km4city - the Knowledge Model 4 the City

Authors: Pierfrancesco Bellini, Paolo Nesi, Nadia Rauch

referent coordinator: paolo.nesi@unifi.it

Knowledge base accessible as sparql entry point as shown from http://log.disit.org

OWL version accessible from: km4c 1-01.zip, http://www.disit.org/6506

http://www.disit.org/km4city/schema/

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

info from info@disit.org

version 2.1, draft

date 08-09-2014

SOME RIGHTS RESERVED

The ontology is available under Creative Commons Attribution-ShareAlike, 3.0 license.

Aim

The development of a unified and integrated ontology for smart city including transport, infomobility and large set of other open data. This work has been performed at DISIT lab for a number of smart city projects, see also for its use on:

- Linked Open Graph: http://log.disit.org
- Service Map: http://servicemap.disit.org
- Smart city ingestion process document and process in place: http://www.disit.org/6058

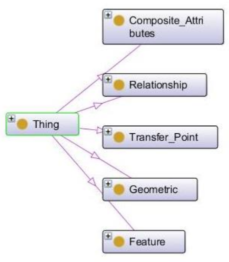
State of the Art

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 [http://researcher.watson.ibm.com/researcher/view_project.php?id=2505].

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, attributes composites (where there are classes like TimeTable, Accident, House_Number_Range, Validity_Period, Maximum_Height_Allowed), relationships (in which we find the Manoeuvre), transfer points (macroclass which includes classes such as Road, Road_Element, Building, and others), geometry (ie 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 SemanticSensorNetwork Ontology, which provides elements for the description of sensors and their observations [http://www.w3.org/2005/Incubator/ssn/ssnx/ssn] and FIPA Ontology which is more focused on the description of the devices and their properties both HW and SW [http://www.fipa.org/specs/fipa00091/PC00091A.html#_Toc511707116]. We therefore believe these ontologies are to be taken into account when we will have data applicable to these areas.

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 semantics to the various entities of our ontology, 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 (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 vocabulary used during the development of the km4city model.

However, another interesting ontology more e-commerce oriented, is the GoodRelations 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; so products and services offered can increase their visibility into latest generation search engines, or recommendations systems and similar applications.

The possible integration between our KnowledgeModel and the GoodRelations ontology could be achieved at class level: the classes km4c:Service and GoodRelations:Locality can be identifying as equivalents from a definition point of view, and then a service XXX can be connected to the respective locality XXX, defined in the GoodRelations ontology (through the ObjectProperty km4c:hasGRLocality).

The Knowledge Model

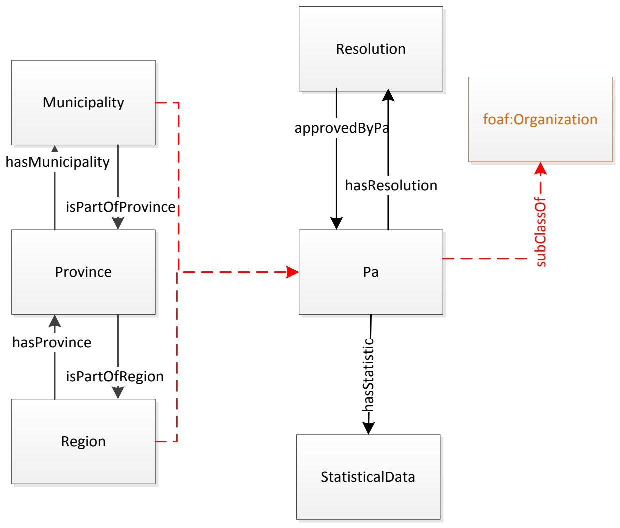
The km4city knowledge model will enable interconnection, storage and the next interrogation of data from many different sources, such as various portals of the Tuscan region (MIIC, Muoversi in Toscana, Osservatorio dei Trasporti), Open Data provided by individual municipalities (mainly Florence). It is therefore evident that the ontology will be built, will not be small, and so it may be helpful to view it as consisting of various macro classes, and to be precise, at present, the following macro-categories have been identified:

1. Administration: the first macroclass that is possible to discover, whose main classes are PA, Municipality, Province, Region, Resolution.
2. Street Guide: formed by classes like Road, Node, RoadElement, AdminidtrativeRoad, Milestone, StreetNumber, RoadLink, Junction, Entry, EntryRule and Maneuver.
3. Points of Interest: includes all services, activities, which may be useful to the citizen, and that may have the need to reach. The classification of individual services and activities will be based on classification previously adopted by the Tuscany Region.

4. Local Public Transport: currently we have access to data relating to scheduled times of the leading LPT, 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 macroclass relative to data coming from sensors is developing. Currently in the ontology have been integrated data collected by various sensors installed along some roads of Florence and in that neighborhood, and those relating to free places in the major parks of the whole region; in our ontology is already present the part relating to events/emergencies, where, however, the collected data are currently very limited in number plus several months old. In addition to these data, in this macroclass were included also data related to Lamma's weather forecast.
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.

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

The first macroclass, Administration is composed as shown in the following figure.



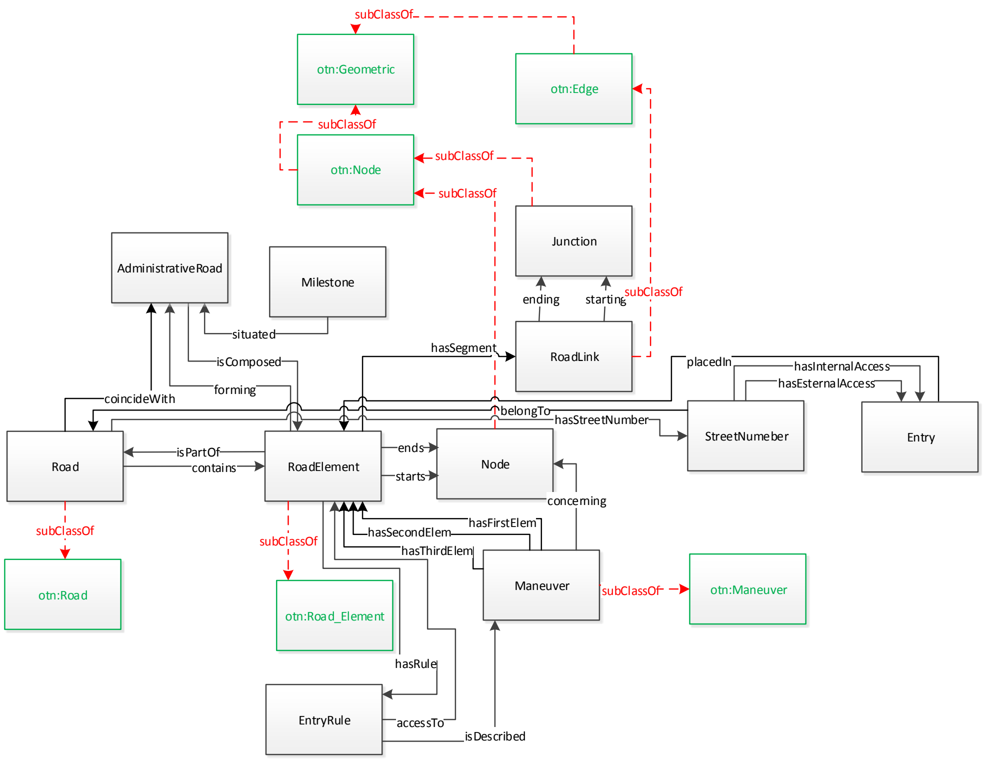
The main class is PA, which has been defined as a subclass of foaf:Organization, link that helps us to assign a clear meaning to our class. The 3 subclasses of PA are automatically defined according to the restriction on ObjectProperties (represented in the figure by solid lines). For example, the Region class is defined as a restriction of the class PA on ObjectProperty "hasProvince", so that only the PA that possess provinces, can be classified as a region. Another example: to define the PA elements that make up the Municipality class was instead used a restriction on ObjectProperty "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 ObjectProperties: "hasProvince" and "isPartOfRegion", "hasMunicipality" and "isPartOfProvince."

