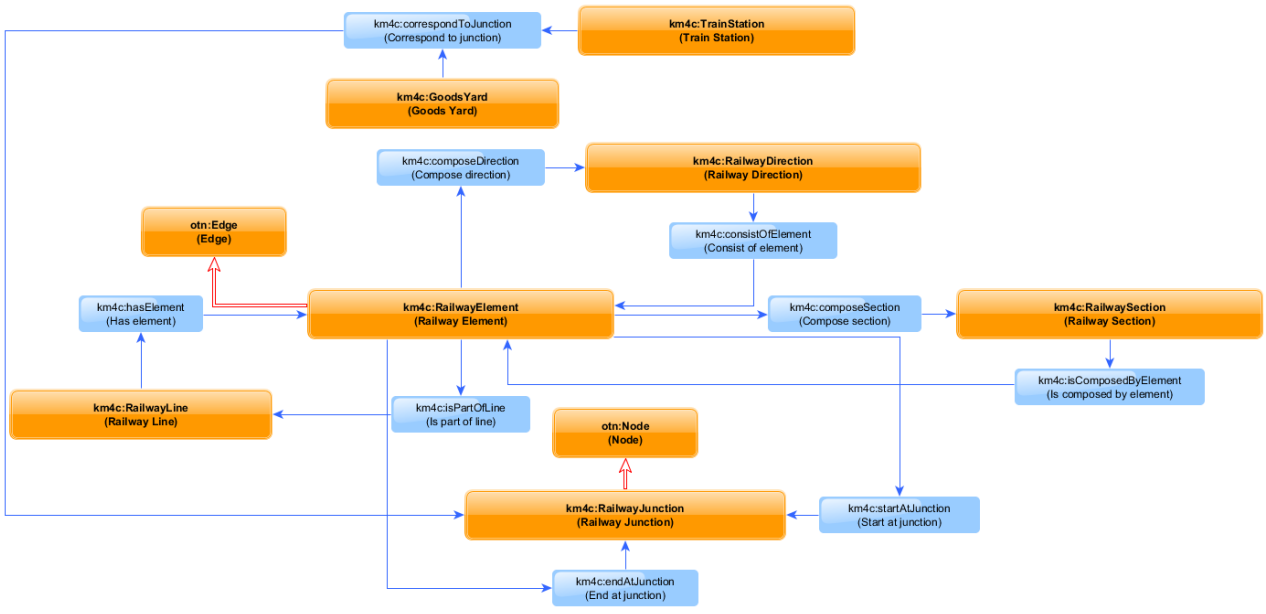


Figure 6 Railway Graph

The part relating to the Railway Graph, depicted in Figure 6, is mainly formed by the km4c:RailwayElement class (defined as a subclass of the OTN:Edge), whose instances represent each a single railway element; each element can compose a railway direction, that is a railway line having particular characteristics of importance for traffic volume and transport relations on which it takes place, and that links the main nodes or centers of the entire rail network, or a railway section (section of the line in which you can find only one train at a time that is usually preceded by a "protective" or "block") or a railway line (i.e. the infrastructure that allows trains or other railway convoys to travel between two places of service). In addition, each rail element begins and ends at a railway junction, i.e. an instance of the class km4c:RailwayJunction, defined

as a subclass of the OTN:Node. There is also the km4c:TrainStation class that is just a train station, and the class km4c:Goodsyard that corresponds to a freight station; usually both the two classes correspond to a single instance of the km4c:RailwayJunction class.

Also the km4c:RailwayElement class is connected to gis:Geometry class, because each railway element can be seen as a line string that represents the true shape of the railway line in that section.

Since February 2019, the ontology includes one further concept that completes the framework of public transport, namely the subway station, represented through the Subway_station class, introduced on the basis of the data collected regarding the city of Helsinki and in particular the HLS public transport agency in the context of the Snap4City project. Still related to transportation, but not merely public transportation, is also the class km4c:Traffic_flasher, that represents those blinking traffic lights typically made up of a single lamp that are sometimes used for delivering warnings related to heavy traffic, queues, and similar. A similar function is sometimes covered (typically in highways) by variable message signs, that also have a dedicated concept in the Ontology, namely the km4c:Variable_message_sign. Other traffic lights that have a dedicated concept in the Ontology are the km4c:Tram_traffic_light.

The Km4City Ontology also includes a comprehensive set of concepts aimed at modelling the real-time events and detections, and the devices that perform such detections and fire such events, with their structuring and organization. It's the *Sensors* macro-area.

A notable subdomain of this macro-area of the Ontology addresses the car parks, with the devices that are capable to detect the quantity of free spaces, and their real-time detections. An overview of the subdomain is provided in Figure 7.

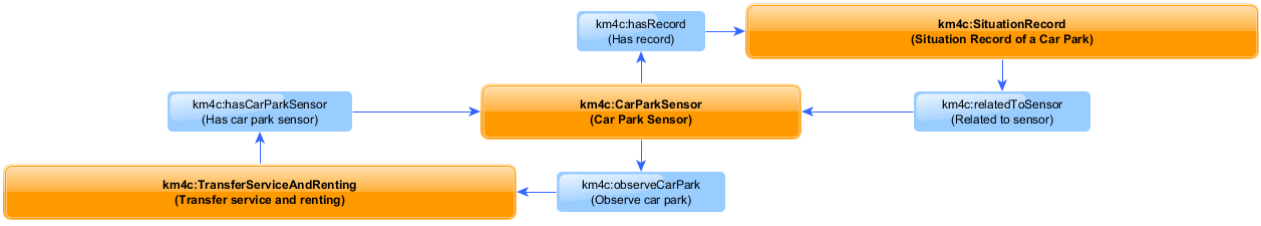


Figure 7 Car parks and their sensors

The km4c:TransferServiceAndRenting class, in fact, is connected to the km4c:CarParkSensor class, which represents the sensor installed in a given parking and which will be linked to instances of the km4c:SituationRecord class, which represent the state of a certain parking at a certain instant; the first link, i.e. the one between km4c:TransferServiceAndRenting class and km4c:CarParkSensor class, is realized through two inverse object properties, km4c:observe and km4c:isObservedBy, while the connection between the km4c:CarParkSensore class and the km4c:SituationRecord class is performed via the inverse object properties km4c:relatedTo and km4c:hasRecord. The class km4c:SituationRecord allows to store information about the number of free and occupied parking spaces, in a given moment (also recorded) for the main car parks in Tuscany region.

Dedicated concepts also can be found that relate to the weather detection, prediction, and reporting activities. More specifically, the Km4City Ontology includes the low level detections of the sensing devices, the weather forecasts (that are modelled as instances of km4c:WeatherPrediction), and the structuring of such detections and forecasts in km4c:WeatherReport instances. The spatial horizon of all this is modelled

through the geospatial coordinates that can be found set on the sensing devices, and through the property km4c:refersToMunicipality and its inverse km4c:hasWeatherReport that keep connected the weather forecasts with the territories they refer to. An overview of all this is provided in Figure 8.

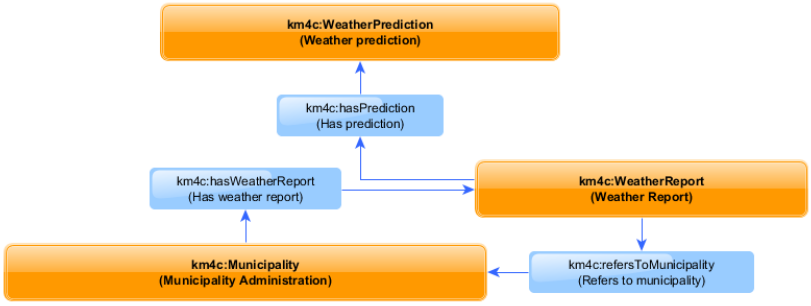


Figure 8 Weather forecasts

The weather forecasts are available for different areas (and thus connected to the class Municipality), thanks to LAMMA. This consortium will update each municipality report 1 or 2 times a day and every report contain a forecast for 5 days divided into range, which have a greater precision (and a higher granularity) for the nearest days until you get to a single daily forecast for the 4th and 5th day. This situation is in fact represented by the km4c:WeatherReport class connected to the km4c:WeatherPrediction class via the object property km4c:hasPrediction. The km4c:Municipality class is instead connected to a report through two inverse object properties, the km4c:refersToMunicipality and the km4c:hasWeatherReport.

