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

Scopo

Lo sviluppo di un ontologia integrata ed unificata per Smart City che includa trasporti, infomobilità' e grandi set di altri Open Data.

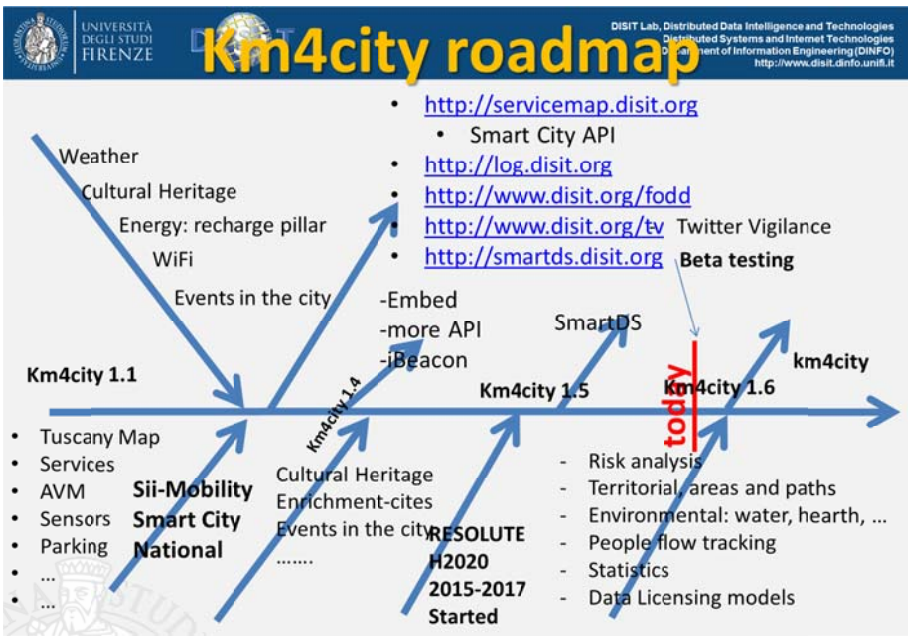
Ci scusiamo per le prossime 3 pagine che non sono state ancora tradotte. In seguito la documentazione dell'ontologia è in italiano.

- **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**

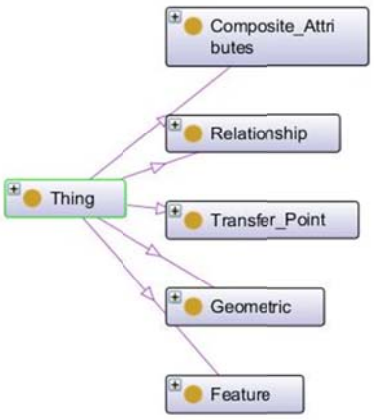
- **DISIT Smart City Articles and conferences, slides, video and presentations:**
  - **European Data Forum**, Luxembourg, november 2015
  - **Streaming on Cognitive Science Community http://cognitive-science.info/community/weekly-update/, 10th of september 2015, streaming of IBM community**
  - **DMS 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 belongs to standardization group: ISO/IEC JTC 1/SG 1 on Smart Cities
  -

Stato dell'Arte

Per interconnettere tra di loro i dati forniti dalla Regione Toscana, gli Open Data del comune di Firenze e gli altri dataset statici e Real Time che ci sono stati forniti, abbiamo iniziato a sviluppare uno Knowledge Model che permetta di raccogliere tutti i dati provenienti dalla città relativi a mobilità, statistiche, grafo strade etc.

L'unica ontologia presentata come un "aiuto per trasformare le città in Smart Cities"', è SCRIBE, realizzata da IBM, della quale però non sono disponibili gratuitamente online molte informazioni [http://researcher.watson.ibm.com/researcher/view_project.php?id=2505].

Tra le altre ontologie che potrebbero essere in qualche modo legate alle Smart Cities, citiamo la OTN, Ontology of Transportation Networks.



Questa ontologia al suo interno definisce l'intero sistema di trasporto in tutte le sue parti, dalla singola strada/ferrovia, fino al tipo di manovra che è possibile effettuare su un segmento stradale o i percorsi del trasporto pubblico. Come è possibile osservare dalla figura a fianco, la OTN raggruppa i concetti che esprime sotto 5 principali macroclassi, quella degli attributi compositi (dove troviamo classi come TimeTable, Accident, House_Number_Range, Validity_Period, Maximum_Height_Allowed), delle relazioni (all'interno della quale troviamo le Manoeuvre), dei punti di trasferimento (macroclasse che comprende classi come Road, Road_Element, Building e altre), delle geometrie (cioè le classi Edge, Node e Face) e infine delle caratteristiche (che raccoglie classi come Railways, Service, Road_and_Ferry_Feature, Public_Transport).

In rete si trovano inoltre molte altre ontologie relative alle sensor networks, come ad esempio la SemanticSensorNetwork Ontology, che mette a disposizione elementi per la descrizione dei sensori e delle loro osservazioni [http://www.w3.org/2005/Incubator/ssn/ssnx/ssn] e la FIPA Ontology nella quale invece si da più importanza alla descrizione dei devices e delle loro proprietà sia HW che SW [http://www.fipa.org/specs/fipa00091/PC00091A.html#_Toc511707116]. Riteniamo quindi tali ontologie siano da tenere in considerazione nel momento in cui entreremo in possesso di dati applicabili in tali ambiti.

Analizzando poi i dati messi a disposizione dalle PA (principalmente dalla Regione Toscana) è stato riscontrato che, in relazione al Grafo Strade, i dati forniti potevano essere facilmente mappati in ampie parti della OTN, relative allo stradario; tenendo conto di queste similitudini riscontrate, riteniamo che potrebbe essere utile, allo scopo di associare una semantica più precisa alle varie entità della nostra ontologia, fare riferimento esplicito alla OTN, in modo tale anche da rendere i concetti espressi più chiari e più facilmente linkabili ad altri, volendo seguire le linee guida per la realizzazione di Linked Open Data.

Seguendo questo principio, è stato quindi realizzato, per ora parzialmente, quello che sarà il Knowledge Model km4city.

Un'altra interessante ontologia, rivolta però maggiormente verso l'e-commerce, è GoodRelations, cioè un vocabolario standardizzato che permette di descrivere dati relativi a prodotti, prezzi, negozi ed aziende in modo tale possano essere inclusi all'interno di pagine web esistenti e possono essere interpretati da altri computer, in modo tale da incrementare anche la visibilità di prodotti e servizi offerti nei motori di ricerca di ultima generazione, sistemi per raccomandazioni e applicazioni simili.

La possibile integrazione di tra l'ontologia km4c e la GoodRelations potrebbe essere realizzata a livello di classe: le due classi km4c:Entry e goodrelations:Locality, posso essere connesse in modo da collegare il servizio al punto di accesso (attraverso l'ObjectProperty km4c:hasAccess).

Anche l'ontologia Schema.org, che integra già il progetto GoodRelation, è stata fondamentale per la definizione degli eventi: la classe creata eredita infatti la struttura della schema:Event ed è stata poi espansa in base alle informazioni in nostro possesso; la stessa ontologia è stata sfruttata per la definizione della contact card delle aziende.

Un'altra ontologia che è stata utilizzata per la definizione delle forme geometriche, è la geoSPARQL, la cui integrazione proietta la Km4City verso l'utilizzo del linguaggio di query geoSPARQL.