Connected to the class PA through the ObjectProperty "hasResolution", we can find the Resolution class, whose instances are represented by the resolutions passed by the various PA note. "hasResolution" ObjectProperty has its inverse, that is, "approvedByPa."

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

The macroclass Street Guide is instead shown in the following figure.



The main class, in the middle of Street Guide macroclass, is RoadElement, which is defined as a subclass of the corresponding element in the ontology OTN, ie Road_Element. Each RoadElement is delimited by a start node and an end node, detectable by the ObjectProperties "startsAtNode" and "endsAtNode", which connect the class object of the class Node. Some restrictions have been specified in the RoadElement class definition, related to the Node class: a road element must have both "startsAtNode" and "endsAtNode" ObjectProperty, both with a cardinality exactly equal to 1. One or more road elements forming a road: the class Road is in fact defined as a subclass of the corresponding class in the OTN, ie the homonymous Road, and with a cardinality restriction on "containsElement" ObjectProperty, which must be a minimum equal to 1, ie cannot exist a road that does not contain at least one road element. Also the class AdministrativeRoad, which represents the administrative division of the roads, is connected to the class RoadElement through two inverse ObjectProperty "hasRoadElement" and "formAdminRoad", while it is connected with only one ObjectProperty to the Road class ("coincideWith"). Better clarify the relationship that exists between Road, AdministrativeRoad and RoadElement: a Road's instance can be connected to multiple instances of AdministrativeRoad (eg if a road crosses the border between the two provinces), but the opposite is also true (eg when a road crosses a provincial town center and assumes different names), ie there is a N:M relationship between the two classes. On each road element is possible to define access restrictions identified by the class EntryRule, which is connected to the class RoadElement through 2 inverse ObjectProperties, ie "hasRule" and "accessToElement". The EntryRule class is defined with a restriction on the minimum cardinality of ObjectProperty "accessToElement" (set equal to 0), which in most cases only one element has an associated road, but in some exceptional cases, there is 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 Maneuver and the class EntryRule are connected by "hasManeuver" ObjectProperty. The term maneuver refers primarily to mandatory turning maneuvers, priority or forbidden, which are described by indicating the order of road elements involving. By analyzing the data from the Roads graph has been verified that only in rare cases maneuvers involving 3 different road elements and then to represent the relationship between the classes manoeuvre and RoadElement, were defined 3 ObjectProperties: "hasFirstElem", "hasSecondElem" and "hasThirdElem", in addition to the ObjectProperty that binds a maneuver to the junction that is interested, that is, "concerningNode" (because a maneuver takes place always in the proximity of a node). In the manoeuvre class definition there are cardinality restrictions, set equal to 1 for "hasFirstElem" and "hasSecondElem" and set maximum cardinality to 1 for "hasThirdElem", as for the maneuvers that affect only 2 road elements, this last ObjectProperty 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 Node class, subclass of the same name belonging to ontology OTN Node class. The Node class has been defined with a restriction on DataProperty geo: lat and geo: long, two properties inherited from the definition of subclass of geo: SpatialThing belonging to ontology Geo wgs84: in fact, each node can be associated with only one pair of coordinates in space, and cannot exist a node without these values.

The Milestone class represents the kilometer stones that are placed along the administrative roads, that is, the elements that identify the precise value of the mileage at that point, the advanced of the route from the starting point. A Milestone may be associated with a single AdministrativeRoad, and is therefore defined a cardinality restriction equal to 1, associated with ObjectProperty "placedInElement". Also the Milestone class is defined as subclass of geo:SpatialThing, but this time the presence of coordinates is not mandatory (restriction on maximum cardinality must be equal to one, but that does not exclude the possible lack of value).

The street number is used to define an address, and it is always logically related to at least one access, in fact every street number always corresponds to a single external access, which can be direct or indirect; sometimes it can also be a internal access. Looking at this relationship from the access point of view, you can instead say that each of these is logically connected to at least one street number. Were then defined StreetNumber and Entry classes.

With the data owned is possible to connect the class StreetNumber to the class RoadElement and to the class Road, respectively through the ObjectProperties "placedInElement" and "belongToRoad". This information is actually redundant and if deemed appropriate, may be delete the link connect to the class Road, in favor of the one towards RoadElement, more easily obtainable from data; it is for this reason that for the ObjectProperty "belongToRoad" has been defined also the inverse "hasStreetNumber".

Within the ontology therefore, the StreetNumber class is defined with a cardinality restriction on the ObjectProperty "belongToRoad", which must be equal to 1.

