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

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: http://www.disit.org/6506

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

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

info from info@disit.org

version 3.0, of this document

referring to version 1.6.2 of the ontology

date 31-08-2015

Questo documento in Italiano: http://www.disit.org/6461

This document in English: http://www.disit.org/5606

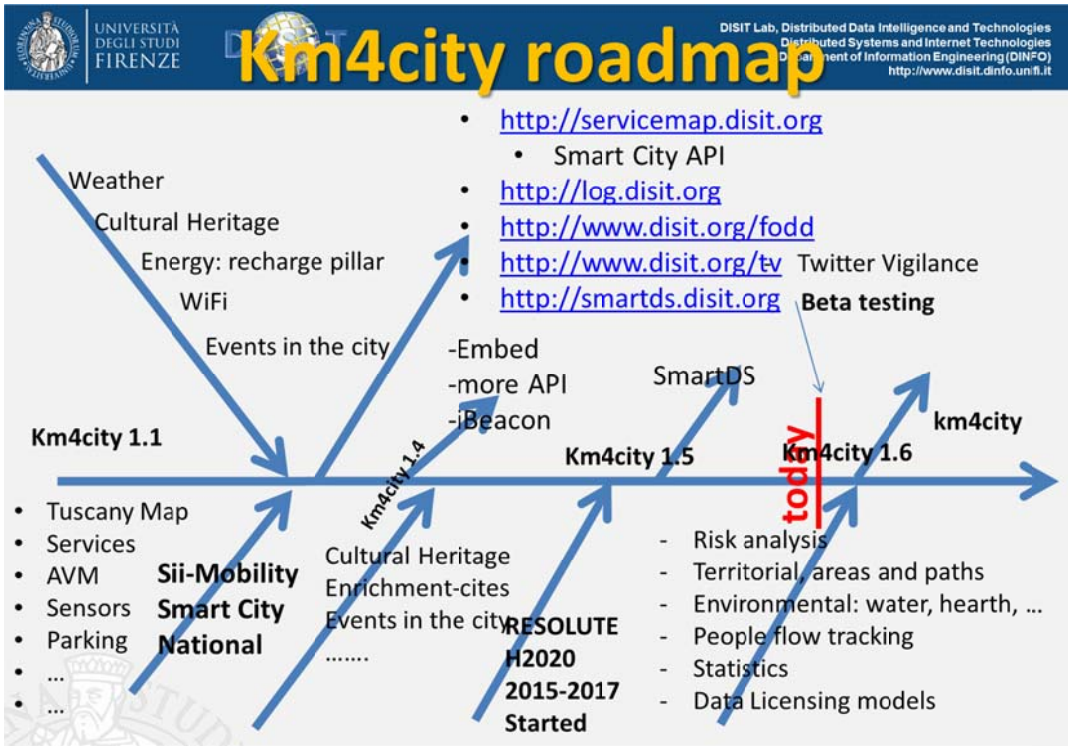
Aim

- **Provides 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**:
  - 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 higly ranked by Ready4SmartCities FP7 CSA http://smartcity.linkeddata.es . On the oher 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:

- 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



You can find updated information on the following links and documents:

- **Service Map tool**: http://servicemap.disit.org a tool for developers to pose geographic queries (learn and generate code queries in an esy manner) and see the knowledge base produced by the harvesting process that includes: "grafo strade" (street graph) from Tuscany region, open data from Florence Municipality, traffic monitoring, geo and weather information from LAMMA, traffic sensors. Some of them in real time. It accessed to an RDF Store based on Florence open data and on Tuscany region mobility data on the basis of **Km4City model**.
    - **Data Sets included into the Km4City model as ServiceMap.disit.org**, coverage all Tuscany for almost all structural data and main services. More details on Florence and Empoli for real time information in the Florence Metropolitan Area.
    - API of the ServiceMap service http://www.disit.org/6597
    - ServiceMap **embedded** into third party web pages
    - Km4City tools and features in **Italiano**, **km4city per principianti**
    - Demonstrative Mobile Application http://www.disit.org/6595
    - **Twitter Vigilance:** to follow the users and sentiments on keywords passing on twitter
    - **Smart Decision System,** decision support system for smart city based on System Thinking, connected to Km4City model.
    - **Origin Destination Matrix** for Smart City and user behavior analysis
- **Km4City Smart City Ontology and services**: knowledge model and ontology for Smart City
    - documentation ENG: http://www.disit.org/5606
    - documentation ITA: http://www.disit.org/6461
    - image: http://www.disit.org/6507
    - km4city ontology .. the OWL and triple versions http://www.disit.org/6506
    - Status, plan and trend of Km4City, next planned developments, http://www.disit.org/6669
    - classification of service categories
    - km4city Schema according to W3C http://www.disit.org/km4city/schema
    - **LOG: Linked Open Graph** http://log.disit.org,
    - **SPARQL entry point test** http://log.disit.org/spqlquery/
    - **RIM: RDF Index Manager**: user manual, for versioning of graph databases RDF stores.
    - **DIM: Data Ingestion Manager:** http://www.disit.org/6732
- **Services and tools**:
    - **Service Map tool**: http://servicemap.disit.org to pose geographic queries and see the knowledge base produced by the harvesting process and provide access to data via API
    - data aggregator and ontological model Km4City, see above
    - **API of the ServiceMap** service http://www.disit.org/6597
    - **Big Data Smart City processes and tools** for km4city, Dec 2014 describing the process for ingesting open data and private data, static and real time data towards and RDF Store
    - **LOG.disit.org** http://LOG.disit.org graph can be used to browse the knowledge model of Smart City, just an example of a Florence segment. http://log.disit.org/service/?graph=71de8caef449ed56143aa95c8c8266ab From that, you can see the whole DISIT knowledge knowledge model for Florence, based on Km4City ontology.
    - **SCE, Smart City Engine** decision support, http://www.disit.org/6515
    - **SmartDS, Smart Decision System**, http://www.disit.org/6711
- **Km4City final user tools:**
    - **WEB Km4City Tool, http://www.km4city.org**
    - **Km4City Mobile Application, for ANDROID on Google Play**
    - **Km4City Mobile Application, for Apple iOS, iPhone, iPad, etc.**
    - **Km4City Mobile Application for ANDROID on Knicket App Search**
    - **Km4City Mobile Application, for Windows Phone, to appear**
    - **Km4City Mobile Application, for other platforms, to appear**
- **DISIT Smart City Articles and conferences, slides, video and presentations:**
    - **European Data Forum**, Luxembourg, november 2015