Dedicated concepts can also be found that address the traffic monitoring, one of the key challenges of all urban environments. The sensing devices are modelled through the km4c:SensorSite concept, and their position is represented through the property km4c:placedOnRoad, that is filled by a km4c:Road instance. The traffic sensors produce detections that are modelled as km4c:Observation instances, and more specifically as km4c:TrafficObservation instances. Dedicated concepts can be found for modelling the many different types of direct and indirect (computed) measurements of the city traffic, such as the traffic speed, the traffic headway, the traffic flow, and the traffic concentration. See Figure 9 for an overview.

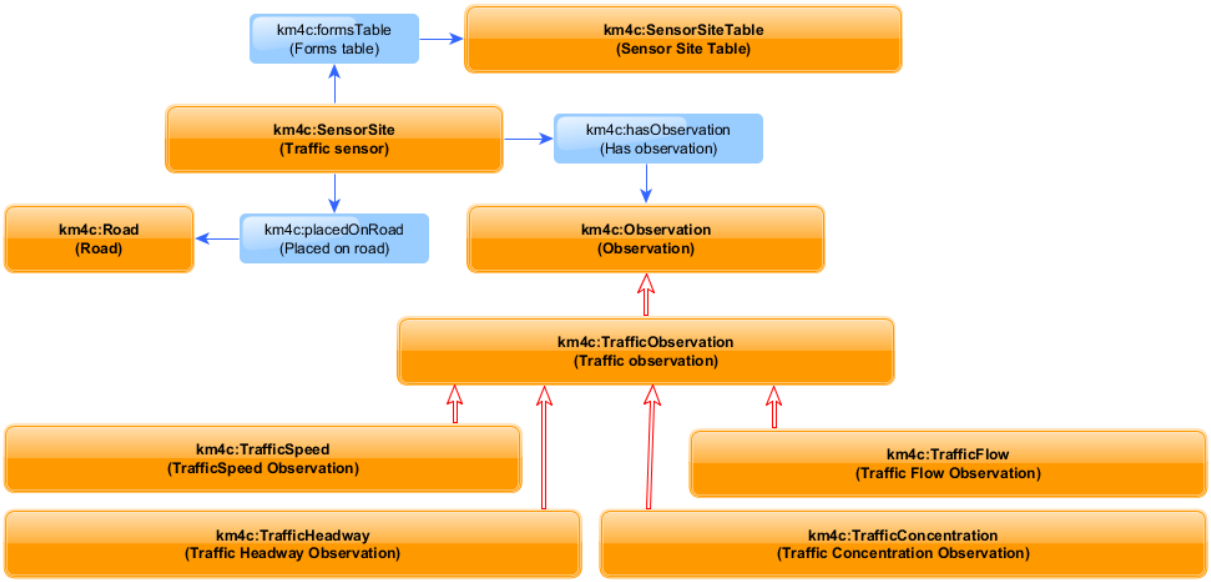


Figure 9 Traffic detection

Thanks to additional information received from the Osservatorio dei Trasporti, it has been possible to accurately geo locate all sensors. Sensors are divided into groups, each group is represented by an instance of the km4c:SensorSiteTable class and each instance of the class km4c:SensorSite (that represent a single sensor) is connected to its group through the object property km4c:formsTable and, as mentioned earlier, each instance of km4c:SensorSite class can be connected only to the km4c:Road class (through the object property km4c:installedOnRoad). Each sensor produces observations, which are represented by instances of the km4c:Observation class and these observations can belong to 4 types, i.e. they can be related to the average velocity (km4c:TrafficSpeed subclass), or related to the car flow passing in front of the sensor (km4c:TrafficFlow subclass), related to traffic concentration (km4c:TrafficConcentration subclass), and finally related to the traffic density (km4c:TrafficHeadway subclass). The classes km4c:Observation and km4c:SensorSite are connected through a pair of inverse object properties, the km4c:hasObservation and the km4c:measuredBySensor. Not only the class km4c:SensorSite has observations anyway. Since May 2018, the domain of the km4c:hasObservation property also encompasses weather sensors, represented by the km4c:Weather_sensor class (and the same can be said for the range of the inverse property).

A fourth remarkable subdomain in the *Sensors* macro-class addresses the Beacons, and their observations. An overview of this peculiar aspect is provided in Figure 10.

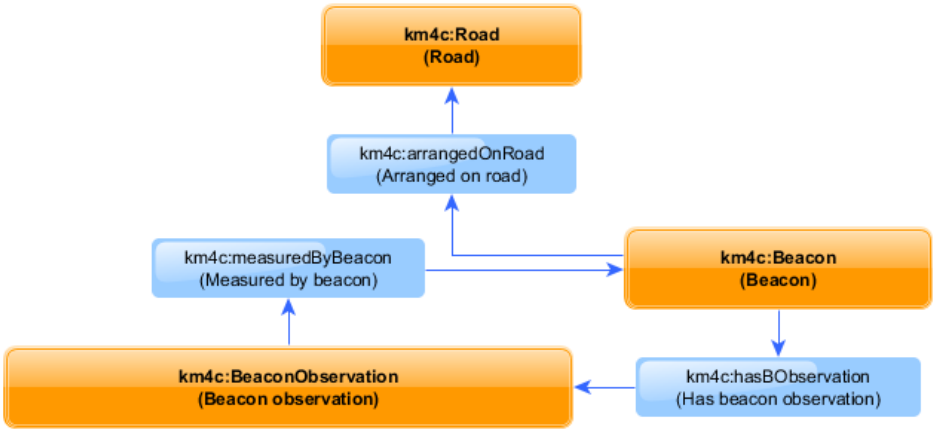


Figure 10 Beacons and their detections

The km4c:Beacon class represents a single device installed in the city, and the km4c:BObservation class, represents instead a single observation reported by each functional element; this two classes are connected each other via a pair of inverse object properties, the km4c:hasBObservation and the km4c:measuredByBeacon. Each beacon is uniquely identified by an identification code, and three other codes, the UUID (Universally Unique Identifier, that contains 32 hexadecimal digits, split into 5 groups, separated by dashes) which uniquely identifies the owner of the beacon (so if a store has 10 beacons, all beacons have the same UUID), major and minor that respectively identify instead a group of beacons and the number of each element within the group identified with the same major code. Geospatial coordinates, that identify the location where the beacons are located, are also available. The information that a beacon is capable to return are the signal strength, the coordinates where it contacts a user, and the date and time of the contact.

Last but not least, the Local Public Transport real-time data subsection, that you can find depicted in Figure 11. The mobility is the main challenge of all urban environments. As for the public transport, the Km4City Ontology includes both concepts for representing the structural elements, such as the lines, the routes, the

bus stops, and concepts for modelling the real-time data related to the public transport such as the AVM records (the real-time position of the buses) and the predictions about when the bus of a given line will reach a given bus stop.

The AVM systems are installed on most of ATAF busses. The km4c:AVMRecord class represents a record sent by the AVM system, that includes information on the last stop done (represented by the km4c:lastStop object property that connects the class km4c:AVMrecord to the class km4c:BusStop), GPS coordinates of the vehicle position, and the identifiers of vehicle and line. The km4c:BusStopForecast class includes instead a variable-length list of the upcoming stops with the planned passage time. This latter class is linked to the class km4c:BusStop through the km4c:atBusStop object property so that it could be possible to recover the list of possible lines provided on a certain stop (the km4c:AVMRecord class is in fact also connected to the km4c:Line class via the km4c:concernLine object property).

Finally, the Temporal macro-class, that you can find depicted in Figure 12, is based on the Time Ontology (http://www.w3.org/TR/owl-time/) but also on experience gained in other projects such as OSIM. It requires the integration of the concept of time as it will be of paramount importance to be able to calculate differences between time instants, and the Time Ontology comes to help us in this task.