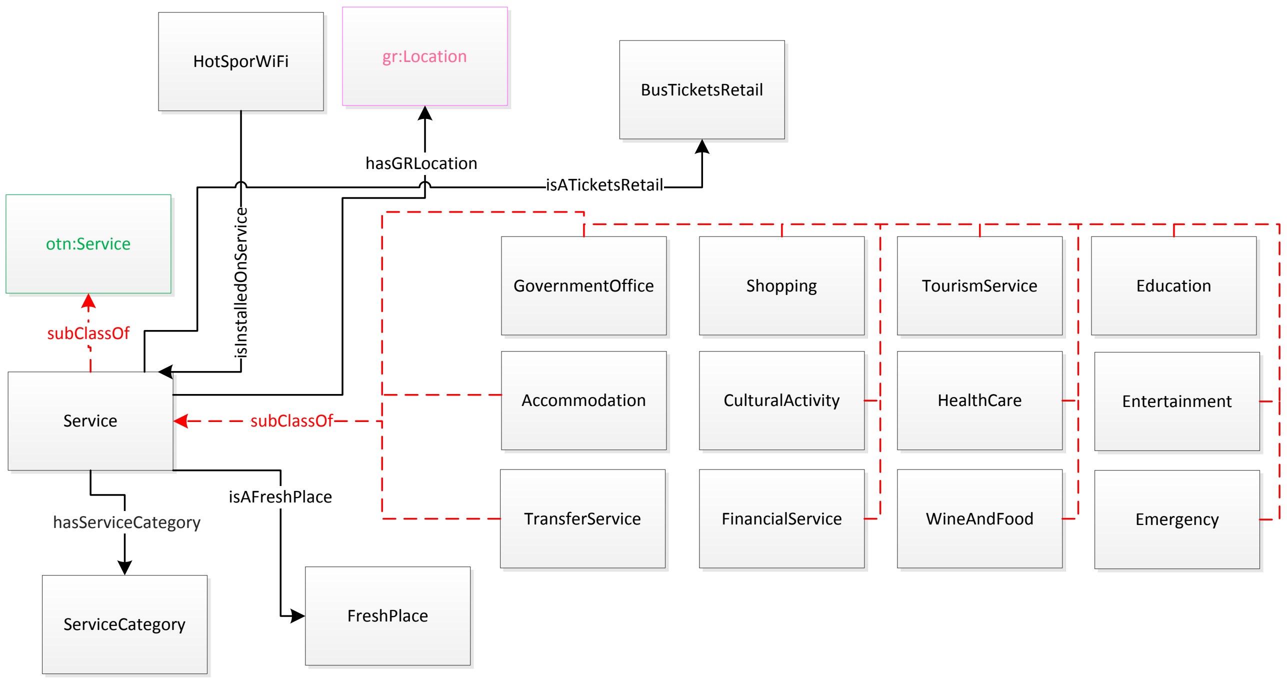
Entry class also can be connected to both the street number and road element where is located. The relationship between Entry and StreetNumber, is defined by two ObjectProperties, "hasInternalAccess" and "hasExternalAccess", on which have been defined cardinality restrictions, since, as mentioned earlier, a street number will always have only one external access, but could also have an internal access (the latter restriction it is in fact defined by setting the maximum cardinality equal to 1, ie they are allowed values 0 and 1). Also the Entry class is defined as a subclass of geo: SpatialThing, and 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 DataProperty of the Geo ontology, set to 1, so which may take 1 value , or none).

The Street macroclass is connected to 2 different Administration through the ObjectProperties "ownerAuthority" and "managingAuthority", which as the name suggests, clearly represent respectively the public administration which has extensive administrative, or public administration that manages the road element . This leaves out the representation, only the streets of private property, for which we have not yet identified the best representation in the ontology .

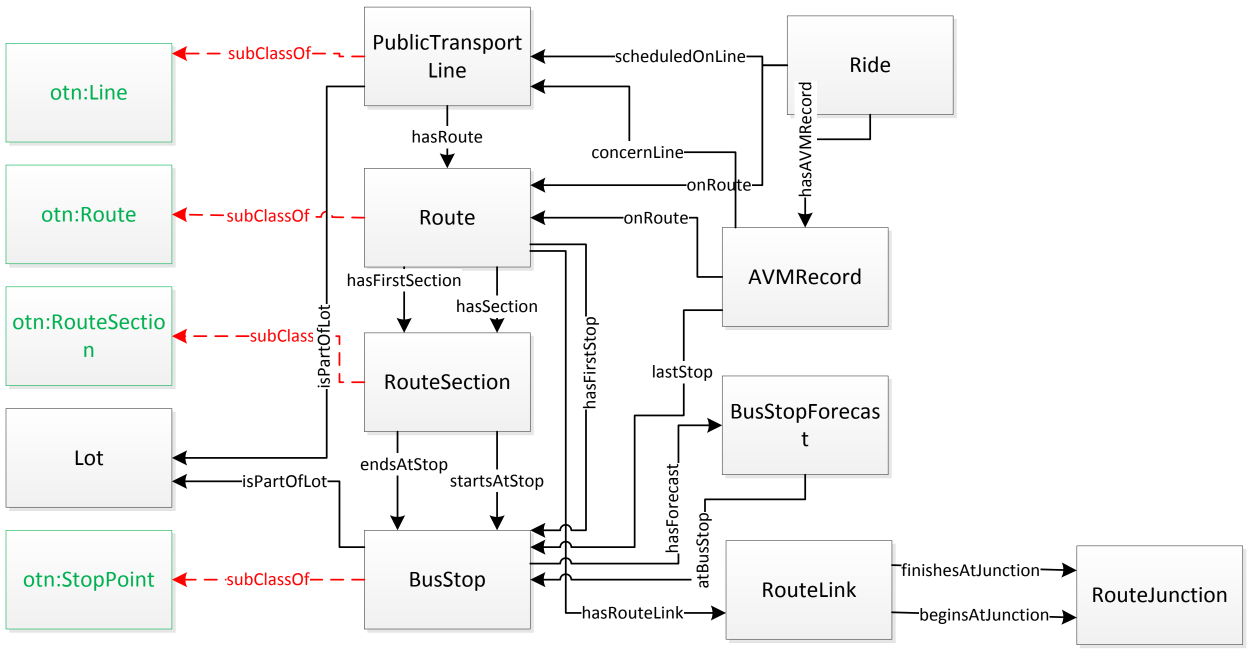
From a cartographic point of view, however, each road element is not a straight line, but a broken line, which will follow the actual course of the road. To represent this situation, the classes RoadLink and Junction have been added: thanks to the interpretation of the KMZ file, we retrieved the set of coordinates that define each RoadElement, and each of these points will be added to the ontology as an instance of Junction (defined as a subclass of geo: SpatialThing, with compulsory single pair of coordinates) . Each small segment between two Junction is instead an instance of the class RoadLink, which is defined by a restriction on the ObjectProperty "endJunction" and "startJunction" (both are obliged to have cardinality equal to 1), which connect the two classes.