- **Streaming on Cognitive Science Community http://cognitive-science.info/community/weekly-update/, 10th of september 2015, streaming of IBM community**
- **DMS 2015,** Canada, Vancouver, Canada, 31 Agust- 2 September, km4city with RIM
- **SMAU Firenze, 2015, finalist**
- **Forum PA: Roma may 2015**
- **Florence Open Data Day 2015, http://www.disit.org/fodd**
- describing Km4City model and the ingestion process, http://www.disit.org/6500, while a newer version is published on the international JLVC of Elsevier in the 2014
- **IEEE ICECCS 2014**, Tianjin, China
- **handimatica 2014**: video, le smart city, intervention at the conference opening
- **handimatica 2014**: Slides, http://www.disit.org/6553
- Past DISIT e smart city: http://www.disit.org/6515
- **La città invisibile**, digital data nei contesti urbani, Aula Magna, Gennaio 2015, Video

- **Projects, smart city:**
  - **Sii-Mobility**: National Smart City project (DISIT coordinator)
  - **RESOLUTE H2020, resilience of smart city(DISIT coordinator)**
  - **Coll@bora**: project Social Innovation, smart city national (DISIT coordinator)
  - Agreement with LAMMA and IBINET CNR for the usage of Twitter Vigilance for climate and environment monitoring of human perception
  - **TesysRail**: CTN SC: Cluster Technological National, Smart City and Communities
  - **SMST:** CTN SC: project on Social Museum and Smart Tourism
- **Collaborations**:
  - Agreement among UNIFI e Florence Municipality
  - Agreement among UNIFI DISIT and Tuscany Region MIIC, transport Observatory
  - Agreement among UNIFI DISIT and LAMMA and CNR IBINET for Blog and Twitter vigilance and meteo source channels assessment, http://www.disit.org/tv
  - DISIT is a Smart City Node of the CINI smart city national consortium.
  - DISIT is a node of the Big Data lab of CINI
- **Citations** and ranking:
  - **km4city** has been ranked by Ready4SmartCities FP7 CSA http://smartcity.linkeddata.es
- **Standards**:
  - DISIT belong to standardization group: ISO/IEC JTC 1/SG 1 on Smart Cities
  -

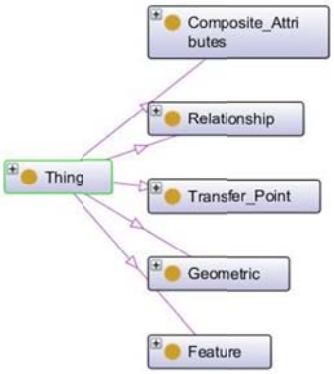
State of the Art

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:Entry and GoodRelations:Locality can be connected to interconnect the represented service to the geolocated couple address-housenumber, (through the ObjectProperty km4c: hasAccess).

Even Schema.org ontology, which already incorporates the GoodRelation 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 projects the Km4City Ontology towards the use of geoSPARQL query language.

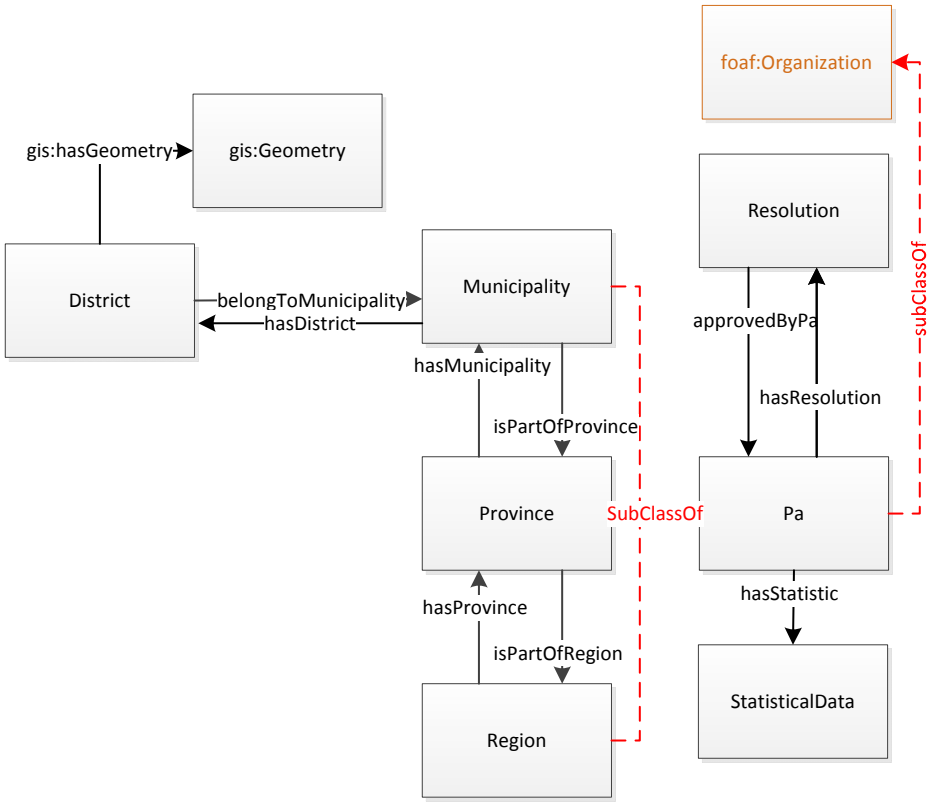
The Knowledge Model

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 and Linked 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. 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 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 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 ObjectProperties (represented in the figure by solid lines). For example, the Region class is defined as a restriction of the class PA on ObjectProperty km4c: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 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 ObjectProperties: km4c:hasProvince and km4c:isPartOfRegion, km4c:hasMunicipality and km4c:isPartOfProvince.