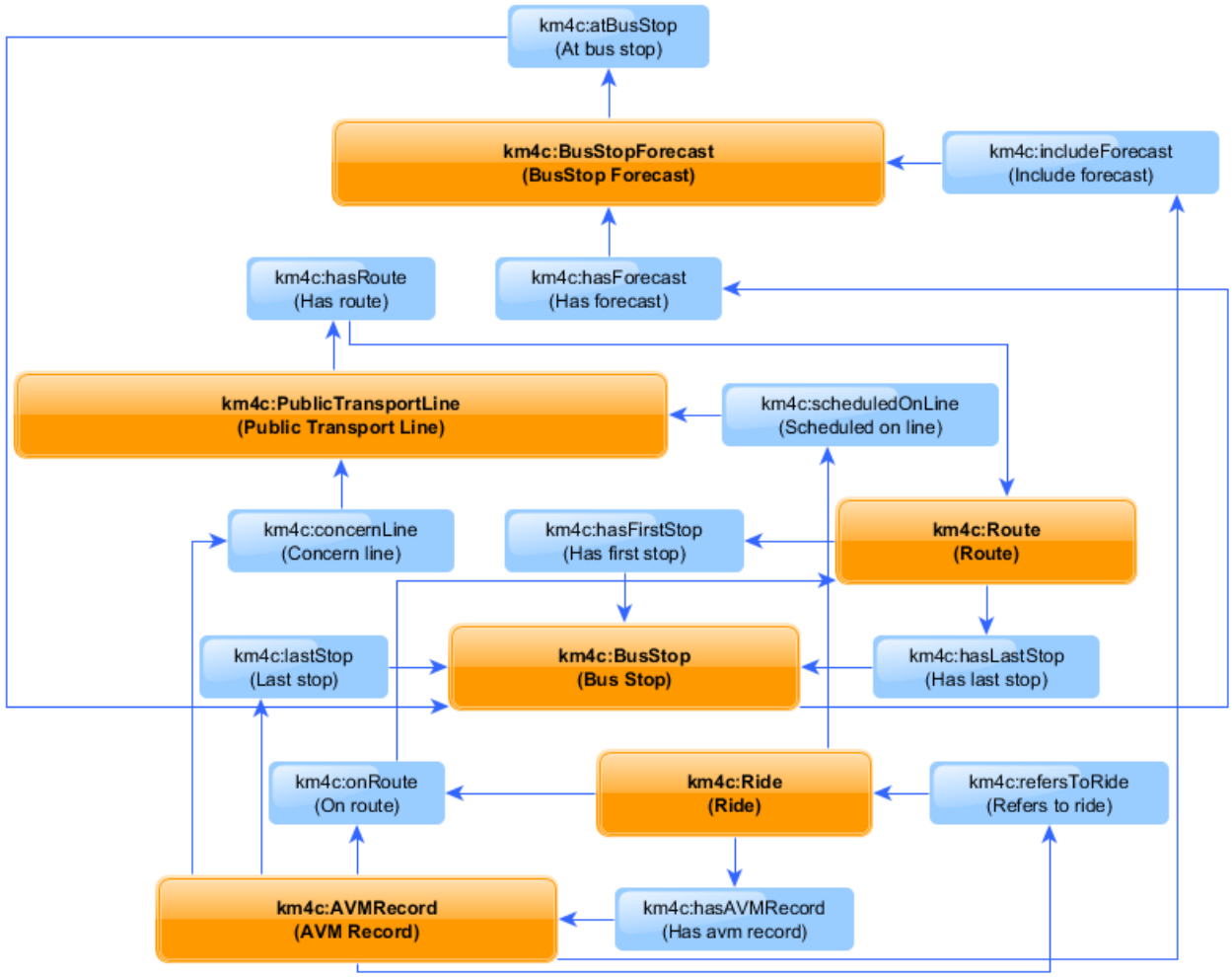


Figure 11 Real-time Public Transport

We define fictitious URI #instantForecast, #instantAVM, #instantParking, #instantWreport, #instantObserv to lately associate them to the identifier URI of a resource referred to the time parameter, i.e. respectively

km4c:BusStopForecast, km4c:AVMRecord, km4c:SituationRecord, km4c:WheatherReport and finally km4c:Observation.

The fictitious URI #instantXXXX, will be formed as a concatenation of two strings: for example, in the case of km4c:BusStopForecast instances, the stop code string (that uniquely identifies the stop) is concatenated to the time instant in the most appropriate format. It is necessary to create a fictitious URI that links a time instant to each resource to not to create ambiguity, because identical time instants associated with different resources may be present (although the format in which a time instant is expressed has a fine scale).

Time Ontology is used to define precise moments as temporal information, and to use them as extreme for intervals and durations definition, a feature very useful to increase expressiveness.

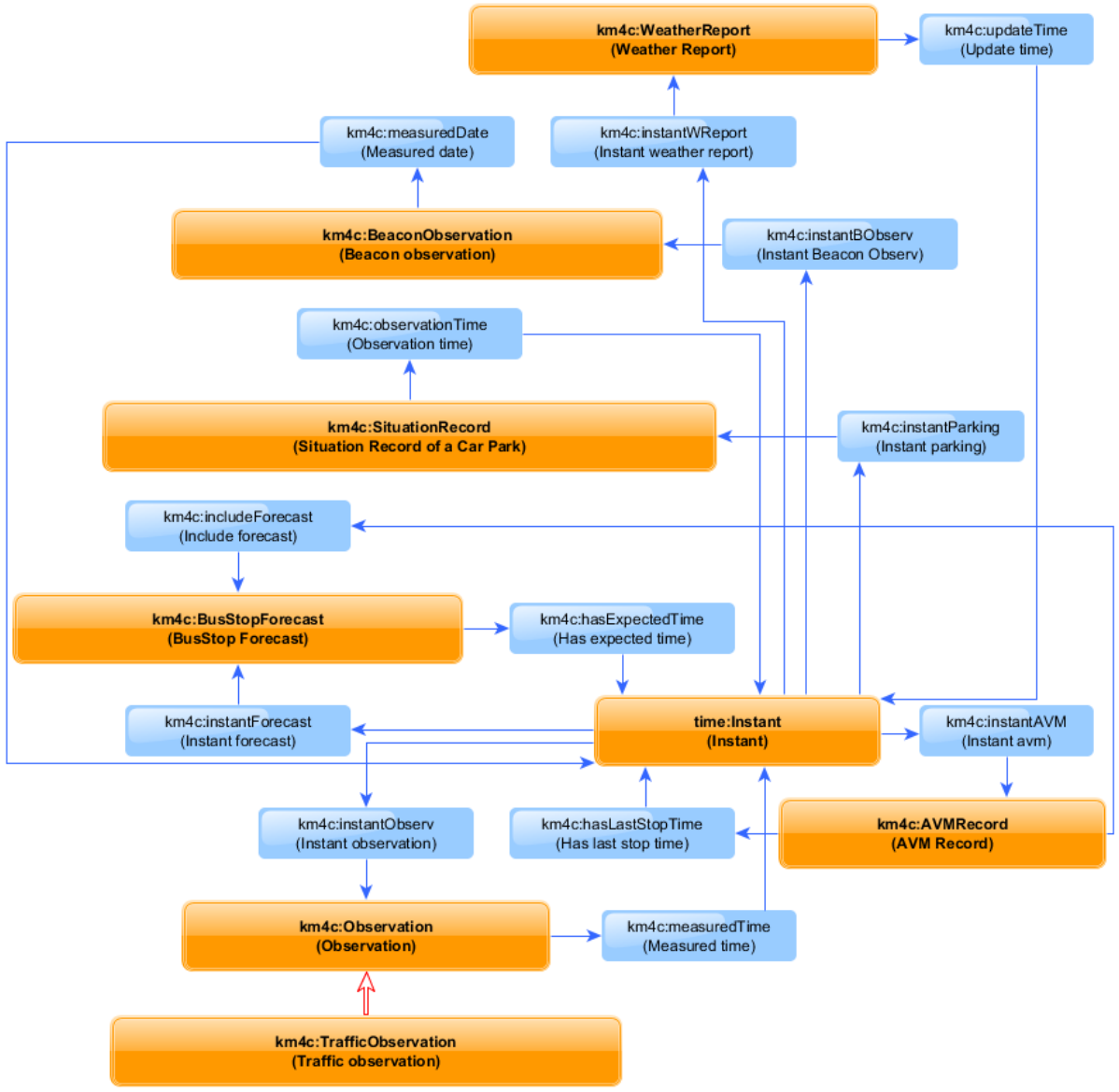


Figure 12 The Temporal domain

Pairs of object properties have also been defined for each class that needs to be connected to the class time:Instant. Between the classes time:Instant and km4c:SituationRecord the inverse object properties km4c:instantParking and km4c:observationTime have been defined. Between the km4c:WeatherReport and the time:Instant classes, the object properties km4c:instantWReport and km4c:updateTime have been defined. Between the classes km4c:Observation and time:instant, the inverse object properties km4c:measuredTime and km4c:instantObserv have been defined. Between the km4c:BusStopForecast and the time:Instant classes, the km4c:hasExpectedTime and the km4c:instantForecast object properties can be found. Between the km4c:AVMRecord class and the time:Instant class, there are the inverse object properties km4c:hasLastStopTime and km4c:instantAVM. Finally, between km4c:BeaconObservation and time:Instant the inverse properties km4c:instantBObserv and km4c:measuredDate can be found.