For the third macro class, that is, points of interest, have defined a generic class Service, which is associated with the ATECO code , ie the code ISTAT classification of economic activities, which could be used in future as a filter to define the various subclasses, in place of the division into the categories defined by the Region of Tuscany, in order to make more precise research of the various types of services. Unfortunately, however, due to the high percentage of codiciAteco missing, the current division of services into subclasses is based on the hasServiceCategory objectProperty: that property associated with each service an individual belonging to the class ServiceCategory (formed exactly by individuals that have been found on the data provided by the Tuscany Region to which has been associated a label in both Italian and English). In the future we envisage the creation of a dictionary of Individuals that facilitates the addition of labels in other languages. All areas of interest have not yet been defined, since we have not yet provided a list of all services that will be included, but for now in fact, relying on a small list of POI recovered from the site of the MIIC, the following classes have been identified: Accommodation, GovernmentOffice, TourismService, TransferService, CulturalActivity, FinancialService, Shopping, healthcare, Education, Entertainment, Emergency and WineAndFood. The Accommodation class for example, was defined as a restriction on the ObjectProperty serviceCategory, that must take one of the following values: holiday_village, hotel, summer_residence, rest_home, religiuos_guest_house, bed_and_breakfast, hostel, farm_house, agritourism, vacation_resort, day_care_center, camping, historic_residence, mountain_dew, boarding_house. Similarly, other classes have been defined, including among the possible values of the ObjectProperty, those corresponding to that type of service.

Other classes were also defined to allow to represent the additional services that an instance of the class Service can offer: in fact there are some services that also offer a wireless internet connection, for which was defined the HotSpotWifi class that is connected to the Service class via the ObjectProperty isInstalledOnService; that class is also connected to the StreetGraph macroclass because some hot spot wifi in the city of Florence are installed along roads. Another type of additional service is the ticket sales, for which was in fact added the class BusTicketsRetail that is connected to the Service class using the ObjectProperty isATicketsRetail. The city of Florence has also a list of cool places where citizens can stay in the summer to freshen up a bit; so, was included into the ontology the FreshPlace class, which is connected to the class Service using the ObjectProperty isAFreshPlace.



The fourth macro class TPL is not complete yet, it only presents data and classes relating to metropolitan bus lines of Florence and tramway, we assume that this distribution can easily adapt also to the suburban lines of the bus, but we were able to check this statement only in presence of official data.



Each TPL lot, represented by Lot class, is composed of a number of bus or tram lines (class PublicTransportLine), and this relationship is represented by the ObjectProperty "isPartOfLot", which connects each instance of PublicTransportLine to the corresponding instance of Lot. The 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 PublicTransportLine class was in fact connected to the class Ride through the ObjectProperty "scheduledOnLine", on which is also defined as a limitation of cardinality exactly equal to 1, because each stroke may be associated to a single 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 ObjectProperty that connects the two classes, namely "OnRoute"), and the paths can be in a variable number even if referring to a same line: in most cases are 2, ie 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.

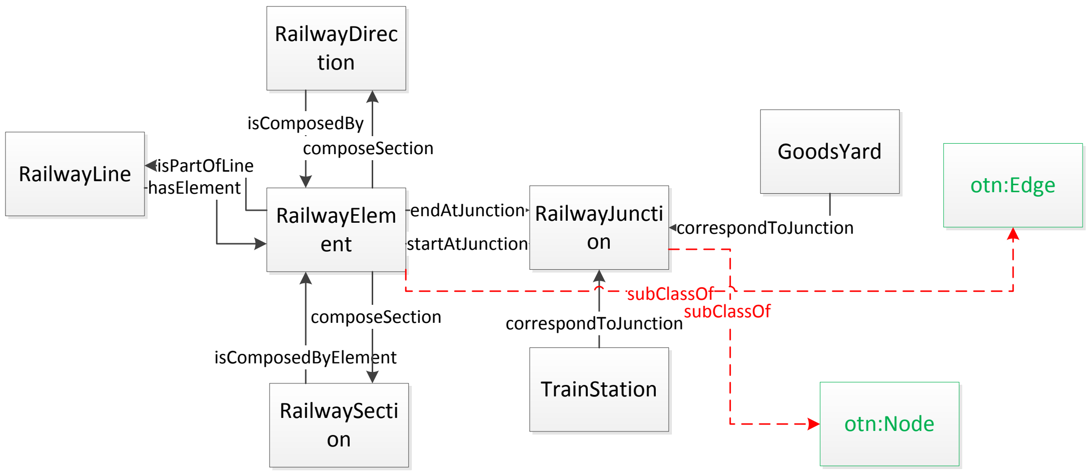
Each path is considered as consisting of a series of road segments delimited by subsequent stops : to model this situation, it was decided to define two ObjectProperty linking Route and RouteSection classes, "hasFirstSection" and "hasSection", since, from a cartographic point of view, wanting to represent 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 ObjectProperty "hasFirstStop", which connects the classes Route and BusStop .

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

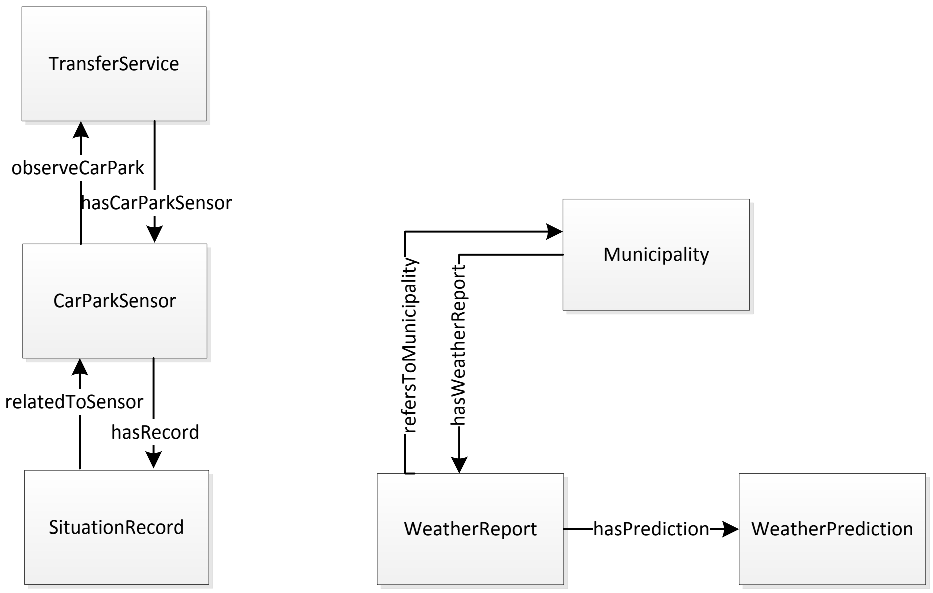
Possessing also the coordinates of each stop, the class BusStop was defined as a subclass of geo:SpatialThing, and was also termed a cardinality equal to 1 for the two DataProperty geo:lat and geo:long.