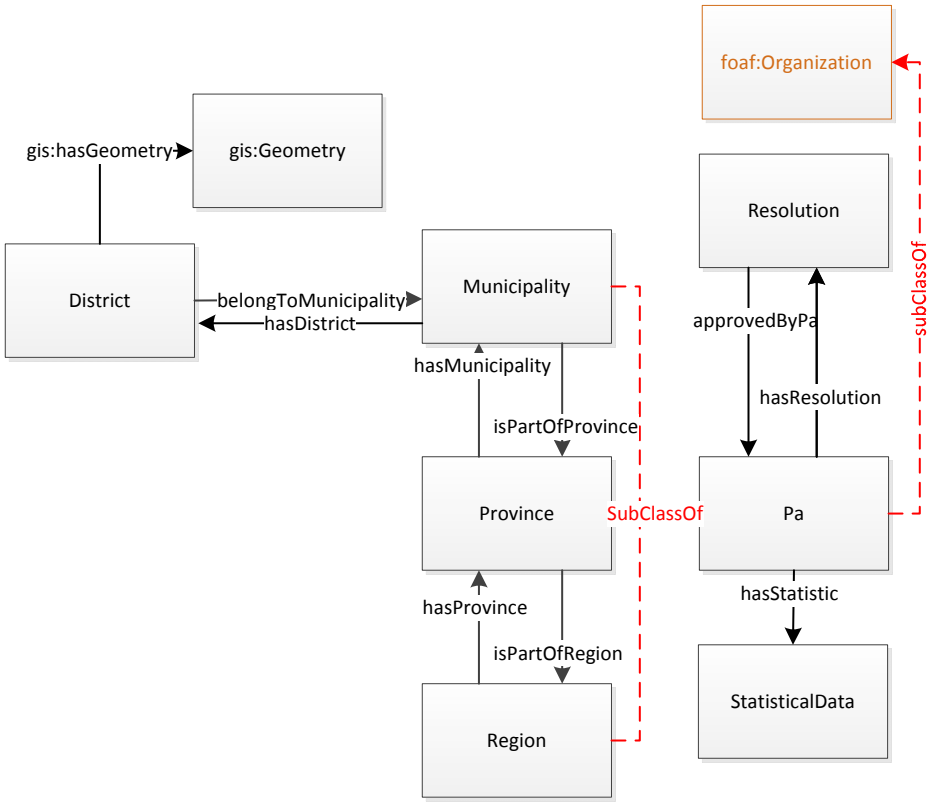
L'ontologia

Km4city dovrà permettere l'interconnessione, la memorizzazione e la successiva interrogazione, di molti dati provenienti da differenti fonti, quali i vari portali della regione toscana (MIIC, muoversi in toscana, osservatorio dei trasporti) o gli stessi Open Data e Linked Data messi a disposizione dai singoli comuni (principalmente Firenze). È evidente quindi che l'ontologia che si andrà a realizzare non sarà di piccole dimensioni, e quindi potrebbe essere utile vederla come costituita da varie macroclassi, e per la precisione, attualmente, sono state individuate le seguenti macroclassi:

1. Amministrazione: la prima macroclasse che è possibile individuare, le cui classi principali sono PA, Municipality, Province, Region, Resolution.
2. Stradario: formata dalle classi Road, Node, RoadElement, AdminidtrativeRoad, Milestone, StreetNumber, RoadLink, Junction, Entry, EntryRule e Maneuver.
3. Punti di Interesse: comprende tutti i servizi, le attività, che possono essere utili al cittadino e che quindi quest'ultimo può aver necessità di ricercare e di raggiungere. La classificazione dei singoli servizi e attività si baserà sulla classificazione precedentemente adottata dalla Regione Toscana. Sono inoltre inclusi in questa macroclasse le Digital Location e gli eventi programmati (dati Real Time) del comune di Firenze
4. Trasporto Pubblico Locale: attualmente disponiamo dell'accesso ai dati relativi agli orari programmati delle principali aziende TPL, al grafo ferroviario, e ai dati relativi al tempo reale dell'ATAF. Tale macroclasse è quindi formata dalle classi TPLLine, Ride, Route, AVMRecord, RouteSection, BusStopForeast, Lot, BusStop, RouteLink, TPLJunction.
5. Sensoristica: anche la macroclasse relativa ai dati provenienti da sensori è in via di sviluppo. Attualmente sono stati integrati nell'ontologia i dati raccolti da vari sensori installati lungo alcune strade di Firenze e dintorni, e quelli relativi ai posti liberi nei principali parcheggi dell'intera regione; è già presente anche la parte relativa agli eventi/emergenze, dove però attualmente i dati raccolti sono in numero molto limitato e oltretutto vecchi di mesi. Oltre a questi dati, sono stati inseriti in questa macroclasse anche quelli relativi alle previsioni meteo del Lamma.
6. Temporale: macroclasse che punta all'inserimento di concetti legati al tempo (istanti di tempo e intervalli di tempo) all'interno dell'ontologia, in modo da poter associare una linea temporale agli avvenimenti registrati e poter riuscire a fare previsioni.
7. Metadati: macroclasse di triple associate al context di ciascun dataset; tali triple raccolgono le informazioni relative a licenza del dataset, se il processo di ingestion è automatizzato completamente, il formato della risorsa, una breve descrizione della risorsa ed altre info sempre legate alla risorsa stessa e al suo processo di ingestion.

Analizziamo adesso una ad una le differenti macroclassi individuate.

La prima macroclasse Amministrazione si compone come riportato nella seguente figura.



La classe principale è km4c:PA, che è stata definita come sottoclasse della foaf:Organization, collegamento che ci aiuta ad assegnare un più chiaro significato alla nostra classe. Le 3 sottoclassi di km4c:PA sono automaticamente definite in base alla restrizione sulle ObjectProperties (rappresentate in figura con linee continue). Ad esempio, la classe km4c:Region è definita come restrizione della classe km4c:PA sulla ObjectProperty km4c:hasProvince, in modo tale che solo le PA che posseggono provincie, possono essere classificate come regione. Altro esempio: per definire gli elementi di PA che compongono la classe Municipality è stata invece utilizzata una restrizione sull'ObjectProperty km4c:isPartOfProvince, quindi se una PA non è assegnata ad una provincia, non può essere considerata un comune.

Nella costituzione della gerarchia all'interno della classe PA, per ciascun passaggio, sono state definite coppie di ObjectProperties inverse: km4c:hasProvince e km4c:isPartOfRegion, km4c:hasMunicipality e km4c:isPartOfProvince.