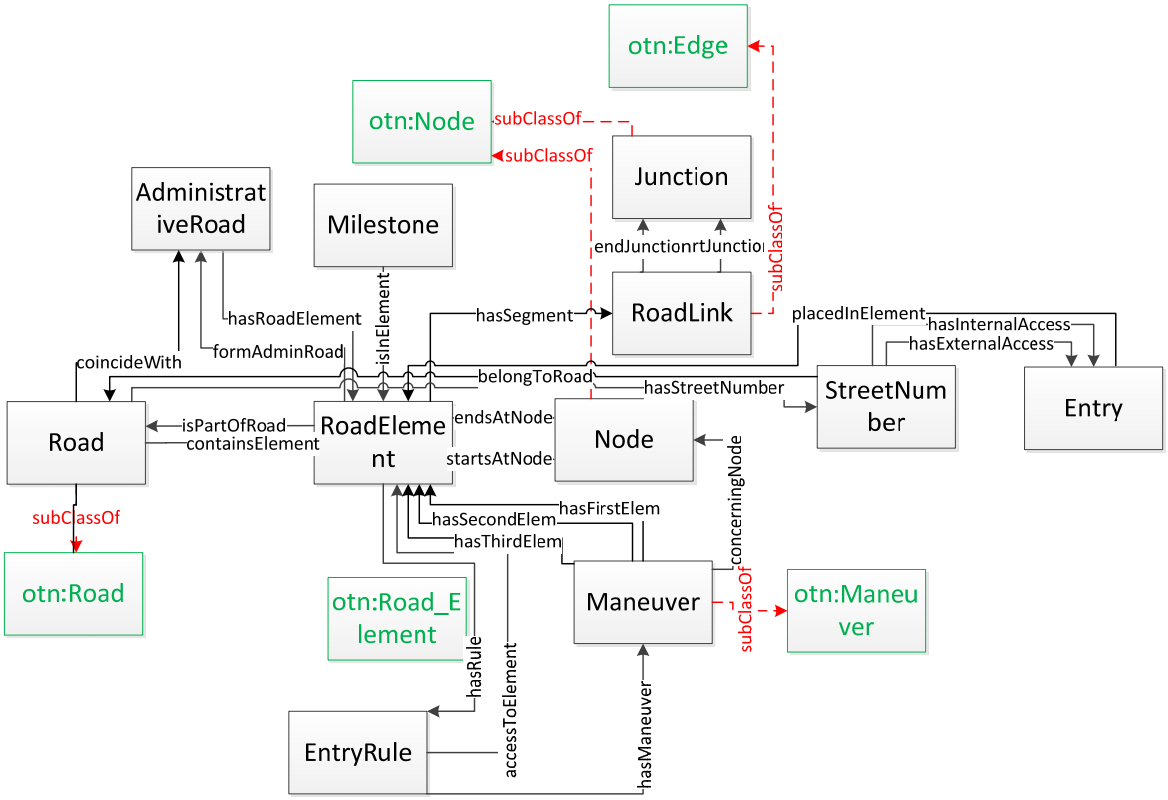
Connected to the class km4c:PA through the ObjectProperty km4c:hasResolution, we can find the km4c:Resolution class, whose instances are represented by the resolutions passed by the various PA note. km4c:hasResolution ObjectProperty has its inverse, that is, km4c:approvedByPa.

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 reverse ObjectProperty: km4c:belongToMunicipality and km4c: hasDistrict.

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, 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

km4c:StatisticalData is connected to both km4c:Pa and km4c:Road through the ObjectProperty km4c:hasStatistic.

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



Clearly this macroclass is the most complex of the Ontology, since it represents the entire Street Graph, various house numbers, access to these latter, but also rules of access to various roads, and finally permitted maneuvers.

The main class, in the middle of Street Guide macroclass, is km4c: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 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 km4c:startsAtNode and km4c:endsAtNode ObjectProperty, both with a cardinality exactly equal to 1. One or more road elements forming a road: the class km4c: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 km4c:containsElement ObjectProperty (whose inverse property is km4c:isPartOfRoad), 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 km4c:AdministrativeRoad, which represents the administrative division of the roads, is connected to the class km4c:RoadElement through two inverse ObjectProperty km4c:hasRoadElement and km4c:formAdminRoad, while it is connected with only one ObjectProperty to the km4c:Road class (km4c:coincideWith). Better clarify the relationship that exists between km4c:Road, km4c:AdministrativeRoad and km4c:RoadElement: a Road's instance can be connected to multiple instances of km4c: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 km4c:EntryRule, which is connected to the class km4c:RoadElement through 2 inverse ObjectProperties, ie km4c:hasRule and km4c:accessToElement. The km4c:EntryRule class is defined with a restriction on the minimum cardinality of ObjectProperty km4c: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 km4c:Maneuver and the class km4c:EntryRule are connected by km4c: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 km4c:Maneuver and km4c:RoadElement, were defined 3 ObjectProperties: km4c:hasFirstElem, km4c:hasSecondElem and km4c:hasThirdElem, in addition to the ObjectProperty that binds a maneuver to the junction that is interested, that is, km4c:concerningNode km4c: (because a maneuver takes place always in the proximity of a node). In the km4c:Maneuver class definition there are cardinality restrictions, set equal to 1 for km4c:hasFirstElem and km4c:hasSecondElem and set maximum cardinality to 1 for km4c: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 the km4c:Node class, subclass of the same name belonging to ontology OTN Node class. The km4c: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 km4c: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 km4c:AdministrativeRoad, and is therefore defined a cardinality restriction equal to 1, associated with ObjectProperty km4c:placedInElement. Also the km4c: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 km4c:StreetNumber to the class km4c:RoadElement and to the class km4c:Road, respectively through the ObjectProperties km4c:placedInElement and km4c:belongToRoad. This information is actually redundant and if deemed appropriate, may be delete the link connect to the class km4c:Road, in favor of the one towards km4c:RoadElement, more easily

obtainable from data; it is for this reason that for the ObjectProperty km4c:belongToRoad has been defined also the inverse km4c:hasStreetNumber.

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

Km4c:Entry class also can be connected to both the street number and road element where is located. The relationship between km4c:Entry and km4c:StreetNumber, is defined by two ObjectProperties, km4c:hasInternalAccess and km4c: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 km4c: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 km4c:ownerAuthority and km4c: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 km4c:RoadLink and km4c:Junction have been added: thanks to the interpretation of the KMZ file, we retrieved the set of coordinates that define each road element, 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 km4c:RoadLink, which is defined by a restriction on the ObjectProperty km4c:endJunction and km4c: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, a generic class Service have been defined; it has been implemented a review of the classes and sub-classes division closely linked to the ATECO code, i.e. the ISTAT code classification of economic activities; the division available in previous versions of the ontology, was in fact based on the companies classification at regional level. Unfortunately, the old services already ingested, which are devoid of ATECO code, will not have a value for the DataProperty km4c:codiceATECO, but according to the information possessed, will still be reclassified within the new hierarchy created.

The hierarchy has not created with a total correspondence with the entire list of ATECO codes, but has created to provide more details of subclasses of interest for our purposes (service providers, retail sales, etc.), and stay more general, where there is less interest (wholesale, industry etc.).