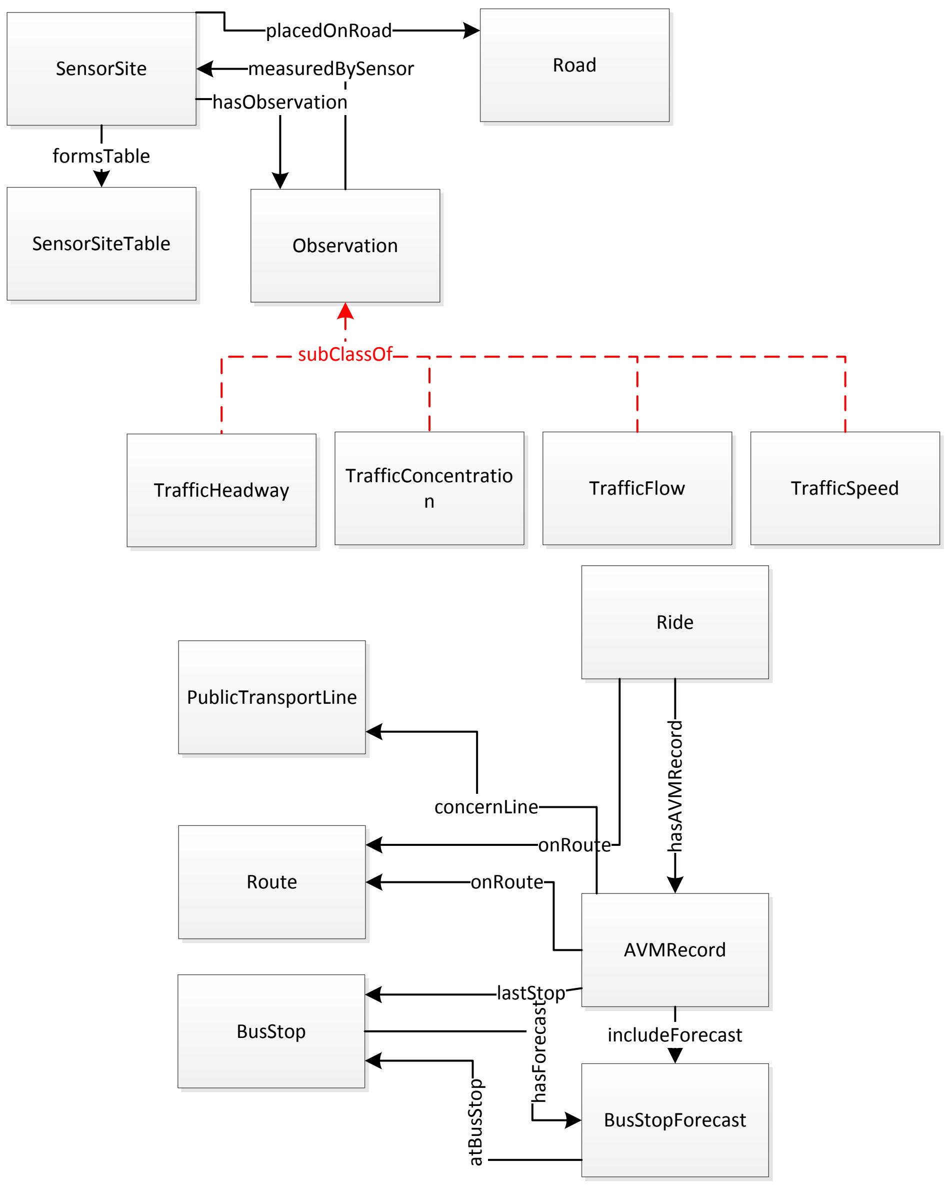
Wishing then to represent to a cartographic point of view the path of a bus, ie a Route instance, we need to represent the broken line that composes each stretch of road crossed by the means of transport itself and to do so, the previously used modeling has been reused to the road elements: we can see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

The part relating to RailwayGraph is mainly formed by the RailwayElement class (defined as a subclass of the OTN:Edge), whose instances represent 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 (ie 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, ie an instance of the class RailwayJunction, defined as a subclass of the OTN:Node. There is also the TrainStation classes that is just a train station, and the class Goodsyard that corresponds to a freight station; usually this both classes correspond to a single instance of the RailwayJunction class.



Sensors Macro class has not yet been completed, but for now it consists of parts shown in the figure, and respectively relating to the car parks sensors, to the weather sensors, to the sensors installed along roads and rails and to the AVM systems.





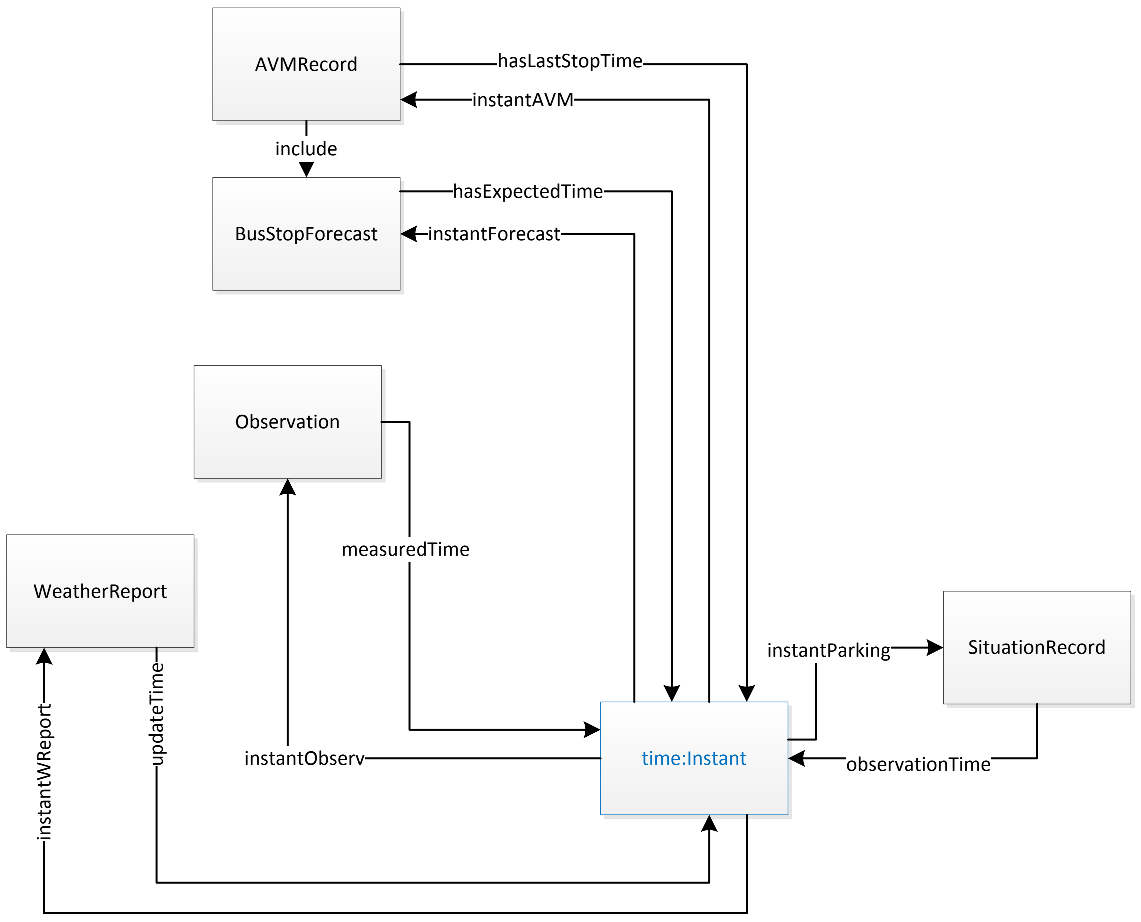
The first part shown in the figure is focused on the real-time data related to parking. The TransferService class, in fact, is connected to the CarParkSensor class, which represents the sensor installed in a given parking and which will be linked to instances of the SituationRecord class, which represent the state of a certain parking at a certain instant; the first link, ie the one between TransferService class and CarParkSensor class, is realized through two invrse ObjectProperty, "observe" and "isObservedBy", while the connection between CarParkSensore class and SituationRecord class, is performed via the reverse ObjectProperty, "relatedTo" and "hasRecord". The class 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.