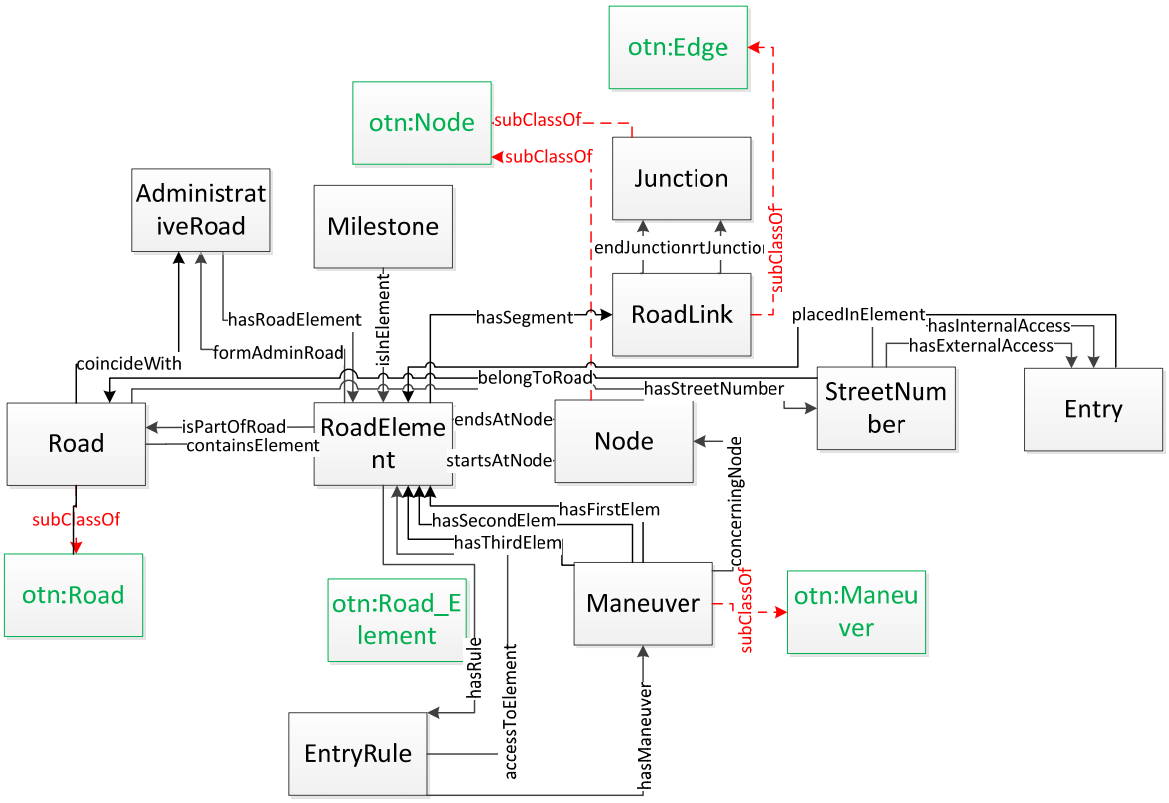
Collegata alla classe km4c:PA attraverso l'ObjectProperty km4c:hasResolution, troviamo la classe km4c:Resolution, le cui istanze sono rappresentate dalle delibere approvate appunto dalle varie PA. km4c:hasResolution ha la sua ObjectProperty inversa, cioè km4c:approvedByPa.

La Classe km4c:District rappresenta invece i quartieri in cui una città può essere amministrativamente suddivisa; tale classe è connessa alla classe km4c:Municipality attraverso una coppia di ObjectProperty inverse: km4c:belongToMunicipality e km4c:hasDistrict.

L'ultima classe presente in questa macroclasse è km4c:StatisticalData: vista la grande quantità di dati statistici legati sia ai vari comuni, alla regione, ma anche alle singole strade, è stata inserita questa classe,

condivisa sia dalla macroclasse Amministrazione che dallo Stradario. Come vedremo poi anche nella descrizione della prossima macro classe, km4c:StatisticalData è collegata sia a km4c:Pa che a km4c:Road, attraverso l'ObjectProperty km4c:hasStatistic.

La macroclasse Stradario è invece riportata nella figura seguente.



È evidente che tale macroclasse è la più complessa, in quanto rappresenta tutto il grafo strade, i vari numeri civici, gli accessi a questi ultimi, ma anche le regole di accesso alle varie strade e le manovre consentite.

La classe principale, al centro della macroclasse Stradario, è il km4c:RoadElement, definito come sottoclasse del corrispondente elemento all'interno dell'ontologia OTN, cioè Road_Element. Ciascun km4c:RoadElement è delimitato da un nodo iniziale ed un nodo finale, individuabili tramite le ObjectProperties km4c:startsAtNode ed km4c:endsAtNode, che collegano la classe in oggetto alla classe km4c:Node. Nella definizione della classe km4c:RoadElement sono state specificate alcune restrizioni, legate appunto alla classe km4c:Node: un elemento stradale deve avere definite entrambe le ObjectProperty km4c:startsAtNode e km4c:endsAtNode, entrambe con una cardinalità precisamente pari ad 1. Uno o più elementi stradali formano una strada: la classe Road è infatti definita come sottoclasse della corrispondente classe nella OTN, cioè l'omonima Road, e con restrizione sulla cardinalità dell'ObjectProperty km4c:containsElement (che ha proprietà inversa km4c:isPartOfRoad), che deve essere minimo pari ad 1, cioè non può esistere una strada che non contiene almeno un elemento stradale. Anche la classe km4c:AdministrativeRoad, che rappresenta le estese amministrative, è collegata alla classe km4c:RoadElement tramite le due ObjectProperty inverse km4c:hasRoadElement e km4c:formAdminRoad, mentre è collegata con una sola ObjectProperty alla classe km4c:Road (km4c:coincideWith). Chiariamo bene la relazione che c'è tra km4c:Road, km4c:AdministrativeRoad e km4c:RoadElement: un'istanza di

Road può essere collegata a più istanze di km4c:AdministrativeRoad (ad esempio se una strada attraversa il confine tra due provincie), ma vale anche il contrario (ad esempio quando una strada provinciale attraversa un centro città e assume differenti nomi), esiste cioè tra le due classi una relazione N:M. Entrambe però hanno una relazione 1:N con i km4c:RoadElement, cioè sia una istanza di Road che di km4c:AdministrativeRoad, sono collegate a più istanze di km4c:RoadElement.

Su ciascun elemento stradale è possibile definire delle restrizioni di accesso, individuate dalla classe km4c:EntryRule, che è connessa alla classe km4c:RoadElement attraverso 2 ObjectProperties inverse, cioè km4c:hasRule e km4c:accessToElement. La classe km4c:EntryRule è definita con una restrizione sulla minima cardinalità della ObjectProperty km4c:accessToElement (imposta pari a 0), che nella maggior parte dei casi ha associato un 1 solo elemento stradale, ma in alcuni casi eccezionali tale associazione viene a mancare. Le regole di accesso permettono di definire in modo univoco un permesso o limitazione di accesso sia sugli elementi stradali (per esempio a causa della presenza di una ZTL) come appena visto, ma anche sulle manovre; per questo motivo la classe Maneuver e la classe km4c:EntryRule sono collegate dall'ObjectProperty km4c:hasManeuver. Per manovra si intendono principalmente le manovre di svolta obbligatorie, prioritarie o proibite, che sono descritte indicando l'ordine degli elementi stradali che interessano. Analizzando i dati provenienti dal Grafo Strade è stato verificato che solo in rari casi le manovre interessano 3 differenti elementi stradali e quindi, per rappresentare la relazione che intercorre tra le classi km4c:Maneuver ed km4c:RoadElement sono state definite 3 ObjectProperties: km4c:hasFirstElem, km4c:hasSecondElem ed km4c:hasThirdElem, oltre all'ObjectProperty che lega una manovra alla giunzione che interessa, cioè km4c:concerningNode (perché una manovra avviene sempre in prossimità di un nodo). Nella definizione della classe km4c:Maneuver sono state inoltre aggiunte restrizioni di cardinalità, fissata obbligatoriamente ad 1 per km4c:hasFirstElem e km4c:hasSecondElem e cardinalità massima ad 1 per km4c:hasThirdElem, in quanto per le manovre che interessano solo 2 elementi stradali, questa ObjectProperty non è definita.

Come precedentemente accennato, ciascun elemento stradale è delimitato da due nodi (o giunzioni), uno di inizio e uno di fine. È stata quindi definita la classe km4c:Node, sottoclasse della omonima classe Node appartenente all'ontologia OTN. La classe km4c:Node è stata definita con una restrizione sulle DataProperty geo:lat e geo:long, due proprietà ereditate dal fatto che la classe è stata definita come sottoclasse della geo:SpatialThing appartenente all'ontologia Geo wgs84: ciascun nodo infatti può essere associato a soltanto una coppia di coordinate nello spazio, e non può esistere un nodo senza tali valori.