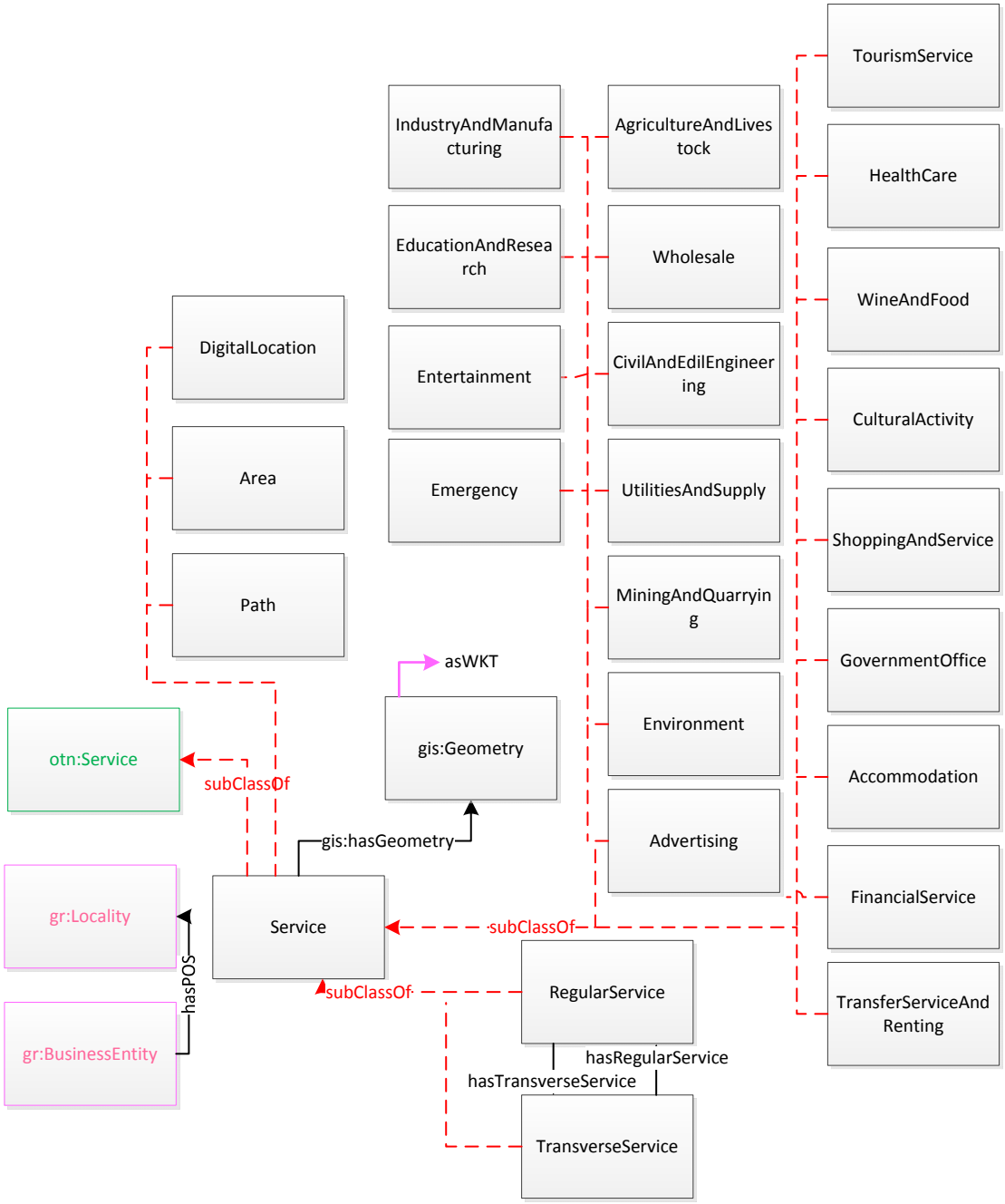
Currently the following classes have been identified: Accommodation, GovernmentOffice, TourismService, TransferServiceAndRenting, CulturalActivity, FinantialService, ShoppingAndService, healtcare, EducationAndResearch, Entertainment, Emergency, WineAndFood, IndustryAndManufacturing,

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 has 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. Currently within Km4City were defined 512 subclasses of services divided as in the following Table:

Accommodation	18
FinancialService	10
Environment	12
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 two classes, called km4c:Path and km4c:Area, collect respectively linear services the first, represented by a broken line on the map, while the second collect polygonal services, which are represented by an area on the map. It is clear therefore that, each not punctual service must be also an instance of one of these two classes.

It was also defined a further services subdivision, i.e. regular services (km4c: RegularService) and transversal services (km4c: TransversalService); this subdivision was necessary since, with the inclusion of new services the possibility that services are connected with each other, has been discovered: for example, thinking about a restaurant that also offers WiFi service with free access. 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 services or regular and secondary services that can be connected to a regular service, through the appropriate reverse ObjectProperties km4c:hasRegularService and km4c:hasTransversalService.