The domain of all object properties with instantXXXX name is defined by elements Time:temporalEntity, so that it could be possible to expand the defined properties not only to time instant, but also to time intervals.

The seventh macro-class (Figure 13), relates to the metadata associated with each dataset. Sesame [http://rdf4j.org/] allows to define, in the ontology, the Named Graphs, i.e. the graphs to which is associated a name, also called the context. The context is in practice an additional field that allows to expand the triple model into a quadruple model. Owlim [http://www.ontotext.com/products/ontotext-graphdb-owlim-new-2/], during the triple loading phase, allows to associate different contexts to different sets of triples. In this macroclass have been then defined all the data properties that allow to store relevant information related to a certain dataset, such as the date of creation, data source, original file format, description of the dataset, license associated with the dataset, type of ingestion process, how much the entire ingestion process is automated, type of access permitted to the dataset, overtime, period, associated parameters, update date, triples creation date.

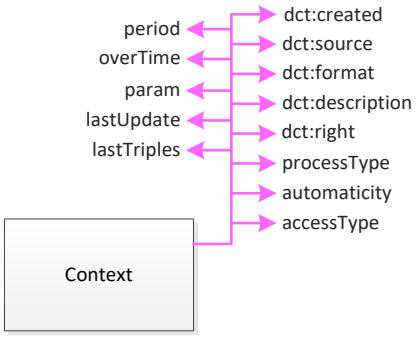


Figure 13 Metadata

Internet-of-Things peculiar concepts, depicted in Figure 14, have been included in the Km4City Ontology since version 1.6.5. Sensors and actuators are modelled through the concepts km4c:IoTSensor and km4c:IoTActuator respectively. Both sensors and actuators interface with the external through dedicated hardware/software components, the so-called brokers. A generic broker is modelled through the concept km4c:IoTBroker. Each broker implements one or more communication protocols, such as the NGSI, the MQTT, the AMQP, and other. Specialized concepts have been introduced for grouping those km4c:Brokers that communicate through the same protocol, such as the km4c:NGSIBroker, the km4c:MQTTBroker, the km4c:AMQPBroker, and other. The measurements that sensors are capable to perform, and the input signals that actuators accept, are modelled through the concept km4c:DeviceAttribute. For providing the km4c:DeviceAttribute instances of a semantic, the km4c:value_type property has been introduced. This

way, all sensors that measure a temperature will be linked to a km4c:DeviceAttribute instance whose km4c:value_type property is filled by a specialization of the ssn:Property concept that generically represents the *temperature*, while all traffic sensors that measure a vehicle flow will be linked to a km4c:DeviceAttribute instance whose km4c:value_type property will be filled with a specialization of the ssn:Property that generically represents a *traffic flow*. The Semantic Sensor Network Ontology, and the IoT-Lite Ontology are reused for a better interoperability with other ontologies.

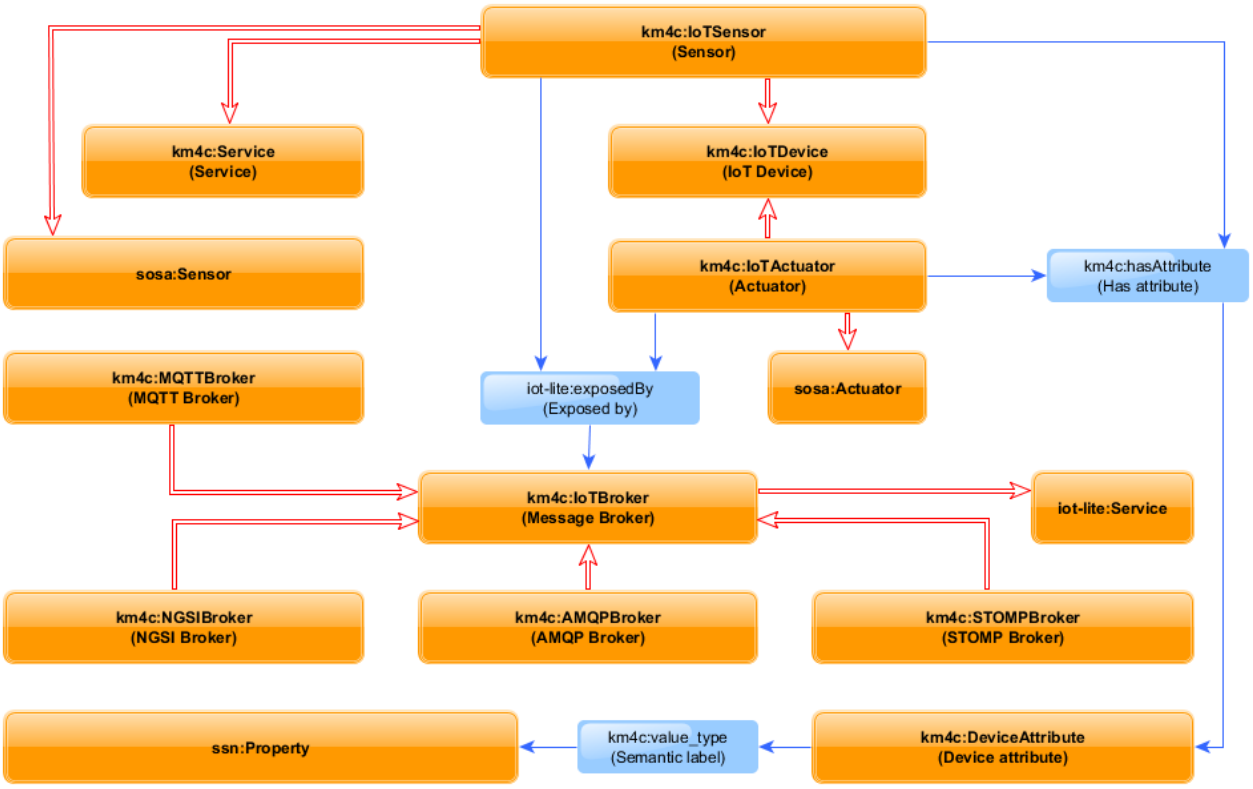


Figure 14 Internet-of-Things

Since August 2020, the Km4City/Snap4City Ontology also encompasses concepts and properties that are suitable for modelling the flowing of water and other substances in pipes deployed in the context of production lines of industries. This way, we have started moving in the direction of making the data model suitable for Smart Industry projects. In some more detail, a new vocabulary has been imported, namely the *SAREF extension for building devices* available at https://saref.etsi.org/saref4bldg/, and a set of concepts and properties have been added to the Km4City/Snap4City Ontology building upon that SAREF ontology. In some more detail, concepts have been added that enable the representation of the physical structuring of the industry (such as http://www.disit.org/saref4bldg-ext/Site to model portions of production buildings, and http://www.disit.org/saref4bldg-ext/Area to model portions of sites), but also of the logical structuring of the industry (such as the http://www.disit.org/saref4bldg-ext/ProductionLine that is made up of one or more http://www.disit.org/saref4bldg-ext/Flow that can be of http://www.disit.org/saref4bldg-ext/Water, http://www.disit.org/saref4bldg-ext/Energy, or http://www.disit.org/saref4bldg-ext/ChemicalCompound, and that http://www.disit.org/saref4bldg-ext/startIn and http://www.disit.org/saref4bldg-ext/endIn some https://saref.etsi.org/saref4bldg/PhysicalObject such as a https://saref.etsi.org/saref4bldg/Pump, for example the http://www.disit.org/saref4bldg-ext/pump1). See also Fig. 15 for an overview of the above.