The second part of the received data in real time, concerns the weather forecast, 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 contains forecast of 5 days divided into range, which have a greater precision (and a higher number) 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 WeatherReport class connected to the WeatherPrediction class via the ObjectProperty "isComposedOf". Municipality class is instead connected to a report by 2 reverse ObjectProperty: "RefersTo" and "has".

The third part of the Real Time data concerns the sensors placed along the roads of the region, which allow to make different detection related to traffic situation. Unfortunately, the location of these sensors is not very precise, it is not possible to place them in a unique point thanks to coordinate, but only to place them within a toponym, which for long-distance roads such as FI- PI-LI road, it represent a range of many miles. Sensors are divided into groups, each group is represented by SensorSiteTable class and each instance of the class SensorSite (that represent a single sensor) is connects to its group through the ObjectProperty "forms" and, as mentioned earlier, each instance of SensorSite class can be connected only to the Road class (through the ObjectProperty "installedOn"). Each sensor produces observations, which are represented by instance of Observation class and these observations can belong to 4 types, ie they can be related to the average velocity (TrafficSpeed subclass), or related to the car flow passing in front of the sensor (TrafficFlow subclass), related to traffic concentration (TrafficConcentration subclass), and finally related to the traffic density (TrafficHeadway subclass). The classes Observation and Sensor are connected via a pair of reverse ObjectProeprty, "hasProduced" and "measuredBy".

The fourth part of RealTime macroclass concerns the AVM systems installed on most of ATAF busses, and it is mainly represented by two classes, AVMRecord and BusStopForecast: the first class mentioned represents a record sent by the AVM system, in which, as well as information on the last stop done (represented by the "lastStop" ObjectProperty that connects the classes AVMrecord to BusStop), GPS coordinates of the vehicle position, and the identifiers of vehicle and line, we also find a list of upcoming stops with the planned passage time; this list have a variable length and it represents instances of the BusStopForecast class . This latter class is linked to the class BusStop through "atThe" ObjectProperty so as to be able to recover the list of possible lines provided on a certain stop (AVMRecord class is in fact also connected to Line class via "concern" ObjectProperty).

Finally, the last macroclass, called Temporal Macro class, is now only "sketchy" within the ontology, and it 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.



We define fictitious URI #instantForecast, #instantAVM, #instantParking, #instantWreport, #instantObserv to following associate them to the identifier URI of a resource referred to the time parameter, ie respectively BusStopForecast, AVMRecord, SituationRecord, WheatherReport and finally Observation.

The fictitious URI #instantXXXX, will be formed as concatenation of two strings: for example, in the case of BusStopForecast instances, it will be concatenate the stop code string (which allows us to uniquely identify them) and the time instant in the most appropriate format. Is necessary to create a fictitious URI that links a time instant to each resource, to not 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.

Pairs of ObjectProperties have also been defined for each class that needs to be connected to the class Instant: between classes Instant and SituationRecord were defined the inverse ObjectProperties “instantParking” and “observationTime”, between classes WeatherReport and Instant, “instantWReport” and “updateTime” ObjectProperties have been defined, between classes Observation and Time there are the reverse ObjectProperties “measuredTime” and “instantObserv”, between BusStopForecast and Time we can find “hasExpectedTime” and “instantForecast” ObjectProperties, and finally, between AVMRecord and Time, there are the reverse ObjectProperties “hasLastStopTime” and “instantAVM”.

The domain of all ObjectProperties with instantXXXX name is defined by elements Time:temporalEntity, so as to be able to expand the defined properties not only to time instant, but also to time intervals.

DataProperties of main classes

Inside the ontology a lot of DataProperties, for which there was the certainty of not being able to use the equivalent values defined on other ontologies, have been defined.

We analyze below, for the main classes in which they were defined, these properties.

Within the class PA were defined only 3 DataProperties, foaf:name that represents the name of the public administration represented by the considered instance, and its unique identifier present in the regional system, dtc:Identified, from DublinCore ontology and dct:alternative where will be stored the municipal code present in the tax code. More information about PA can be found through the link with the Service class, where in fact there are more details that would otherwise be redundant. The Resolution class, whose instance as seen above are the municipality resolutions, has some DataProperties for the definition of an Id, dtc:Identified, the year of approval, km4c:year, and some other property that is coming from the ontology DublinCore dct:subject (the resolution object), dct:created (the date on which the PA has resolved) and foaf:page that represents the URL to which the resolution is available online.

Each instance of the Route class is uniquely identified using the DataProperty dtc:Identified 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. The roadType instead, represents the type of toponym, for example:

- Locality, Square, Plaza, Road, Boulevard, Alley, Lane, etc.

Inside the street graph there are also 2 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 can be associated with the DataProperty km4c:roadName, while the long name, will be store in another DataProperty ie km4c:extendexName and finally, for all the aliases that can be formed (such as for Via S. Marta a possible alternative could be Via Santa Marta. etc.) is used the dct:alternative DataProperty, so that you can define more than one alternate name, and then facilitate the subsequent reconciliations phase.