La classe Milestone rappresenta i cippi chilometrici che sono posizionati lungo le estese amministrative, cioè degli elementi puntuali che identificano il valore della chilometrica in quel preciso punto, ovvero la progressiva del tracciato rispetto al punto d'inizio. Un milestone può essere associato ad un'unica km4c:AdministrativeRoad, ed è quindi stata definita una restrizione sulla cardinalità associata alla ObjectProperty km4c:isInElement, imposta esattamente pari ad 1. Anche la classe km4c:Milestone è definita come sottoclasse della geo:SpatialThing, ma stavolta la presenza delle coordinate non è obbligatoria (restrizione sulla cardinalità massima che deve essere pari ad uno, ma che comunque non esclude la possibile mancanza del valore).

Il numero civico serve a definire un indirizzo, ed è sempre logicamente relazionato ad almeno un accesso, infatti ad ogni civico corrisponde sempre un solo accesso esterno, che può essere diretto o indiretto; a volte può essere presente anche un accesso interno. Vedendo tale relazione dal punto di vista dell'accesso, invece si può invece affermare che ciascuno di questi è connesso logicamente ad almeno un numero civico. Sono quindi state definite le classi km4c:StreetNumber ed km4c:Entry.

Con i dati posseduti è possibile collegare la classe km4c: StreetNumber sia alla classe km4c:RoadElement che alla classe km4c:Road, rispettivamente attraverso le ObjectProperties km4c:placedInElement e km4c:belongToRoad. Tale informazione risulta effettivamente ridondante e nel caso si ritenga opportuno potrebbe essere eliminato il collegamento alla classe km4c:Road a favore di quello verso km4c:RoadElement, più facilmente ricavabile dai dati; è infatti per questo motivo che per l'ObjectProperty km4c:belongToRoad è stata definita anche l'inversa km4c:hasStreetNumber.

All'interno dell'ontologia quindi, la classe km4c:StreetNumber è definita come avente una restrizione sulla cardinalità della ObjectProperty km4c:belongToRoad, che deve essere pari ad 1.

Anche la classe km4c:Entry può essere collegata sia al numero civico che all'elemento stradale in cui si trova. La relazione tra km4c:Entry e km4c:StreetNumber, è definita da 2 ObjectProperties, km4c:hasInternalAccess e km4c:hasExternalAccess, sulle quali sono state definite delle restrizioni di cardinalità, in quanto, come già detto in precedenza, un numero civico avrà sempre un solo accesso esterno, mentre potrebbe avere anche un accesso interno (quest'ultima restrizione è infatti stata definita impostando la massima cardinalità pari ad 1, cioè sono ammessi i valori 0 ed 1). Anche la classe km4c:Entry è definita come sottoclasse della geo:SpatialThing, ed è quindi possibile associare massimo una coppia di coordinate geo:lat e geo:long a ciascuna istanza (restrizione sulla cardinalità massima delle due DataProperty della Geo ontology, imposta a 1, così che possano assumere o 1 valore, o nessuno).

La macroclasse Stradario è collegata alla macroclasse Amministrazione attraverso 2 differenti ObjectProperties, km4c:ownerAuthority e km4c:managingAuthority, che rappresentano chiaramente come suggerisce il nome, rispettivamente la pubblica amministrazione che possiede l'estesa amministrativa, o la pubblica amministrazione che gestisce l'elemento stradale. Restano così fuori dalla rappresentazione, soltanto le strade di proprietà privata, per le quali ancora non è stata individuata la miglior rappresentazione all'interno dell'ontologia.

Da un punto di vista cartografico però, ciascun elemento stradale non è una retta, ma una spezzata, che seguirà l'effettivo andamento della strada. Per poter rappresentare questa situazione, sono state quindi aggiunte le classi km4c:RoadLink e km4c:Junction: grazie all'interpretazione dei file KMZ, abbiamo recuperato la serie di coordinate che definiscono ciascun road element, e ciascuno di questi punti, verrà inserito nell'ontologia come una istanza di Junction (definita come sottoclasse di geo:SpatialThing, con obbligatoriamente un'unica coppia di coordinate). Ciascun segmentino tra due Junction è invece un'istanza della classe km4c:RoadLink, che è definita con una restrizione sulle ObjectProperty km4c:endJunction e km4c:startJunction (entrambe devono obbligatoriamente avere cardinalità pari ad 1), che connettono le due classi.

Per la terza macro classe, cioè Punti di Interesse, è stata implementata una revisione della suddivisione in classi e sub-Classi strettamente connessa con i codici ATECO, cioè il codice ISTAT di classificazione delle attività economiche; la suddivisione presente nelle precedenti versioni dell'ontologia, era infatti basata sulla classificazione a livello regionale delle aziende. Purtroppo i vecchi servizi già ingeriti, i quali erano sprovvisti di codice ATECO, non avranno un valore per la DataProperty km4c:codiceATECO, ma in base alle informazioni possedute, saranno comunque riclassificati all'interno della nuova gerarchia creata.

La gerarchia creata non presenta una totale corrispondenza con l'intero elenco dei codici ATECO, ma è stata creata in modo da fornire maggiori dettagli delle sottoclassi di maggiore interesse per i nostri scopi (fornitori di servizi, vendite al dettaglio etc.), e restare più generica, nelle aree di minore interesse (vendita all'ingrosso, industria etc.).

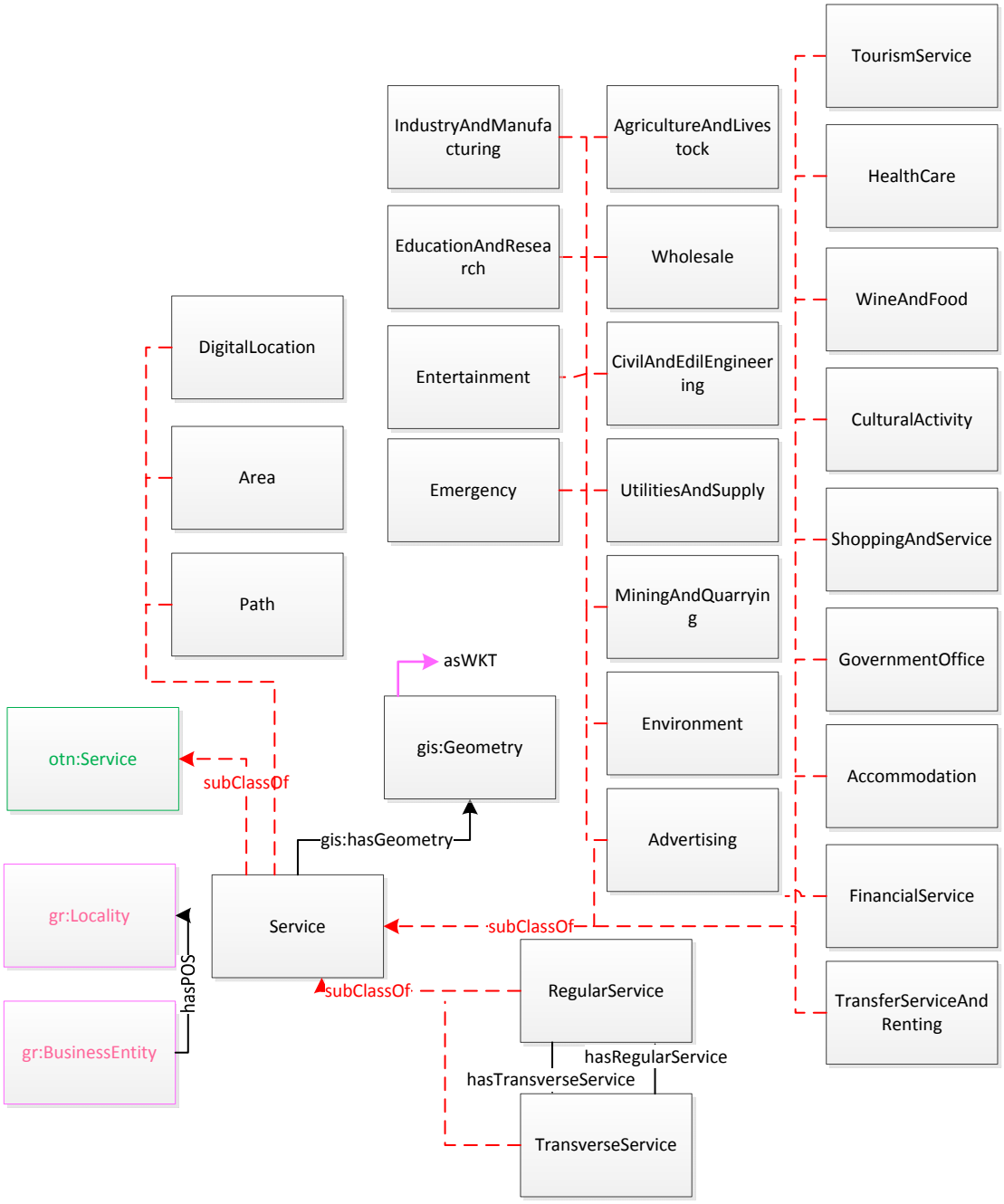
Attualmente quindi sono state individuate le classi Accommodation, GovernmentOffice, TourismService, TransferServiceAndRenting, CulturalActivity, FinantialService, ShoppingAndService, HealtCare, EducationAndResearch, Entertainment, Emergency, WineAndFood, IndustryAndManufacturing, AgricultureAndLivestock, UtilitiesAndSupply, CivilAndEdilEngineering, Wholesale, Advertising, MiningAndQuarrying and Environment. Ciascuna di esse presenta una serie di sottoclassi che definiscono quali tipi di servizi appartengono a tale classe: ad esempio per la classe Accommodation, le sottoclassi definite sono 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 e Summer_camp. Attualmente all'interno di Km4City sono state definite 512 sottoclassi di servizi cosi divise:

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

La classe Service presenta anche una coppia di sottoclassi che permettono di identificare più rapidamente i servizi non puntuali; queste due classi sono km4c:Path e km4c:Area, che raccolgono rispettivamente i servizi lineari rappresentati da una spezzata sulla mappa, la prima, mentre la seconda raccoglie i servizi poligonali, cioè rappresentati da un'area sulla mappa. È chiaro quindi che, ciascun servizio non puntuale, dovrà essere anche istanza di una di queste due classi.

È stata inoltre definita un ulteriore suddivisione dei servizi, cioè i servizi regolari (km4c:RegularService) ed i servizi trasversali (km4c:TransversalService); tale suddivisione si è resa necessaria in quanto con l'inserimento di nuovi servizi è stata scoperta la possibilità che i servizi siano tra di loro connessi: ad esempio si pensi ad un ristorante che offre anche servizio di WiFi con libero accesso. Per gestire questa situazione è quindi necessario collegare tra di loro i due servizi, ma dare anche la possibilità a chi ha

interesse di individuare solo le Wi-Fi, di trovarla come servizio. È quindi evidente che ci sono due tipi di servizi, i servizi primari o regolari ed i servizi secondari che possono essere collegati anche ad un servizio regolare, tramite le apposite ObjectProperties inverse km4c:hasRegularService e km4c:hasTransversalService.



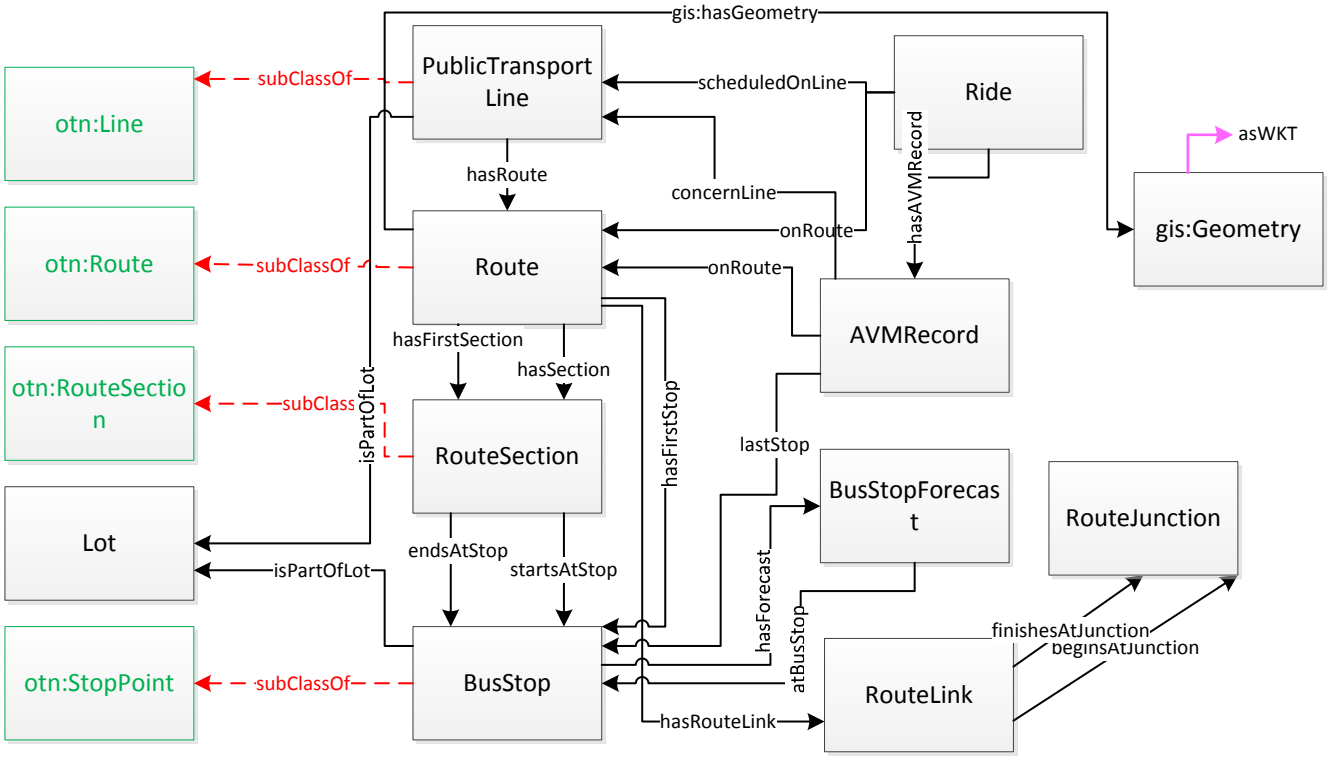
Un ulteriore parte della macroclasse Point of Interest è rappresentata dall'integrazione delle Digital Location del comune di Firenze. Le Digital Location sono dati raccolti e pubblicati dal comune di Firenze e sotto forma di Linked Data, rappresentanti location ritenute interessanti per la città stessa, come ad esempio percorsi di Jogging, aree verdi e giardini, monumenti o palazzi storici, etc. per un totale di circa 3200 differenti Digital Location. Tali elementi sono stati ri-mappati sulla classe km4c:Service e, laddove alcuni dei servizi siano già precedentemente stati inseriti, grazie ad una fase di riconciliazione sarà possibile collegare la Service-digital location ed il servizio che fanno riferimento allo stesso edificio, servizio,

monumento, grazie ad una tripla owl:sameAs, in modo da rendere evidente che rappresentano lo stesso elemento.