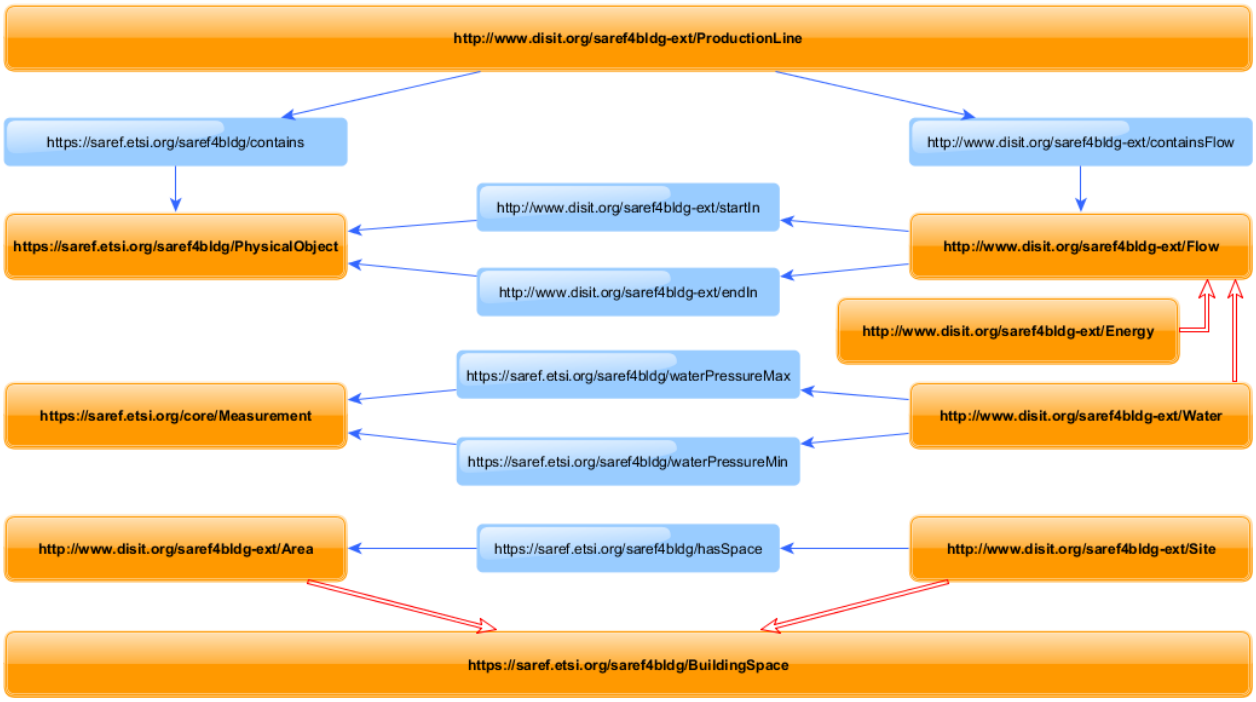


Figure 15 Smart Industry

Data properties of main classes

Data properties from included ontologies have been reused when possible. When it has not been possible to identify a suitable property to reuse from the included ontologies, a new ad-hoc property has been defined in the Km4City Ontology. We analyze below, partitioned by domain class, the data properties that have been defined within the Km4City Ontology.

The data property km4c:typeLabel is defined for all existing classes in the ontology, and it is the field where it is saved the type of the instance; thanks to this property, it is possible to perform a full-text search that also includes the types of instance.

Within the class km4c:PA have been defined only three data properties:

- foaf:name, that represents the name of the Public Administration represented by the considered instance, and its unique identifier present in the regional system;
- dct:identifier, from the DublinCore Ontology;
- dct:alternative, where the municipal code present in the tax code can be found.

More information about a PA can be found navigating the link to its corresponding Service class. The km4c:Resolution class, whose instances as seen above are the municipality resolutions, has the following properties:

- dct:identifier, a unique identifier;
- km4c:year, the year of approval;
- dct:subject, the resolution subject, from the DublinCore Ontology;

- dct:created, the resolution date, from the DublinCore Ontology.

Each instance of the km4c:Road class is uniquely identified using the data property dct:identifier where it is stored the toponym identifier for the entire regional network, consisting of 15 characters and defined according to the following rule: RT followed by ISTAT code of the municipality to which the toponym belongs (6 characters), followed by 5 characters representing the sequential from the character value of the ISTAT code, and finally the letters TO. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance. The km4c:roadType instead, represents the type of toponym. Notable types of toponym follow:

- Locality;
- Square;
- Plaza;
- Road;
- Boulevard;
- Alley;
- Lane.

Inside the street graph there are also two names for each toponym: the name and the extended name, which also includes the toponym’s type. The name without type is a string, and it can be found in the data property km4c:roadName, while the long name can be found in the data property km4c:extendexName. For representing the alternative/abbreviated names, such as “Via S. Marta” for “Via Santa Marta”, the dct:alternative data property is reused. Many alternatives can be found set for a single toponym.

Concerning the km4c:AdministrativeRoad class, in addition to the data properties dct:alternative and km4c:adRoadName, that respectively contain the possible alternative names of the road and its official name.
The data property km4c:adminClass has been defined to represent the administrative classification, that is if a road is:

- Highway;
- regional road;
- road;
- municipal road;
- military road;
- driveway.

Finally, the property dct:identifier is filled by an identifier which complies with the following rule defined at the regional level: 15 characters, starting with the letters RT followed by ISTAT code of the municipality that owns the administrative road (6 characters), followed by 5 characters representing the sequential number of the road within the group of the roads that have the same ISTAT code, and finally the letters PA. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier

of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance.

The instances of km4c:RoadElement are uniquely identified through the data property dct:identifier, composed as follows: RT characters followed by 6 characters for the ISTAT code of the belonging municipality, followed by 5 characters that represent the progressive from the ISTAT code, and finally the ES characters. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance.

The km4c:elementType can also be found defined for the class km4c:RoadElement, and it can take the following values:

- roadway trunk;
- structured traffic area;
- toll booth;
- level crossing;
- square;
- roundabout;
- crossing;
- structured car park;
- unstructured traffic area;
- car park;
- competence area;
- pedestrian;
- connection, motorway link road, interchange;
- controviale;
- ferry boat (dummy element);

In the street graph dataset provided by the Tuscany Region, the functional classification of the road elements can also be found, and it is modelled in the Km4City Ontology through the km4c:elementClass data property, whose possible values are:

- highway;
- main suburban;
- secondary suburban;
- thoroughfare;
- district urban;
- local/to private use.

The km4c:composition data property instead has been defined to indicate the composition of the road to which the road element belongs to and the values that it can assume are "single track" or "separate roadways".

The property km4c:elemLocation represents the location of the element, and it can take the following values:

- street level;
- bridge;
- ramp;
- tunnel;
- bridge and tunnel;
- bridge and ramp;
- tunnel and ramp;
- tunnel, bridge and ramp.

For representing the width of the road elements, the data property km4c:width has been defined, whose possible values are:

- less than 3.5m;
- between 3.5 and 7.0m;
- greater than 7.0 meters;
- not detected.

For the length, the data property is km4c:length has been defined, where an unconstrained value that represents the length of the segment in meters can be found.

Other data available on the streets graph, essential for defining the maneuvers permitted, is the travel direction of the road element, indicated by the km4c:trafficDir data property, which may take one of the following four values:

- road section opened in both directions (default)
- road section opened in the positive direction (from initial node to final node)
- road section closed in both directions
- road section opened in the negative direction (from final node to initial node)

The km4c:operatingStatus data property serves instead to track the operating status of the different road elements and can take the following values:

- in use;
- under construction;
- abandoned.

Finally, there is also a data property that takes into account the speed limits on each road element, namely the km4c:speedLimit.

In the class km4c:StatisticalData data properties can be found that allow to associate the actual value (stored in km4c:value), with semantic information and other meta data. Examples are the data properties:

- dct:identifier;
- dct:description;
- dct:created;
- dct:subject.

A node or junction is a point of intersection of the axes of two road elements, and it always is a punctual entity, represented in geometric terms by a pair of geospatial coordinates. Instances of the km4c:Node class can be uniquely identified thanks to the data property dct:identifier, composed of of 15 characters, according to the following rule: the first two characters are RT, followed by the 6-character ISTAT code of municipality where is located the node, followed by 5 characters of progressive starting from the value of ISTAT code, and finally GZ characters. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance.