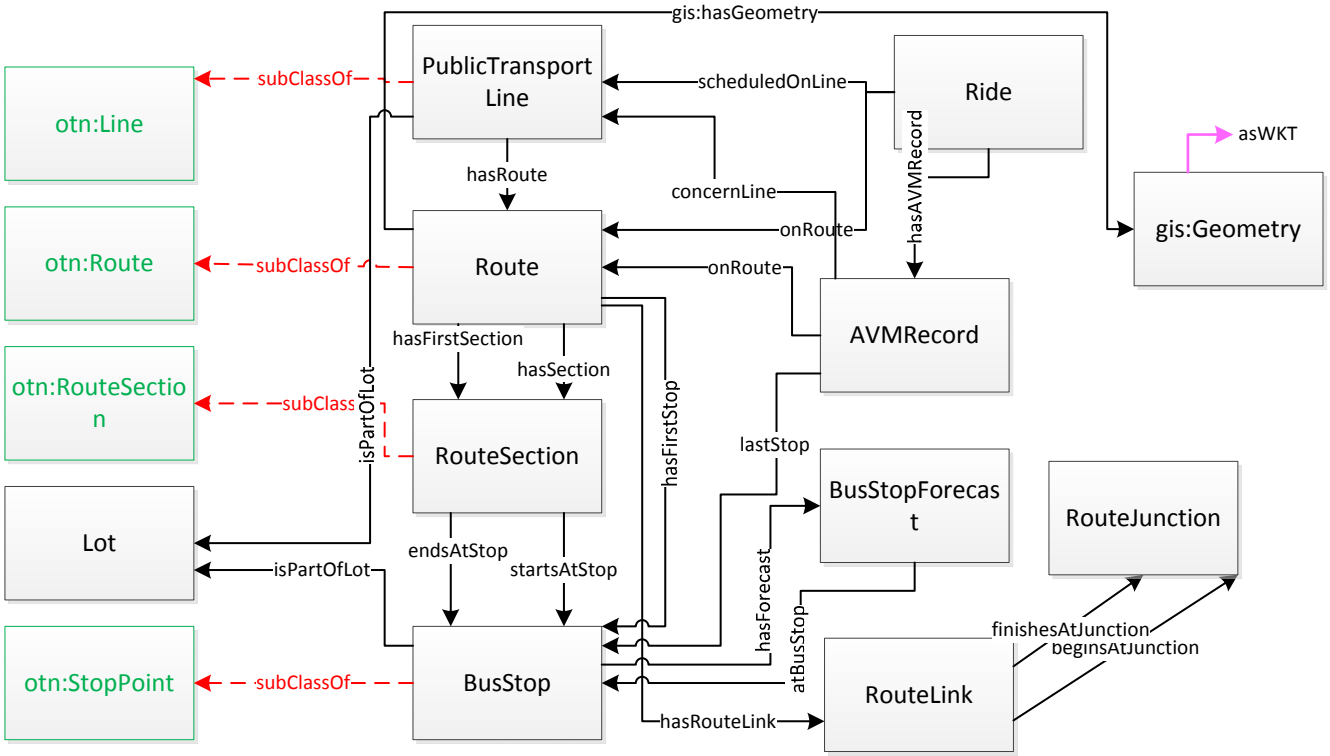


A further part of the Point of Interest macroclass is represented by the integration of Digital Locations provided by the municipality of Florence. The Digital Location data are collected and published by the municipality of Florence as Linked Data, representing locations deemed attractive for the city itself, such as jogging paths, green areas and gardens, monuments and historic buildings, etc. for a total of approximately 3200 different elements. These Digital Location have been re-mapped into the km4c:Service class and, if some of this services have previously been entered, thanks to a reconciliation phase will be possible to connect the pair Service/DigitalLocation that refer to the same building, service, monument, thanks to an owl:sameAs triple, in order to make clear that they represent the same object.

The introduction of geoSPARQL ontology, has also allowed to associate to each services a geometric shape: in fact, 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 in fact has many additional properties and it is also connected, thanks to the ObjectProperty scheme:location, to the km4c:Service class, which in this case corresponds to the place where the event will take place.

The fourth macro class TPL is not complete yet, it only presents data and classes related to metropolitan bus lines of Florence tramway and rail network; the urban bus lines distribution of other areas and the suburban bus lines, are easily adaptable to the model made for metropolitan data in Florence.



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

Each TPL lot, represented by km4c:Lot class, is composed of a number of bus or tram lines (class km4c:PublicTransportLine), and this relationship is represented by the ObjectProperty km4c:isPartOfLot, which connects each instance of km4c:PublicTransportLine to the corresponding instance of km4c: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 km4c:PublicTransportLine class was in fact connected to the class km4c:Ride through the ObjectProperty km4c: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 km4c: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. The ObjectProperty

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 ObjectProperty 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 ObjectProperty linking km4c:Route and km4c:RouteSection classes, km4c:hasFirstSection and km4c: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 km4c:hasFirstStop, which connects the classes km4c:Route and km4c:BusStop .

Applying the same type of modeling used for road elements, two ObjectProperty 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 ObjectProperty 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 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.

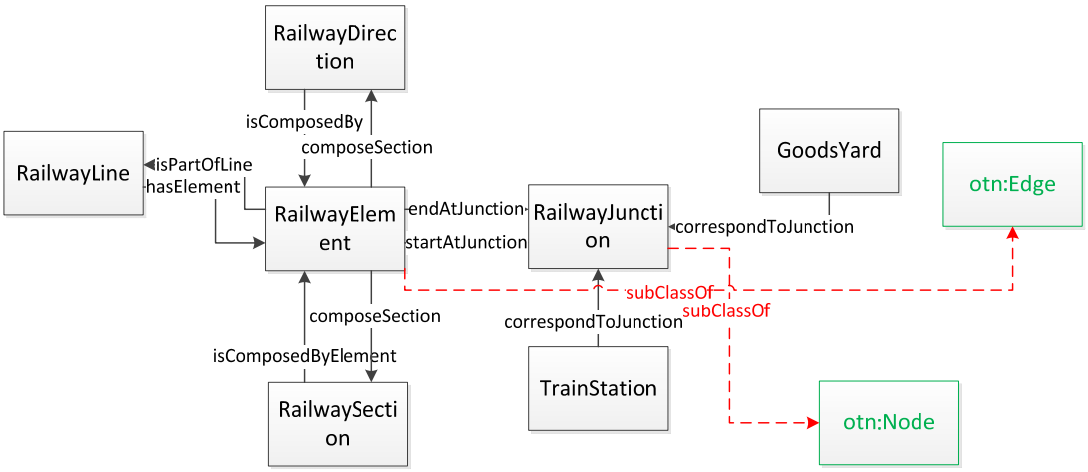
Wishing then to represent to a cartographic point of view the path of a bus, ie a Route instance, we need to represent the broken line that composes each stretch of road crossed by the means of transport itself and to do so, the previously used modeling has been reused to the road elements: we can see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