Concerning the AdministrativeRoad class, in addition to the DataProperties dct:alternative and km4c:adRoadName, that respectively contain the possible alternative names of the road and its official name, the DataProperty AmminClass was defined to represent the administrative classification, that is if a road is:

- highway
- regional road
- road
- municipal road
- Military Road
- Driveway

Finally the field dct:identifier will store 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 all road that have the same ISTAT code, and finally the letters PA.

RoadElement instances are uniquely identified by the DataProperty dtc:Identified, a field of 15 characters 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. Even the elementType DataProperty has been defined for the class RoadElement, and 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 of the Tuscany region, the functional classification is also associated to the RoadElement class, that is defined within the ontology as the ElementClass DataProperty, whose possible values are:

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

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

Concerning the class road element width, a reference to DataProperty width was made, which allows to detect the band of belonging: "less than 3.5m", "between 3.5 and 7.0m," "greater than 7.0 meters" or "not detected"; for the length instead the reference DataProperty is length, a freely insertable value that does not refer to any band. Other data available on the streets graph, essential for defining the maneuvers permitted, is the travel direction of the road element, indicated by trafficDir DataProperty which may take one of the following 4 values:

- road section open 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 operatingStatus DataProperty instead serves to track the operating status of the different road elements and can take the values "in use", "under construction" or "abandoned". Finally there is also a DataProperty that takes into account the speed limits on each road element, ie speedLim. Likewise to the Road class, also the road element class requires metadata that enable the proper updating of the dataset, that are dct:created and dct:source, of the Dublin Core ontology.

In the class StatisticalData were defined DataProperty that allow to associate the actual value (stored in value), data necessary to maintain intact its meaning, such as the field dct:description, dct:created and dct:subject.

A node or junction is a point of intersection of the axes of two road elements, and is always a punctual entity, represented in geometric terms, by a coordinates pair. Instances of the Node class can be uniquely identified thanks to the DataProperty dtc:Identified, made, like the previous code, of 15 characters, according to the following rules: 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. Each node is also characterized by a type, represented by the DataProperty nodeType, which can assume the following values:

- 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 DataProperty for localization, namely geo:lat and geo:long.

The access rules are described by instances of the EntryRule class, uniquely identifiable through a dtc:Identified 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. The access rules are then characterized by a type, represented by DataProperty restrictionType, which can assume the following values:

- 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 DataProperty restrictionValue, which can assume different range 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 presents a substantial difference from other 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 ObjectProperties representing this situation.

Within the 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", has been associated with a DataProperty only to the type of maneuver, namely ManoeuvreType, which can take the following values:

- fork
- manovra proibita calcolata (Calculated manoeuvre)
- manovra obbligatoria (Mandatory manoeuvre)
- manovra proibita (Prohibited manoeuvre)
- manovra prioritaria (Priority manoeuvre)

The StreetNumber class, also presents a code dtc:Identified, to uniquely identify the instance of this class, in the same format as above: 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. In Florence there are 2 numberings, the current in black, and the original red, so was inserted the ClassCode DataProperty, which takes into account just the color of the civic and can take the following values : red, black, no color. Each number can also be formed, besides the numerical part always present, by a literal part, represented respectively by number and exponent DataProperties. It was also inserted an additional value, extendNumber, in which will be stored the number together with exponent in order to ensure greater compatibility with the different formats in which could be written/researched instances of this class. Even in this case, metadata are essential to keep track of updates: DataProperties dct:created and dct:source have been inserted from the Dublin Core Ontology.

The Milestone class, as seen above, identifies the value of the mileage progressively, with respect to its starting point. Even this class has a unique identification code consists of 15 characters, represented by dtc:Identified: 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.

At each milestone usually is written the mileage, which corresponds to that point and the value of this writing is stored through the DataProperty 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, could then be further defined an ObjectProperty linking the Milestone class to the Road class; this information is still recoverable with an extra step through the RoadElement class, but it can be easily inserted if deemed appropriate in the future. Also in this case the DataProperties for localization, ie geo:lat and geo:long, are defined.

Access is the point element that identifies on the territory directly or indirectly external access to a specific place of residence/business; access materialized in practice the "plate" of the street number. As previously mentioned, all access is logically connected to at least one number. Each instance of the Entry class is uniquely identified by DataProperty dtc:Identified, consisting of 15 characters, like all other codes seen: RT followed by 6 characters of municipality ISTAT code, then another 5 character of the progressive from ISTAT code and finally the AC characters. There are only three types of accesses, and this value is stored into entryType DataProperty: "direct external access", "indirect external access" and "internal access", as well as the type of access for this class can be useful to know if there is an access road to property or not (DataProperty porteCochere). Also in this case, the DataProperty for localization, ie geo: lat and geo: long, are present.

The 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, foaf:page, schema:streetAddress, schema:addressLocality, schema:postalCode, schema:region 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 DataProperty were defined: km4c:housenumber to isolate the street number from the address, the km4c:atecoCode, for storing the corresponding Ateco code of each service, and, when possible, the DataProperties geo:lat and geo:long for localization.

In addition to DataProperty just seen, the class CarParkSensor has other properties specifics for car parks: the dct:Identified, always defined at the regional level through a 15 characters code beginning with RT and ending with the initials of the belonging province, capacity, ie the number of parking places, fillrate and exitrate, respectively the number of vehicles entering/leaving, carParkStatus, ie a string that describes the current state of the car park (possible values are "enoughSpacesAvailable", "carParkFull" "noParkingInformationAvailable", etc.), a unique id ie. dct:identified, the validity status of the record (the km4c:validityStatus DataProperty, which can only be "active" for parking), the km4c:parkOccupancy, ie the number of occupied space, or the corresponding percentage that is instead called km4c:occupied, and finally the free places, for which it uses the DataProperty km4c:free.

As regards SituationRecord class, some DataProperties have been defined: a unique id dct:Identified, the record validity status (DataProperty validityStatus, which can only be "active" in the case of parking lots), the parkOccupancy, ie the number of parking places occupied, or the corresponding percentage that is called instead occupied, and finally the vacancy parking places for which there is the DataProperty free. Of course these properties are in addition to the DataProperty dct:created and dct:source.

The WeatherReport class is characterized by DataProperty dct:Identified containing the unique id which identifies the different reports, timestamp that indicates the time when the report was created in milliseconds, dct:created and dct:source . Were added also DataProperties to express the phase of the moon (Lunarphase), the time of sunset/sunrise (that is sunrise and sunset) and the time of moonrise and moonset (moonrise and moonset properties). There is also heightHour and sunHeight DataProperties which represent the time when the sun reaches its maximum height and at which height. Each instance of WeatherPrediction is characterized by DataProperties day which is the day that is referenced in prediction (the day together with the id of the report, form a unique way to identify individual forecasts) the minimum and maximum temperature values or real and perceived temperature values (respectively represented by DataProperties minTemp , maxTemp , recTemp , perTemp), wind direction (wind), humidity that is the percentage of humidity, the level at which there is snow (snow), hour representing the part of the day that is referenced by each individual forecast contained in a report, and the UV index of the day.