Each node is also characterized by a type, represented by the data property km4c:nodeType, whose allowed values are:

- street level intersection/ fork;
- toll booth;
- mini roundabout (radius of curvature< 10m);
- change seat;
- end (beginning or end RoadElement);
- change place name / ownership / manager;
- change width class;
- unstructured traffic area;
- level crossing;
- support node (to define loop);
- change in technical/functional classification;
- change in operating status;
- change in composition;
- intermodal hub for rail;
- intermodal hub for airport;
- intermodal hub for port;
- region boundary;
- dummy node.

Even in this case, it is useful the insertion of the data properties for the localization, namely the geo:lat and the geo:long.

The access rules are described by instances of the km4c:EntryRule class, uniquely identifiable through a dct:identifier of 15 characters thus formed: the RT characters followed by 6 characters representing the ISTAT code of the municipality, 5 other characters that represent the progressive starting from that ISTAT code, and finally the characters PL. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance.

The access rules are then characterized by a type, represented by the data property km4c:restrictionType, whose allowed values are:

- Blank (only in case of maneuver);
- Traffic flow direction;
- Blocked way;
- Special restrictions;
- Under construction;
- Information about tolls;
- Fork;
- Forbidden manoeuvres;
- Vehicles restrictions.

In addition to the type, the access rules have also a description, also called restriction value and represented by the data property km4c:restrictionValue, which can assume different ranges of values, depending on the type of restriction concerned:

- Blank possible values:
    - Default Value = “-1”;
- Possible values for Traffic flow direction & Vehicles restrictions:
    - Closed in the positive direction;
    - Closed in the negative direction;
    - Closed in both directions;
- Blocked way possible values:
    - Accessible only for emergency vehicles;
    - Accessible via key;
    - Accessible via guardian;
- Special restrictions possible values:
    - No restrictions (Default);
    - Generic Restriction;
    - Residents only;
    - Employees only;
    - Authorized personnel only;
    - Staff only;
- Under construction possible values:
    - Under construction in both directions;
    - Under construction in the travel direction of the lane;
    - Under construction in the travel opposite direction of the lane;
- Information about tolls possible values:
    - Toll road in both directions;
    - Toll road in the negative direction;
    - Toll road in the positive direction;
- Fork possible values:
    - multi lane bifurcation;
    - simple bifurcation;
    - exit bifurcation;
- Forbidden manoeuvres possible values:
    - prohibited maneuver;
    - turn implicit.

The class maneuver is substantially different from the classes seen so far: each maneuver is indeed uniquely identified by an ID consisting of 17 digits. A maneuver is described by the sequence of the road elements that affects, ranging from 2 to 3, and from the node of interest, then it will be almost completely described through object properties representing this situation.

Within the Tuscany Streets Graph, we find data related to the operation type, bifurcation type and maneuver type prohibited, but since the last two types are almost always "undefined", it has been associated with a data property the type of maneuver only, namely the km4c:maneuverType, which can take the following values:

- fork;
- manovra proibita calcolata (Calculated Maneuver);
- manovra obbligatoria (Mandatory Maneuver);
- manovra proibita (Prohibited Maneuver);
- manovra prioritaria (Priority Maneuver).

The km4c:StreetNumber class also has a dct:identifier property to uniquely identify the instances of this class, composed as follows: the RT characters, followed by 6 characters for the ISTAT code of the municipality, 5 other characters for the progressive from the ISTAT code and finally the characters CV. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance.

In Florence there are two independent numberings, the black, and the red, so it was inserted the km4c:classCode data property, which takes into account the color of the civic and can take the following values: red, black, no color. No colors is intended to be used in those cities where just one numbering system exist. A number can also be formed, besides the numerical part always present, by a literal part, represented respectively by the km4c:number and the km4c:exponent data properties. It was also inserted an additional property, km4c:extendNumber, in which it will be stored the number together with the exponent in order to ensure greater compatibility with the different formats in which could be written/researched the instances of this class.

The km4c:Milestone class, as seen above, identifies the value of the mileage progressively, with respect to its starting point. Instances of this class have a unique identifier made up of 15 characters, represented by dct:identifier: the characters RT, followed by 6 characters for the ISTAT code of the municipality, other 5 characters for the progressive from ISTAT code, and finally, the CC characters. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance.

At each milestone, it is usually written the mileage, which corresponds to that point and the value of this writing is stored through the data property km4c:text. Thanks to the information contained in the street graph, it is trivial to retrieve the name of the street, highway, etc. where the milestone is located. It could

therefore be furtherly defined an object property linking the km4c:Milestone class to the km4c:Road class. At today, we have avoided doing so, because this information is still recoverable with an extra step through the km4c:RoadElement class. Anyway, it can be easily inserted if deemed appropriate in the future. Also in this case the data properties for localization, namely the geo:lat and the geo:long, are defined.

The km4c:Access class models the punctual item that identifies on the territory, directly or indirectly, the external access to a specific place of residence/business. The access is materialized in the practice as the *plate* of the street number. As previously mentioned, each access is logically connected to at least one number. Each instance of the km4c:Entry class is uniquely identified by the data property dct:identifier, consisting of 15 characters, and composed as follows: RT, followed by 6 characters of municipality ISTAT code, then another 5 character of the progressive from ISTAT code, and finally the AC characters. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance.

There are only three types of accesses, and this value is stored into the km4c:entryType data property:

- direct external access;
- indirect external access;
- internal access.

Also, it could be useful to know if the access can be traversed with motor vehicles or not. A dedicated data property, namely the km4c:porte-cochere, has been defined for such a purpose.

In this case also, the data properties for localization, namely the geo:lat and the geo:long, are present.

Coming to the most recent improvements in the street graph modelling, and more specifically to the modelling of the lanes, the only data property of the class km4c:Lanes is the km4c:direction, that can be found only in those cases when two traffic directions can be found in the road or segment of road, for identifying the traffic direction in which the lanes are traversed. Indeed, if a road has two traffic directions, it is connected to two instances of km4c:Lanes, one for each of the two traffic directions. Instead, the instances of the concept km4c:LanesCount usually have a data property named km4c:undesignated that is filled by an integer number that indicates how many of the lanes are not reserved to specific typologies of vehicle, and also have some other data properties such as km4c:bus, km4c:motor_vehicle, km4c:psv that borrow their names from typologies of vehicles that are defined in the Open Street Map, and that are filled by integer numbers that indicate how many of the lanes are reserved to the specific typology of vehicles. It should be remembered that each km4c:LanesCount instance is connected to a km4c:Lanes instance that represent a set of lanes. Finally, the concept km4c:Lane has a data property km4c:position that indicates the position of the lane in the set of lanes of which it is part. The km4c:Lane instances can also have a km4c:turn property through which it is indicated the mandatory maneuver that drivers must perform once they reach the end of the lane.

Instead, as for the traffic restrictions, the km4c:TurnRestriction concept has two data properties, namely the km4c:restriction that indicates whether the maneuver is mandatory or forbidden, and the km4c:except that can be only found in those cases when the restriction does not apply to one or more categories of vehicle. The km4c:AccessRestriction has instead four data properties:

- km4c:who, where the categories of vehicle that are concerned by the restriction can be found;
- km4c:direction, where the traffic direction to which the restriction applies is indicated;
- km4c:access, where it is indicated if the specified vehicles have a reserved access, or if they cannot access, the road (or element) that fills the km4c:where property;
- km4c:condition, that can be only found in those case when the restriction applies under a set of conditions other than the typology of vehicles and traffic direction. In brackets, this data property can be found set for all the km4c:Restriction instances, if some condition other than those that are modelled through dedicated properties must be represented.

Finally, the km4c:MaxMinRestriction has the following data properties:

- km4c:what, that indicates what the restriction refers to;
- km4c:limit, that indicates the limitation itself;
- km4c:direction, that indicates the traffic direction in those cases when only one of the two traffic direction is affected by the restriction.

The km4c:Service class has been equipped with the contact card of the schema.org ontology (https://schema.org) to make the description of the various companies more standardized:

- schema:name;
- schema:telephone;
- schema:email;
- schema:faxNumber;
- schema:url;
- schema:streetAddress;
- schema:addressLocality;
- schema:postalCode;
- schema:addressRegion to which we added skos:note for any additions such as the opening hours of an activity sometimes present in the data.

In addition, other data properties have been defined:

- km4c:houseNumber to isolate the street number from the address;
- the data properties geo:lat and geo:long for localization.