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

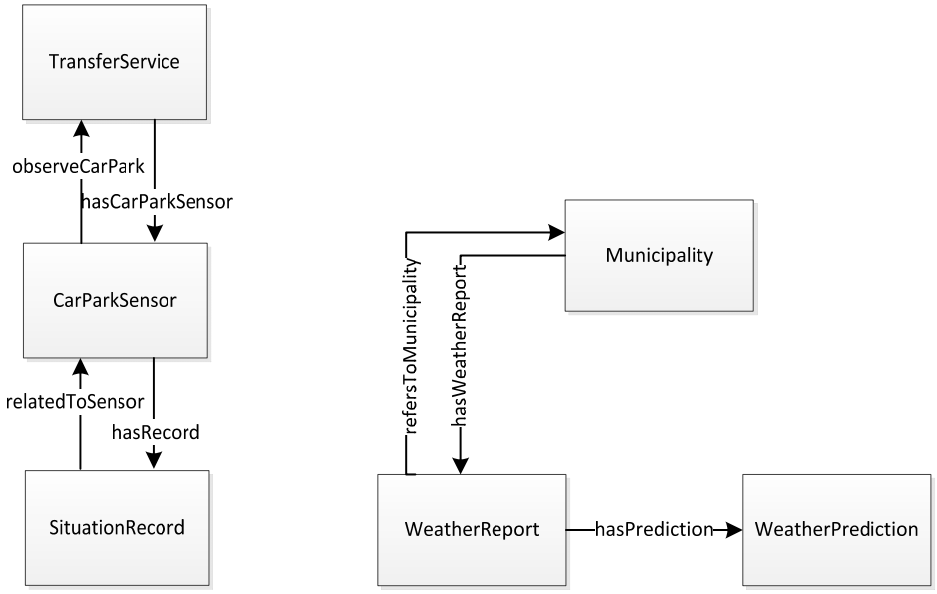
The part relating to RailwayGraph is mainly formed by the km4c:RailwayElement class (defined as a subclass of the OTN:Edge), whose instances represent a single railway element; each element can compose a railway direction, that is a railway line having particular characteristics of importance for traffic 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 km4c:RailwayJunction, defined as a subclass of the OTN:Node. There is also the TrainStation classes that is just a train station, and the class km4c:Goodsyard

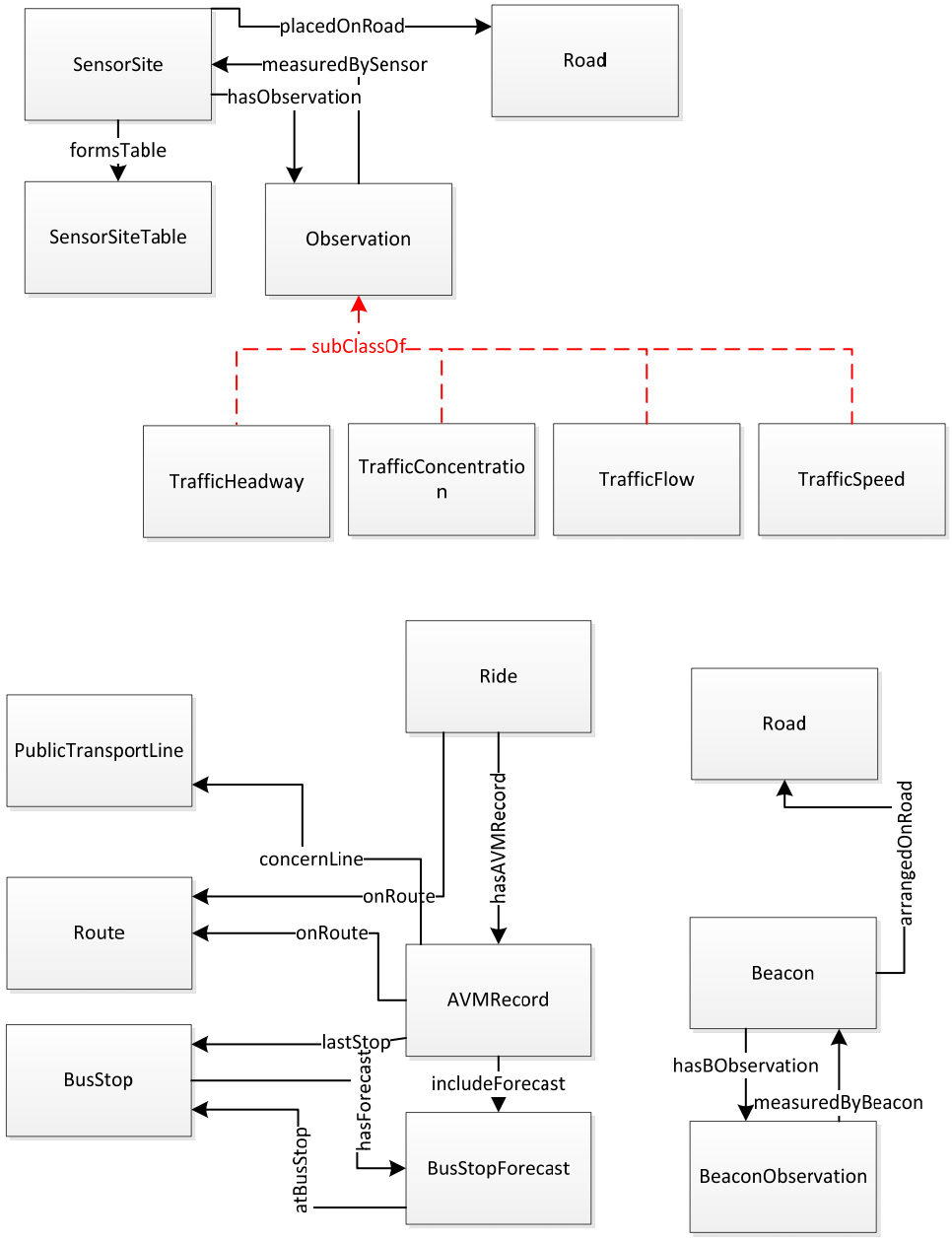
that corresponds to a freight station; usually this both classes correspond to a single instance of the km4c:RailwayJunction class.

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.



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 km4c:TransferService class and km4c:CarParkSensor class, is realized through two invrse ObjectProperty, km4c:observe and km4c:isObservedBy, while the connection between km4c:CarParkSensore class and km4c:SituationRecord class, is performed via the reverse ObjectProperty, 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.

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 km4c:WeatherPrediction class via the ObjectProperty km4c:hasPrediction. Km4c:Municipality class is instead connected to a report by 2 reverse ObjectProperty: km4c:refersToMunicipality and km4c:hasWeatherReport.