The SensorSite and Sensor classes have only one DataProperty concerning the id, represented by dct:identified. The Observation class instead is completed by DataProperties dct:Identified, dct:date, dct:source and dct:created, from DublinCore ontology and DataProperty averageDistance and AverageTime (representing distance and average time between two cars), occupancy and concentration (relative to the percentage of occupation of road referred to the number of cars and the car concentration), vehicleFlow (flow of vehicles detected by the sensors) and data related to the average velocity and the calculated speed percentile: averageSpeed, thresholdPerc and speedPercentile.

The TPLLine class and Lot class have both DataProperties as the dct:identifier and dct:description from DublinCore ontology, representing respectively the number of the line/lot and the description of the path/lot.

The Route class rather than dct:identifier and dct:description DataProperties, presents the field routeLenght, that is the path length in meters and the path direction (direction).

The BusStop class has DataProperties like dct:identifier, dct:created for the date of creation of the record, foaf:name for the name of the bus stop, and geo:lat and geo:long belonging to Geo Ontology. These last two DataProperties are also the only ones in the TPLJunction class.

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

The AVMRecord class requires instead DataProperty to identify the means to which the record refers (vehicle), the arrival time to last stop (lastStopTime), the record creation date and the data source (dct:created and dct:source, respectively), the ride state, that is, whether it is early, late or in time (rideStatus), the managing company and the company that own the AVM system (managingBy and owner properties), the unique identifier of the record (dct:identifier), and coordinates geo:lat and geo:long, which indicated the exact vehicle position at the report time.

Finally, the class Ride has only the dct:identifier DataProperty, like the RouteLink class. The RouteSection class, rather than the identifier, has also the DataProperty km4c:distance, where it is saved the distance between two successive stops within a route.

The TrafficGate class represents the gates of the limited traffic zone present in Florence; this class has as DataProperty the dct:identifier and dct:description, inherited from DublinCore Ontology, the passage type dataProperty namely typeOfGate and the DataProperty for localization geo:lat and geo:long.

The classes HotSpotWiFi, BusTicketsRetail and FreshPlace, all have DataProperty dct: identifier, geo:lat and geo:long useful in identifying the location; moreover the HotSpotWiFi class presents the property dct:description, while the BusTicketRetail class is also defined the dataProperty km4c:typeOfRetail, that indicates the type of store that sells tickets, so to ensure a more rapid detection of the retail.

The Context class has a large number of information from the tables that describe the individual ETL processes which process different public/private data; for each process, in fact, we defines a source type stored within the field dct:source, the date of ingestion into the ontology, namely dct:created; the original data format (CSV, DBF, etc.) stored in the dct:format; a brief description of the dataset in dct:description; the dataset license bound is instead saved into the DataProperty dct:right; the type of process to which it refers is stored into the km4c:ProcessType DataProperty; the DataProperty km4c:automaticity 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 of a person to send and receive emails; DataProperty km4c:accessType talk about how the data are recovered (HTTP calls, Rest, etc..); km4c:period contains the time (in seconds) between two calls of the same process; km4c:Overtime indicates the time after which a process must be killed; The DataProperty km4c:param contains the resource link, if it is an OpenData set retrievable via http; finally km4c:lastUpdate represents the date of the last data update, while km4c:lastTriples that of the last triple generation.

The class RailwayLine has only 3 DataProperties, the dct:identifier that contains the unique identifier of the railway line (a 12 char code starting with the letters RT, followed by 3 characters to identify the region - T09 for Tuscany - 5 char of sequential number and finally the letters PF), the foaf:name in which the convention naming is saved and the dct:alternative in which is instead saved the official name of the Railway Line.

The RailwayDirection class instead has only the first two DataProperty specified for RailwayLine, with the same use: dct:identifier, where is stored in the code consists of 12 char starting with the letters RT, followed by 3 characters that identify the region - T09 for Tuscany - 5 char to the sequential number and finally the letters ED, and the DataProperty foaf:name where is stored in the convention naming.

The class RailwayElement, has the usual field dct:identifier, consisting of 12 characters that follow the following rules: 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. In addition to this, the DataProeprty km4c:elementType (which can take the following three values "ordinary railroad" "railroad AC/AV" and "other") has been defined with the operatingStatus DataProperty, that can take only the values "railway construction", "railroad in operation" and "disused railway"; the km4c:elemLocation indicates the rail element location (and can take the only the values "grade-level", "on bridge/viaduct" and "in tunnel"); the three last DataProperties defined are: the supply DataProperty that specifies whether this is an "electrified line" or a "non-electrified" line, the km4c:gauge, a field that specified if the gauge is "reduced" or "standard", and finally the 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

Other DataProperty that have been defined for the RailwayElement class are km4c:length that is the item length in meters, km4c:numTrack ie the number of tracks (0 if the line is under construction or abandoned), and finally km4c:tracktype, which specifies if the element consists of "single track" or "double track".

The class RailwaySection requires the km4c:axialMass DataProperty, ie 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

Were then defined DataProeprties 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 code 12 char that begins with RT characters, followed by the regional code and 5 number for the sequential number and that ends with a TR), and the foaf:name ie the naming convention of line.

For RailwayJunction class only 3 DataProperties were defined: dct:identifier, that is the identification code of 12 char format as in previous cases, but ending in GK, foaf:name, ie the official name for junctions, stations and rail yard, and finally 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 TrainStation presents the usual 12-digit DataProperty dct:identifier (consisting of RT followed by 3 char of regional identification - T09 for Tuscany - 5 char of progressive number and finally the characters SF), the DataProperty foaf:name in which the official name is memorized; the address retrieved from the list posted on the RFI's website is stored in the fields v:country-name, v:locality, v:postal-code, v:street-address, and the managing body found on RFI's website is stored into the DataProperty km4c:managingAuth; the DataProperty category contains the category to which the station belongs, and finally km4c:state DataProperty, contains the state of the station which can take only the values "Active", "not Active" and "optional stops on demand".

The Goodyard class, in addition to the 12 char code (format as all of the above but ending in SM) stored in dct:identifier, has the DataProperty foaf:name in which the name of freight facility is saved; the railDepartment DataProperty keeps the name of the railway compartment, whereas km4c:railwaySiding is the definition of the physical characteristic of the junction number, km4c:yardType indicates whether the yards are public (value "public yard") or if the junctions are for private use (value "junction in line") , and finally the DataProeprty km4c:state indicates if the yard is "active" or "under construction."