L'introduzione della Ontologia geoSPARQL, ha inoltre permesso di associare ai servizi una forma geometrica: un servizio infatti può essere individuato da un punto (nel caso di un negozio, un monumento, etc.), oppure da un percorso (una linea spezzata che connette vari punti di interesse, ad esempio un percorso turistico) oppure da un area (un poligono che ad esempio corrisponde ad un giardino pubblico o un parco).

Un'altra classe che è stata aggiunta a questa Macroclasse è la classe km4c:Event, la quale eredita la maggior parte delle sue proprietà dalla classe schema:Event, ma non solo. Tale classe è connessa tramite la ObjectProperty schema:location, alla classe km4c:Service, la quale in questo caso fa la parte del luogo nel quale l'evento si svolgerà.

La quarta macroclasse TPL non è ancora completa, presenta infatti soltanto dati e classi relative alle linee dei bus metropolitani di Firenze, tranvia e rete ferroviaria; la distribuzione delle linee urbane delle altre zone e delle linee extraurbane degli autobus, sono facilmente adattabili sul modello realizzato per i dati metropolitani fiorentini.



Analizziamo prima la parte relativa al trasporto metropolitano fiorentino che comprende trasporto su gomma e tramvia.

Ciascun lotto TPL, rappresentato dalla classe km4c:Lot, è composto da un certo numero di Linee di autobus o tranvia (classe km4c:PublicTransportLine), e tale relazione è rappresentata dalla ObjectProperty km4c:isPartOfLot, che collega ciascuna istanza di km4c:PublicTransportLine alla corrispettiva istanza di km4c:Lot. La classe km4c:PublicTransportLine è definita come sottoclasse della otn:Line. Ciascuna linea prevede almeno una corsa, individuate attraverso un codice fornito dall'azienda TPL; la classe

km4c:PublicTransportLine è stata infatti collegata alla classe Ride attraverso l'ObjectProperty km4c:scheduledOnLine, sulla quale è definita anche una limitazione di cardinalità pari esattamente ad 1, perché a ciascuna corsa può essere associata una sola linea.

Ciascuna corsa segue almeno un percorso (non è stato verificato su tutti i dati se la corrispondenza è 1:1, non appena sarà verificato, eventualmente sarà inserita una restrizione sulla cardinalità per l'ObjectProperty che collega le due classi, cioè km4c:onRoute), e i percorsi possono essere in numero variabile anche se riferiti ad una stessa linea: nella maggior parte dei casi sono 2, cioè percorso in avanti e percorso indietro, ma a volte arrivano ad essere anche 3 o 4, in base ad eventuali prolungamenti di percorsi o deviazioni magari effettuate solo in particolari orari. L'ObjectProperty km4c:onRoute è utilizzata anche per collegare la classe km4c:Ride, rappresentante le singole corse definite dall'operatore TPL su appunto un certo percorso. Tramite l'ObjectProperty gis:hasGeometry, le istanze della classe km4c:Route possono invece essere collegate all'istanza della classe gis:Geometry che contiene la line string rappresentante il percorso effettivo di quella corsa.

Ciascun percorso è considerato come costituito da una serie di tratti stradali delimitati da fermate successive: per modellare questa situazione, si è scelto di definire due ObjectProperty che collegano le classi km4c:Route e km4c:RouteSection, km4c:hasFirstSection e km4c:hasSection, in quanto, da un punto di vista cartografico, volendo rappresentare il percorso che segue un certo autobus, definito quale sia il primo segmento e conosciuta la fermata di partenza, si è in grado di recuperare tutti gli altri segmenti che costituiscono il percorso e partendo dalla seconda fermata individuata come fermata differente dalla prima, ma contenuta nel primo segmento, siamo in grado di ricostruire l'esatta sequenza delle fermate e quindi dei segmenti, che costituiscono l'intero percorso. A tale scopo è stata definita quindi anche l'ObjectProperty km4c:hasFirstStop, che collega le classi km4c:Route e km4c:BusStop.

Applicando lo stesso tipo di modellazione utilizzato per gli elementi stradali, sono state definite due ObjectProperty km4c:endsAtStop e km4c:startsAtStop, per collegare ogni istanza di km4c:RouteSection a 2 istanze della classe km4c:BusStop, classe a sua volta definita come sottoclasse della otn:StopPoint. Ciascuna fermata è inoltre connessa alla classe km4c:Lot, attraverso l'ObjectProperty km4c:isPartOfLot, relazione 1:N in quanto esistono fermate condivise da linee urbane e extraurbane che quindi appartengono a due differenti lotti.

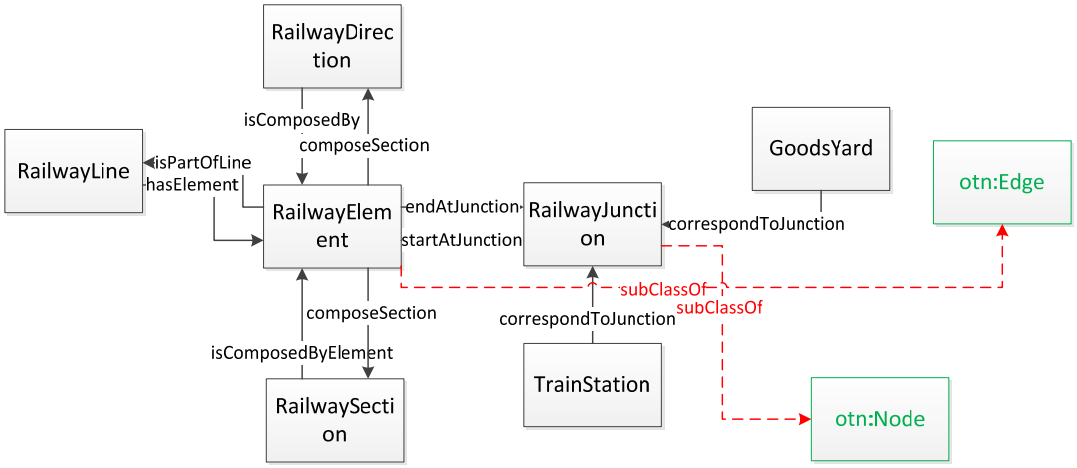
Disponendo inoltre delle coordinate di ciascuna fermata, la classe km4c:BusStop è stata definita come sottoclasse della geo:SpatialThing, ed è stata inoltre definita una cardinalità pari ad 1 per le due DataProperty geo:lat e geo:long.

Volendo poi da un punto di vista cartografico rappresentare il percorso di un autobus, quindi una istanza di km4c:Route, abbiamo bisogno di rappresentare la spezzata che compone ciascun tratto di strada attraversato dal mezzo stesso e per far ciò, è stata riutilizzata la modellazione precedentemente usata per gli elementi stradali: ciascun percorso lo possiamo vedere come un insieme di piccoli segmenti, ciascuno delimitato da due giunzioni: sono quindi state definite le classi km4c:RouteLink e km4c:RouteJunction, e le ObjectProperty km4c:beginsAtJunction e km4c:finishesAtJunction. La classe km4c:RouteLink è stata definita come restrizione di cardinalità su entrambe le ObjectProperty appena citate, imponendo che sia sempre pari ad 1. La classe km4c:Route è invece stata collegata alla classe km4c:RouteLink attraverso la objectProperty km4c:hasRouteLink.