With the introduction of the DigitalLocation, the km4c:Service class has been enriched with the following properties:

- km4c:hasGeometry, the property defined to store the set of coordinates associated with the digital location, which can be a POINT, a LINESTRING and a POLYGON;
- km4c:routeLength, the property where the length of the paths is stored (e.g. for jogging services);
- km4c:stopNumber, the property that indicates the number of the stop, that is local to the line;
- km4c:lineNumber, the property that indicates the line to which a stop belongs;
- km4c:managingBy, the property indicating the manager of the DigitalLocation;
- dct:description, the property where a description of the DigitalLocation is stored;
- km4c:owner, the property that indicates the owner of the DigitalLocation;
- km4c:abbreviation, the property showing the abbreviation of the DigitalLocation name;

- km4c:type, the property indicating the category of the DigitalLocation manager, or the LTZ type or the museum type or the reference area;
- km4c:time, the property that indicates the opening hours of a DigitalLocation;
- km4c:firenzeCard, the property that indicates if the DigitalLocation is affiliated with the FirenzeCard program;
- km4c:multimediaResouce, the property to associate media files to a DigitalLocation;
- km4c:districtCode, the property that indicates the district where the DigitalLocation is located;
- km4c:routePosition, the property that indicates the DigitalLocation position within a thematic route;
- km4c:areacode, the property indicating a municipal code for the referenced area;
- km4c:routeCode, the property that specifies the number of thematic route to which the DigitalLocation refers;
- scheme:price, the property that indicates the possible cost of entry.

The class km4c:Event borrows its properties from the schema:Event class. Nevertheless, additional properties have been added based on information provided by the Municipality of Florence. The resulting full list of the km4c:Event data properties is proposed below here:

- schema:startDate, the property that indicates the event start date;
- schema:endDate, the property that indicates the event end date;
- schema:description, the property that contains the event description;
- schema:image, the property that contains, if any, the URL of the most representative image of the event;
- schema:name, the property that contains the name of the event;
- schema:url, the property that contains the website address of the event;
- schema:telephone, the property that contains the reference telephone number for the event;
- schema:streetAddress, the property that contains the address where the event will be held;
- schema:addressLocality, the property that contains the city where the event will be held;
- schema:addressregion, the property that contains the province where the event will be held;
- schema:postalCode: the property that contains the zip code of the location where the event will be held;
- km4c:houseNumber, the property that contains the civic number of the place where the event will be held;
- skos:note, the property that contains generic notes and information about the event;
- scheme:price, the property that indicates the possible cost of entry;
- dct:identifier, the property that contains the identifier assigned to the event by the municipality;
- km4c:eventCategory, the property that contains the category of the event;
- km4c:placeName, the property that contains the name of the place where the event will take place;
- km4c:freeEvent, the property that indicates if the event is free or not;
- km4c:eventTime, the property that contains the event start time.

In addition to these properties, each event is also provided of a pair of geospatial coordinates.

The gis:Geometry class, instead, presents only the data property gis:asWKT, which allows to define a unique string representing the set of coordinates that define the geometric shape of the object, for example:

LINESTRING(11.21503989 43.77879298, 11.21403307 43.77936126, 11.21385829 43.77947115)

The class km4c:CarParkSensor has other properties specific for car parks:

- the dct:identifier, always defined at the regional level through a 15 characters code beginning with RT and ending with the initials of the belonging province. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance;
- km4c:capacity, i.e. the number of parking places;
- km4c:fillrate and km4c:exitrate, respectively the number of vehicles entering/leaving;
- km4c:carParkStatus, i.e. a string that describes the current state of the car park (possible values are "enoughSpacesAvailable", "carParkFull" "noParkingInformationAvailable", etc.);
- the validity status of the record (the km4c:validityStatus data property, which can only be "active" for parking);
- the km4c:parkOccupancy, i.e. the number of occupied space, or the corresponding percentage that is instead called km4c:occupied;
- the free places, for which the data property km4c:free has been defined.

As for the km4c:SituationRecord class, some data properties have been defined:

- a unique id, stored in the dct:identifier data property;
- the record validity status (data property km4c:validityStatus, which can only be "active" in the case of parking lots);
- the km4c:parkOccupancy, i.e. the number of parking places occupied, or the corresponding percentage that is called instead km4c:occupied;
- the vacancy parking places for which there is the data property km4c:free.

The km4c:WeatherReport class is characterized by:

- dct:identifier, containing the unique id which identifies the different reports;
- km4c:timestamp, that indicates the time when the report was created in milliseconds

Other properties have been added to express:

- the phase of the moon (km4c:lunarphase);
- the time of sunset/sunrise (that is km4c:sunrise and km4c:sunset);
- the time of moonrise and moonset (km4c:moonrise and km4c:moonset properties).

There are also the km4c:heightHour and km4c:sunHeight data properties, which represent the time when the sun reaches its maximum height, the height.

Each instance of km4c:WeatherPrediction is characterized by data properties such as:

- the km4c:day, which is the day that is referenced in prediction (together with the id of the report, it uniquely identifies a forecast);
- the minimum and maximum temperature values, and the real and perceived temperature values (respectively represented by the data properties km4c:minTemp, km4c:maxTemp, km4c:recTemp, km4c:perTemp);
- wind direction (km4c:wind);
- km4c:humidity, that is the percentage of humidity;
- the level at which there is snow (km4c:snow);
- km4c:hour, representing the part of the day that is referenced by each individual forecast contained in a report;
- the UV index of the day, represented by km4c:uv.

The km4c:SensorSiteTable and km4c:SensorSite classes have only one data property, the dct:identifier.

The km4c:Observation class is instead completed by the data properties:

- dct:identifier;
- dct:date;
- km4c:averageDistance and km4c:averageTime, representing distance and average time between two cars);
- km4c:occupancy and km4c:concentration, relative to the percentage of occupation of the road referred to the number of cars and the car concentration;
- km4c:vehicleFlow, flow of vehicles detected by the sensors;
- data related to the average velocity and the calculated speed percentile, contained in the km4c:averageSpeed, km4c:thresholdPerc and km4c:speedPercentile properties.

The km4c:PublicTransportLine class and km4c:Lot class have both the following data properties:

- dct:identifier, the number of the line/lot, from the DublinCore Ontology;
- dct:description, the description of the path/lot, from the DublinCore Ontology.

The km4c:Route class, in addition to the dct:identifier, foaf:name and dct:description, presents:

- the km4c:routeLenght property, that is the path length in meters;
- the km4c:direction property, the path direction.

The km4c:BusStop class has:

- dct:identifier;
- foaf:name, for the name of the bus stop;
- geo:lat and geo:long belonging to Geo Ontology.

The km4c:RouteJunction class has:

- a dct:identifier;
- a pair of geospatial coordinates.

The class km4c:BusStopForecast contains only data properties for the time of arrival and the identification, respectively named km4c:expectedTime and dct:identifier.

The km4c:AVMRecord class requires instead data properties to identify:

- the means to which the record refers (km4c:vehicle);
- the arrival time to last stop (km4c:lastStopTime);
- the ride state (i.e. whether it is early, late or in time) stored in the km4c:rideStatus property;
- the managing company and the company that own the AVM system (km4c:managingBy and km4c:owner properties);
- the unique identifier of the record (dct:identifier);
- the coordinates geo:lat and geo:long, which indicate the exact vehicle position at the report time.

Finally, the class km4c:Ride has only the dct:identifier data property, like the km4c:RouteLink class. The km4c:RouteSection class, in addition to the identifier, has the data property km4c:distance, where it is saved the distance between two successive stops within a route.

We now analyze the data properties for the classes km4c:Beacon and km4c:BeaconObservation.

The first class, in addition to the data property dct:identifier, has:

- the property km4c:owner, which stores the owner' name of each beacon;
- schema:name, that contains the name associated with a beacon;
- km4c:uuid;
- km4c:minor;
- km4c:major;
- km4c:public, that defines if the beacon is public or not;
- the location, i.e. geo:lat and geo:long.

The second class, km4c:BeaconObservation, has instead, in addition to the dct:identifier that makes unique each reading, the following properties:

- the coordinates where the connection has made, stored in geo:lat and geo:long data properties;
- the dct:date, that stores date and time of observation;
- the data property km4c:power, indicating the power with which a beacon can connect to a user, which also allows to precisely determine the distance.