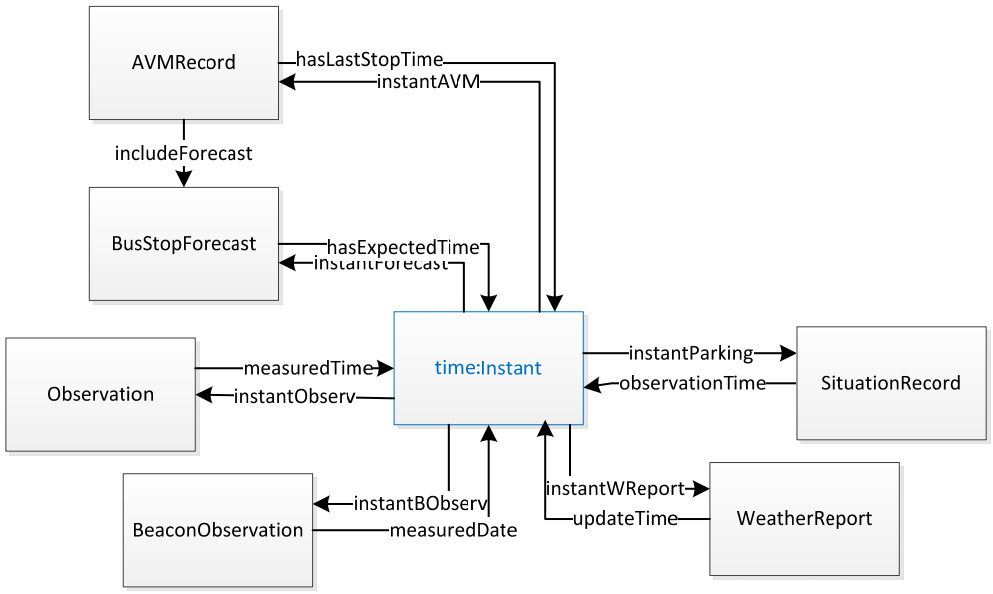
The third part of the Real Time data concerns the sensors placed along the roads of the region, which allow to make different detection related to traffic situation. Thanks to additional information received recently from Osservatorio dei Trasporti, it was possible to accurately geo locate all sensors. Sensors are divided into groups, each group is represented by km4c:SensorSiteTable class and each instance of the class km4c:SensorSite (that represent a single sensor) is connects to its group through the ObjectProperty km4c:formsTable and, as mentioned earlier, each instance of km4c:SensorSite class can be connected only to the km4c:Road class (through the ObjectProperty km4c:installedOnRoad). Each sensor produces observations, which are represented by instance of km4c:Observation class and these observations can belong to 4 types, ie 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 Sensor are connected via a pair of reverse ObjectProeprty, km4c:hasObservation e km4c:measuredBySensor.

The fourth part of RealTime macroclass concerns the AVM systems installed on most of ATAF busses, and it is mainly represented by two classes, km4c:AVMRecord e km4c: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 , km4c:lastStop ObjectProperty that connects the classes , km4c:AVMrecord to , km4c: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 km4c:BusStopForecast class. This latter class is linked to the class km4c:BusStop through km4c:atBusStop ObjectProperty so as to be able to recover the list of possible lines provided on a certain stop (km4c:AVMRecord class is in fact also connected to km4c:Line class via km4c:concernLine ObjectProperty).

The fifth part of the RealTime macroclass, as previously anticipated, regards beacons, for which were created two classes, the km4c:Beacon, which represents a single device installed in the city, and the km4c:BObservation class, which represents instead, a single observation reported by each functional element; this two classes are connected each other via a pair of reverse ObjectProperty km4c:hasBObservation and km4c:measuredByBeacon. Each beacon is uniquely identified by an identification code, and three other codes, the uuid (Universally Unique Identifier, It 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 instead identifying respectively a group of beacons and the number of each element within the group identified with the same major code; coordinates that identify the location where the beacons are located, are also available. The information that a beacon is able to return are the signal strength, the coordinates where it contacts a user, and the date and time of the contact.

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.



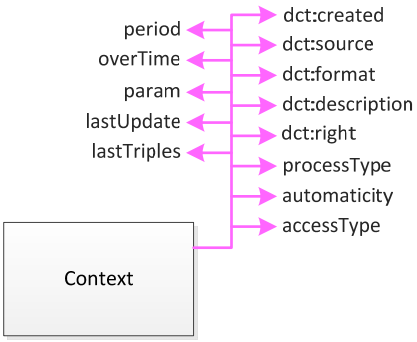
We define fictitious URI #instantForecast, #instantAVM, #instantParking, #instantWreport, #instantObserv to following associate them to the identifier URI of a resource referred to the time parameter, ie respectively km4c:BusStopForecast, km4c:AVMRecord, km4c:SituationRecord, km4c:WheatherReport and finally km4c: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 time:Instant and km4c:SituationRecord were defined the inverse ObjectProperties km4c:instantParking and km4c:observationTime, between classes km4c:WeatherReport and time:Instant, km4c:instantWReport and km4c:updateTime ObjectProperties have been defined, between classes km4c:Observation and time:instant there are the reverse ObjectProperties km4c:measuredTime and km4c:instantObserv, between BusStopForecast and time:Instant we can find km4c:hasExpectedTime and km4c:instantForecast ObjectProperties, and finally, between km4c: AVMRecord and time:Instant, there are the reverse ObjectProperties km4c:hasLastStopTime and km4c:instantAVM; finally between classes km4c:BeaconObservation and time:Instant were defined the reverse properties km4c:instantBObserv and km4c:measuredDate.

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.



The seventh macroclass, as already mentioned above, relates to the metadata associated with each dataset. Sesame [http://rdf4j.org/] allows to define, in the ontology, the Named Graph, that correspond to 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 were then defined all DataProperties that allow to store relevant information, related to a certain dataset, for example: 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

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.

The DataProperty km4c:typeLabel is defined for all existing classes in the ontology, and it is the field where it is saved the type to which owns the instance; thanks to this property, it is possible to activate a FullText search which also includes the type of instances.

Within the class km4c: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, dct: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 km4c:Resolution class, whose instance as seen above are the municipality resolutions, has some DataProperties for the definition of an Id, dct: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).

Each instance of the km4c:Route class is uniquely identified using the DataProperty dct: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 km4c: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 km4c: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 dct: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 km4c:elementType DataProperty has been defined for the class km4c: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 km4c:RoadElement class, that is defined within the ontology as the km4c:elementClass DataProperty, whose possible values are:

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

The km4c: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". 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

Concerning the class road element width, a reference to DataProperty km4c: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 km4c: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 km4c: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, i.e. km4c:speedLimit.

In the class km4c:StatisticalData were defined DataProperty that allow to associate the actual value (stored in km4c:value), data necessary to maintain intact its meaning, such as the field dct:identifier, 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 km4c:Node class can be uniquely identified thanks to the DataProperty dct: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 km4c: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 dct: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 km4c: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 km4c: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 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 presents a code dct: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 km4c: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 km4c:number and km4c:exponent DataProperties. It was also inserted an additional value, km4c: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.

The km4c: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 dct: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 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, could then be further defined an ObjectProperty linking the km4c:Milestone class to the km4c:Road class; this information is still recoverable with an extra step through the km4c:RoadElement class, but it can be easily inserted if deemed appropriate in the future. Also in this case the DataProperties for localization, i.e. geo:lat and geo:long, are defined.