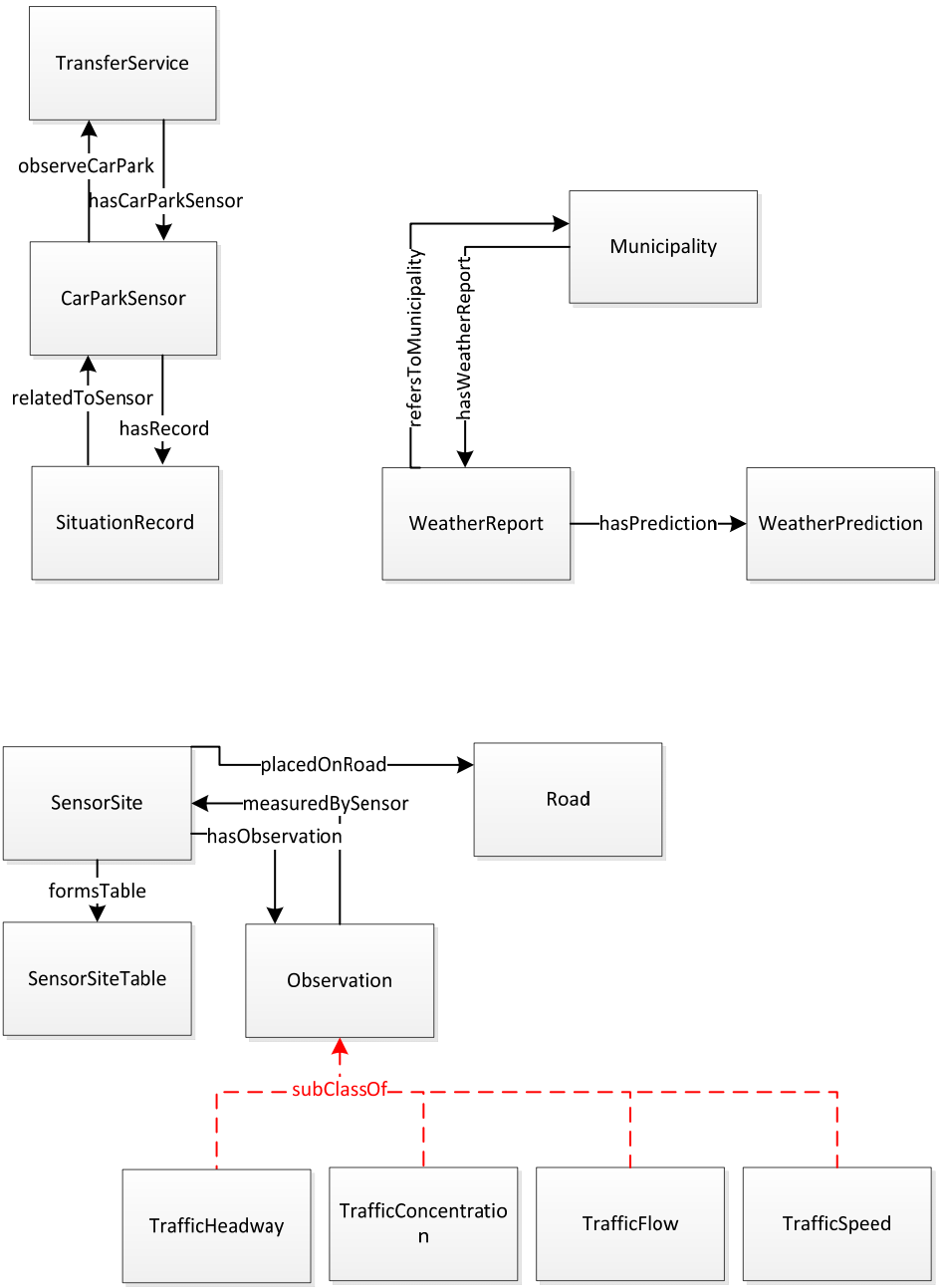
La classe km4c:BusStop è inoltre collegata alla classe km4c:Road della macroclasse Grafo Strade, tramite l'ObjectProperty km4c:isInRoad, grazie alla quale risulta più immediata una localizzazione a livello di comune delle singole fermate.

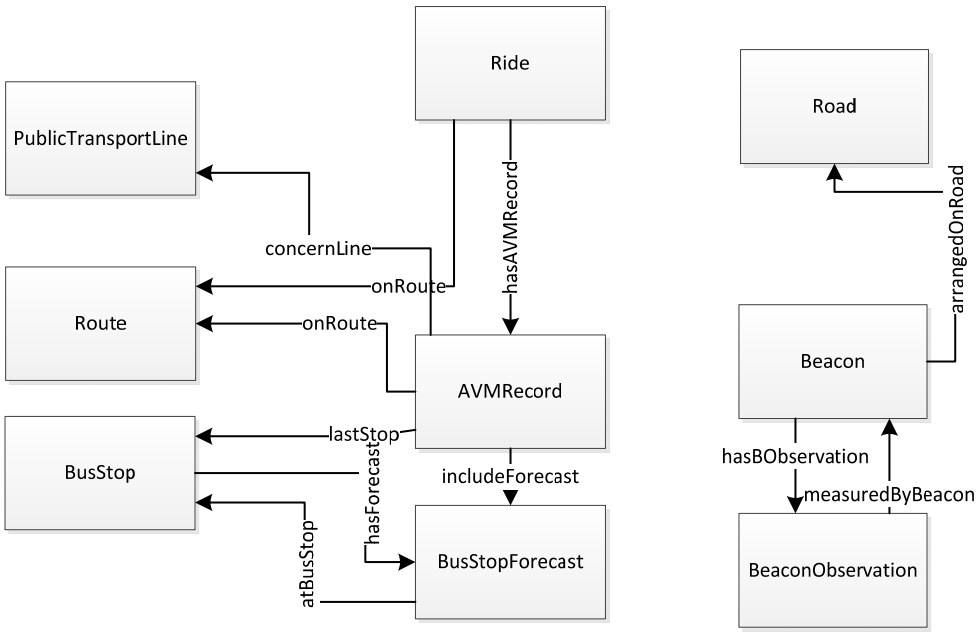
La parte relativa al Grafo Ferrviario è formata principalmente dalla classe km4c:RailwayElement (definita come sottoclasse della OTN:Edge), le cui istanze rappresentano un singolo elemento ferroviario; ciascun elemento ferroviario può comporre una direttrice ferroviaria, cioè una linea ferroviaria avente particolari caratteristiche di importanza per il volume dei traffici e le relazioni di trasporto, che su di essa si svolgono e che congiunge tra loro centri o nodi principali della rete ferroviaria, oppure una sezione ferroviaria (tratto di linea in cui si può trovare solo un treno per volta che solitamente è preceduto da un segnale "di protezione" o "di blocco") oppure una linea ferroviaria (cioè l'infrastruttura atta a far viaggiare treni o altri convogli ferroviari tra due località di servizio). Inoltre ciascun elemento ferroviario, inizia e termina in una giunzione ferroviaria, cioè un istanza della classe km4c:RailwayJunction, definita come sottoclasse della OTN:Node. Si hanno inoltre le classi km4c:TrainStation che rappresenta appunto una stazione ferroviaria, e km4c:GoodsYard cioè uno scalo merci, e solitamente entrambi corrispondono ad un'unica istanza di km4c:RailwayJunction.

Anche la classe km4c:RailwayElement è connessa alla classe gis:Geometry, in quanto ciascun elemento ferroviario può essere visto come una line string che rappresenta il vero andamento della linea ferroviaria in quel tratto.



La macro classe Sensoristica non è stata ancora completata, ma per ora sono state inserite le parti riportate in figura e relative, rispettivamente ai sensori presenti nei parcheggi, alle previsioni del tempo, ai sensori installati lungo le strade, agli apparati AVM e ai Beacons.