Properties were also defined to be associated to the Context, regarding information coming from the tables that describe the individual ETL processes which process different public/private data. For each process, in fact, we define:

- a source type stored within the property dct:source;
- the date of ingestion into the ontology, namely the property dct:created;
- the original data format (CSV, DBF, etc.) stored in the property dct:format;
- a brief description of the dataset in dct:description;
- the dataset license bound, saved into the data property dct:right;
- the type of process to which it refers, stored in the km4c:ProcessType data property;
- the data property km4c:automaticity, that says if the process is completely automated or not, for example, the street graph datasets can not be fully automated because the process to obtaining data needs a person that send and receive the e-mail messages that carry the dataset requests and responses;

- the data property km4c:accessType, that indicates how the data are recovered (HTTP calls, Rest, and so on);
- km4c:period, that contains the time (in seconds) between two calls of the same process;
- km4c:overtime, that indicates the time after which a process must be killed;
- the data property km4c:param that contains the resource link, if it is an open dataset retrievable via http;
- km4c:lastUpdate, that represents the date of the last data update;
- km4c:lastTriples, that represents the date of the last triple generation.

The class km4c:RailwayLine has only three data properties:

- dct:identifier, that contains the unique identifier of the railway line (a 12 characters code starting with the letters RT, followed by 3 characters to identify the region - T09 for Tuscany – and by 5 characters of sequential number and finally the letters PF). It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance;
- the foaf:name, in which the convention naming is saved;
- the dct:alternative, where it is saved instead the official name of the Railway Line.

The km4c:RailwayDirection class has only the first two data properties specified for km4c:RailwayLine, with the same use:

- dct:identifier, where it is stored in the code, that consists of 12 characters starting with the letters RT, followed by 3 characters that identify the region - T09 for Tuscany - 5 characters to the sequential number and finally the letters ED. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance;
- foaf:name, where is stored in the convention naming.

The class km4c:RailwayElement has the usual property dct:identifier, consisting of 12 characters that follow the following rule: RT characters followed by 3 characters of region code (T09 for Tuscany), followed by the 5 numbers of the sequential number, and finally the letters EF. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance.

In addition to this, the following have been defined:

- km4c:elementType, which can take the following three values "ordinary railroad", "railroad AC/AV" and "other";

- km4c:operatingStatus, that can take only the values "railway construction", "railroad in operation" and "disused railway";
- km4c:elemLocation, that indicates the rail element location, and that can take the values "grade-level", "on bridge/viaduct" and "in tunnel";
- km4c:supply, the data property that specifies whether this is an "electrified line" or a "non-electrified" line;
- km4c:gauge, a property that indicates if the gauge is "reduced" or "standard";
- km4c:underpass, that can take the following values:
  - the item is not in underpass of any other object;
  - the element is in underpass of another object;
  - the element is simultaneously in overpass and underpass of other objects.

For the modelling of underpasses, the Ontology also encompasses the km4c:Underpass concept.

Other data properties that have been defined for the km4c:RailwayElement class are:

- km4c:length, that is the item length in meters;
- km4c:numTrack, the number of tracks (0 if the line is under construction or abandoned);
- km4c:tracktype, that indicates if the element consists of "single track" or "double track".

The class km4c:RailwaySection requires the km4c:axialMass data property, i.e. the classification of the line with respect to the axial mass, which may take the following values:

- D4 - corresponding to a mass per axle equal to 22.5 t;
- C3 - corresponding to a mass per axle equal to 20.0 t;
- B2 - corresponding to a mass per axle equal to 18.0 t;
- A - corresponding to a mass per axle equal to 16.0 t;
- Undefined.

Other data properties that have been defined are:

- km4c:combinedTraffic, that can assume the values "PC80", "PC60", "PC50", "PC45", "PC32", "PC30", "PC25", "PC22", "lines with the loading gauge FS "and "undefined";
- dct:identifier, the usual 12 characters code that begins with RT characters, followed by the regional code and 5 number for the sequential number and that ends with a TR. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance;
- foaf:name, i.e. the naming convention of line.

For the km4c:RailwayJunction class, only three data properties have been defined:

- dct:identifier, that is the identification code of 12 characters formatted as in previous cases, but ending in GK. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has

originated the production of the instance, and a two-character label that denotes the type of the generated instance;

- foaf:name, i.e. the official name for junctions, stations and rail yard;
- km4c:juncType, which can take one of the following values:
  - rail crossing;
  - terminal (beginning or end);
  - junction (junction or branch);
  - station/stop/rail toll;
  - freight;
  - interporto;
  - change of state (COD_STA);
  - change of venue (COD_SED);
  - variation in the number of tracks (Num_bin);
  - power variation (COD_ALI);
  - administrative boundary.

The class km4c:TrainStation presents:

- the usual identifying code of 12 characters contained in the dct:identifier, that consists of RT followed by 3 char of regional identification - T09 for Tuscany - 5 char of progressive number and finally the characters SF. It should be noted here that for those instances that are generated through the triplification of the Open Street Map, instead of through the ETL ingestion of the data sets that come from the Tuscany Region, the identifier takes a different form. In more detail, it starts by “OS”, and it also contains the numeric identifier of the Open Street Map element that has originated the production of the instance, and a two-character label that denotes the type of the generated instance;
- foaf:name, in which the official name is stored;
- the address retrieved from the list posted on the RFI's website is stored in the fields schema:addressRegion, schema:addressLocality, schema:postalCode, and schema:streetAddress;
- the managing body found on RFI's website is stored into the data property km4c:managingAuth;
- the data property km4c:category contains the category to which the station belongs;
- the data property km4c:state contains the state of the station, which can take only the values "Active", "not Active" and "optional stops on demand".

The km4c:Goodyard class, in addition to the identifying code (formatted as all of the above but ending in SM) stored in dct:identifier, has:

- the data property foaf:name, in which the name of freight facility is saved;
- the km4c:railDepartment, where the name of the railway compartment can be found;
- the km4c:railwaySiding, the definition of the physical characteristic of the junction number;
- the km4c:yardType, that indicates whether the yards are public (value "public yard") or if the junctions are for private use (value "junction in line");
- the data property km4c:state, that indicates if the yard is "active" or "under construction".

Finally, coming to the Internet-of-Things, the main properties that can be found set on the instances of km4c:IoTSensor and km4c:IoTActuator that are peculiar of the Km4City Ontology are:

- km4c:format, the format (e.g. csv, xml, json) of the data that the device produces in output or accepts in input;
- km4c:macaddress, the MAC address of the device;
- km4c:model, the model of the device;
- km4c:ownership, whether the device is public or private;
- km4c:producer, the manufacturer that produces the device;
- km4c:protocol, the communication protocol supported by the device, e.g. MQTT, NGSI, and so on.

Other relevant properties that we found set on the km4c:IoTSensor instances but that are borrowed from other ontologies are used for representing the name, and the geospatial coordinates of the device.
The same properties can be found set for the km4c:Actuator instances. Instead, the properties defined for the brokers are mainly borrowed from other ontologies. The km4c:created property is indeed the only defined within the Km4City Ontology, and it indicates the date and time when the broker instance has been created in the KB. Other information that can be found about a broker are its endpoint, and its name. Finally, the main properties that can be found set on the km4c:DeviceAttribute instances are:

- km4c:value_name, a label for the specific typology of values produced by the specific device;
- km4c:data_type, that indicates if the values that are produced of the specific typology are integer numbers, floats, strings, or what else;
- km4c:value_unit, the unit of measure;
- km4c:order, it is useful in those cases when the device produces several values in output at each detection, for understanding which of those is described by the km4c:DeviceAttribute;
- km4c:disabled, a Boolean property that indicates whether the attribute should be kept into consideration (and therefore displayed to users, indexed, and so on) or if it has to be discarded;
- km4c:editable, a Boolean property that indicates if attribute values can be written, or if they can be only read;
- km4c:healthiness_criteria, that indicates how it should be evaluated the healthy status of the device. Possibilities are: (i) the device is unhealthy if it produces the same value in output for more that a given number of consecutive detections; (ii) values are not produced in output with the expected periodicity; (iii) values fall outside of a given range;
- km4c:different_values, that indicates, for those cases when the healthiness evaluation is based on criteria (i), the threshold number of detections;
- km4c:value_bounds, that indicates, for those cases when the healthiness evaluation is based on criteria (ii), the range of the acceptable output values;
- km4c:value_refresh_rate, that indicates the expected frequency of generation of the output values.