Km4c: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 dct: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 km4c: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 km4c:porteCochere). Also in this case, the DataProperty for localization, i.e. geo: lat and geo: long, are present.

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

DataProperty were defined: km4c:houseNumber to isolate the street number from the address, and, when possible, the DataProperties geo:lat and geo:long for localization.

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

- km4c:hasGeometry: property defined to store the set of coordinates associated with the digitalLocation, which can be a POINT, a LINESTRING and a POLYGON;
- km4c:routeLength: to store the length of the paths (for example of Jogging);
- km4c:stopNumber: property that indicates the number of stop within the entire route of Tramway Line 1;
- km4c:lineNumber: property that indicates the line to which a stop belonging;
- km4c:managingBy: property indicating the manager of DigitalLocation;
- dct:description: property where a description of DigitalLocation is stored;
- km4c:owner: property that indicates the owner of the DigitalLocation;
- km4c:abbreviation: property showing the abbreviation of the DigitalLocation name;
- km4c:type: property indicating the category of the DigitalLocation manager, or the LTZ type or the museum type or the reference area.
- km4c:time: property that indicates the opening hours of a DigitalLocation;
- km4c:firenzeCard: property that indicates if the DigitalLocation is affiliated with the FirenzeCard program;
- km4c:multimediaResouce: property to associate images and mp3 file to individual DigitalLocation;
- km4c:districtCode: property that specifies the code of the district in which the DigitalLocation is located;
- km4c:routePosition: property that indicates the DigitalLocation position within a thematic route;
- km4c:areacode: property indicating a municipal code for the referenced area;
- km4c:routeCode: property that specifies the number of thematic route to which the DigitalLocation refers.
- scheme:price: property that indicates the possible cost of entry;

The class km4c:Event instead possesses all the properties belonging to the schema:Event class and other properties added based on information provided by the municipality of Florence. For convenience below all of these properties will be included in a list.

- schema:startDate: property that indicates the event start date;
- schema:endDate: property that indicates the event end date;
- schema:description: contains the event description;
- schema:image: contains, if any,the url of an event reference image;
- schema:name is the name of the event;
- schema:url: website;
- schema:telephone: reference telephone number for the event;
- schema:streetAddress: address where the event will be held;
- schema:addressLocality: city where the event will be held;
- schema:addressregion: province where the event will be held;
- schema:postalCode: Zip code where the event will be held;
- km4c:houseNumber: civic number of the place where the event will be held;

- skos:note: property that will contain any notes or other information for each event;
- scheme:price: property that indicates the possible cost of entry;
- dct:identifier: contains the identifier assigned to the event by the municipality;
- km4c:eventCategory: property to indicate the event type;
- km4c:placeName: contains the name of the place where the event will take place;
- km4c:freeEvent: float variable that indicates if the event is free or not;
- km4c:eventTime: property that contains the event start time;

In addition to these properties, an event is also fitted with a pair of coordinates to geo localized it, i.e. geo: lat and geo: long.

The gis:Geometry class, instead, presents only the DataProperty 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))

Continuing, the class km4c: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, 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.), a unique id i.e. dct:identified, the validity status of the record (the km4c:validityStatus DataProperty, 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, and finally the free places, for which it uses the DataProperty km4c:free. As regards km4c:SituationRecord class, some DataProperties have been defined: a unique id dct:Identified, the record validity status (DataProperty 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 occupied, and finally the vacancy parking places for which there is the DataProperty km4c:free.

The km4c:WeatherReport class is characterized by DataProperty dct:Identified containing the unique id which identifies the different reports, km4c:timestamp that indicates the time when the report was created in milliseconds. Were added also DataProperties to express the phase of the moon (km4c:lunarphase), the time of sunset/sunrise (that is km4c:sunrise and km4c:sunset) and the time of moonrise and moonset (km4c:moonrise and km4c:moonset properties). There is also km4c:heightHour and km4c:sunHeight DataProperties which represent the time when the sun reaches its maximum height and at which height. Each instance of km4c:WeatherPrediction is characterized by DataProperties km4c: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 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, and the UV index of the day, represented by km4c:uv.

The km4c:SensorSiteTable and km4c:SensorSite classes have only one DataProperty concerning the id, represented by dct:identified. The Observation class instead is completed by DataProperties dct:Identified, dct:date and DataProperties 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 road referred to the number of cars and the car concentration), km4c:vehicleFlow (flow of vehicles detected by the sensors) and data related to the average velocity and the calculated speed percentile: km4c:averageSpeed, km4c:thresholdPerc and km4c:speedPercentile.

The km4c:PublicTransportLine class and km4c: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 km4c:Route class rather than dct:identifier, foaf:name and dct:description DataProperties, presents the field km4c:routeLenght, that is the path length in meters and the path direction (km4c:direction).

The km4c:BusStop class has DataProperties like dct:identifier, 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 km4c:RouteJunction class, with dct:identifier.

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

The km4c:AVMRecord class requires instead DataProperty to identify the means to which the record refers (km4c:vehicle), the arrival time to last stop (km4c:lastStopTime), the ride state, that is, whether it is early, late or in time (km4c:rideStatus), 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), and coordinates geo:lat and geo:long, which indicated the exact vehicle position at the report time.

Finally, the class km4c:Ride has only the dct:identifier DataProperty, like the km4c: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.

Now analyze the DataProperty for classes km4c:Beacon and km4c:BeaconObservation: the first class, in addition to DataProperty 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 (defines if the beacon is public or not), and finally the DataProperty for storing the location, i.e. geo:lat and geo:long. The second class, km4c:BeaconObservation, has instead, in addition to dct:identifier that makes unique each reading, the coordinates where the connection has made, stored in geo:lat and geo:long DataProperties, the dct:date that stores date and time of observation and finally, the DataProperty km4c:power, indicating the power with which a beacon can connect to a user, which also allows to precisely determine the distance.

Were also defined properties 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 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 km4c: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 km4c:RailwayDirection class instead has only the first two DataProperty specified for km4c: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 km4c: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 km4c: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 km4c: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 km4c:RailwayElement class are km4c:length that is the item length in meters, km4c:numTrack i.e. 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 km4c:RailwaySection requires the km4c:axialMass DataProperty, i.e. the classification of the line with respect to the km4c: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 km4c: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 schema:addressRegion, schema:addressLocality, schema:postalCode, schema:streetAddress, 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 km4c: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 km4c: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."