TransferService
observeCarPark
hasCarParkSensor
CarParkSensor
relatedToSensor
hasRecord
SituationRecord
Municipality
refersToMunicipality
hasWeatherReport
WeatherReport
hasPrediction
WeatherPrediction
SensorSite
placedOnRoad
measuredBySensor
hasObservation
Road
formsTable
SensorSiteTable
Observation
subClassOf
TrafficHeadway
TrafficConcentration
TrafficFlow
TrafficSpeed



La prima parte, cioè quella focalizzata sui dati in tempo reale relativi ai parcheggi. La classe km4c:TransferService infatti è collegata alla classe km4c:CarParkSensor, che rappresenta il sensore installato in un certo parcheggio e al quale saranno collegati le istanze della classe km4c:SituationRecord, che rappresentano lo stato di un certo parcheggio ad un certo istante; il primo collegamento, cioè quello tra la classe km4c:TransferService e km4c:CarParkSensor, è realizzato attraverso due ObjectProperty invrse, cioè km4c:observeCarPark e km4c:hasCarParkSensor km4c:, mentre il collegamento tra la classe km4c:CarParkSensor e km4c:SituationRecord, è realizzato tramite le ObjectProperty inverse, km4c:relatedToSensor e km4c:hasRecord. La classe km4c:SituationRecord permette di memorizzare le informazioni relative a quanti posti liberi ed occupati ci sono in un determinato istante (anch'esso memorizzato) nei vari parcheggi della regione toscana.

La seconda parte dei dati ricevuti in tempo reale, riguarda invece le previsioni meteorologiche, disponibili per i differenti comuni (e quindi legate alla classe km4c:Municipality) grazie al LAMMA. Il consorzio aggiorna i report di ciascun comune 1 o 2 volte al giorno ed ogni report al suo interno riporta le previsioni di 5 gg suddivise in fasce, che presentano una maggior precisione (ed un maggior numero) per i giorni più vicini fino ad arrivare ad un'unica previsione giornaliera per il 4 e 5 giorno. Tale situazione è stata infatti rappresentata dalla classe km4c:WeatherReport connessa tramite l'ObjectProperty km4c:hasPrediction alla classe km4c:WeatherPrediction. Il comune invece è connesso ad un report tramite 2 ObjectProperty inverse: km4c:refersToMunicipality e km4c:hasWeatherReport.

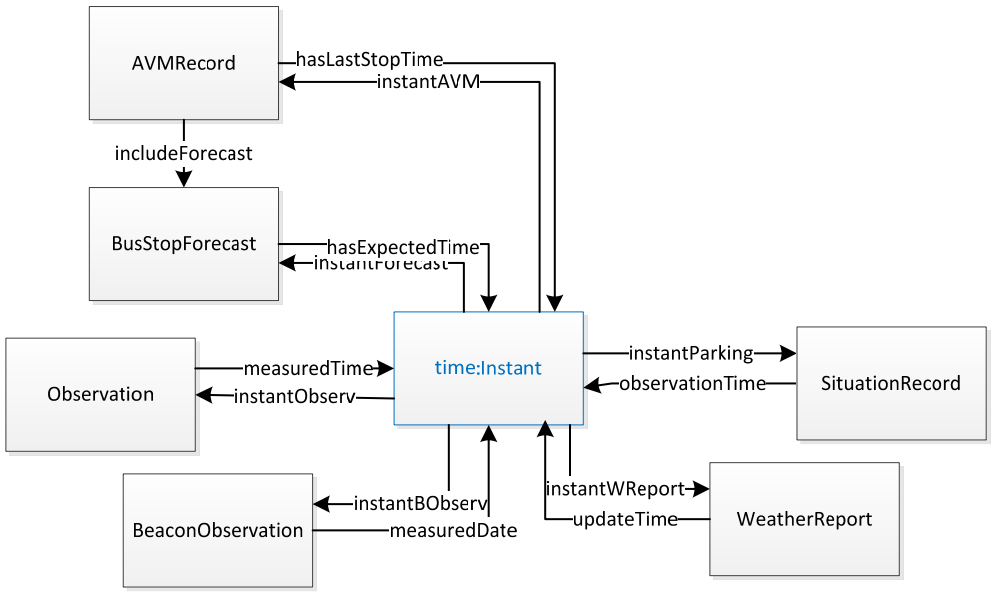
La terza parte dei dati Real Time riguarda invece i sensori disposti lungo le strade della regione, i quali permettono di fare differenti rilevazione in relazione alla situazione del traffico. Grazie alle informazioni aggiuntive ricevute recentemente dall'Osservatorio dei Trasporti, è possibile geo localizzare tutti i sensori con precisione. I sensori sono suddivisi in gruppi, ciascun gruppo è rappresentato dalla classe km4c:SensorSiteTable e ciascun sensore, cioè ciascuna istanza della classe km4c:SensorSite, si connette al proprio gruppo tramite la ObjectProperty km4c:formsTable e, come detto in precedenza ciascuna istanza di quest'ultima classe può esser soltanto collegata alla classe Road (attraverso la ObjectProperty km4c:installedOnRoad). Ciascun sensore produce delle osservazioni, le quali sono rappresentate dalla classe km4c:Observation; tali osservazioni possono essere di 4 tipi cioè relative alla velocità media (sottoclasse km4c:TrafficSpeed), oppure relative al flusso di auto che passa davanti al sensore (sottoclasse

TrafficFlow), alla concentrazione di traffico (sottoclasse km4c:TrafficConcentration) e infine relativi alla densità di traffico (sottoclasse km4c:TrafficHeadway). La classe km4c:Observation la classe Sensor sono connesse tramite una coppia di ObjectProeprty inverse, cioè km4c:hasObservation e km4c:measuredBySensor.

La quarta parte della macroclasse RealTime, riguarda i sistemi AVM installati su buona parte dei mezzi ATAF, ed è rappresentata principalmente da due classi, km4c:AVMRecord e km4c:BusStopForecast: la prima classe rappresenta il record inviato dal sistema AVM, all'interno del quale, oltre informazioni relative all'ultima fermata effettuata (rappresentata tramite l'ObjectProperty km4c:lastStop che collega km4c:AVMrecord a km4c:BusStop), alle coordinate GPS di localizzazione del mezzo, all'identificativo del mezzo e della linea, troviamo anche l'elenco delle prossime fermate con l'orario di passaggio previsto; tale elenco risulta di lunghezza variabile e rappresenta le istanze della classe km4c:BusStopForecast. Quest'ultima classe è stata collegata alla classe km4c:BusStop attraverso l'ObjectProperty km4c:atBusStop in modo tale da poter recuperare anche l'elenco delle eventuali linee previste in arrivo ad una certa fermata (la classe km4c:AVMRecord è stata infatti collegata anche alla classe Line tramite l'ObjectProperty km4c:concernLine).

La quinta parte dei dati Real Time, come precedentemente anticipato, riguarda i beacons, per i quali sono state create due classi, la Beacon, che rappresenta il singolo dispositivo installato nelle città, e la classe BObservation, che rappresenta invece una singola osservazione riportata da ciascun elemento funzionante; le due classi sono connesse tramite coppia di ObjectProperty inverse km4c:hasBObservation e km4c:measuredByBeacon. Ciascun beacon è univocamente identificato tramite un codice identificativo, e da altri tre codici, l'uuid (Universally Unique Identifier, che contiene 32 cifre esadecimali, divise in 5 gruppi separati dal trattino) che identifica univocamente il proprietario del beacon (quindi se un negozio possiede 10 beacons, tutti e 10 avranno lo stesso uuid), il major ed il minor che invece identificano rispettivamente un gruppo di beacons e il numero di ciascun elemento all'interno del gruppo identificato con uno stesso major; saranno poi disponibili delle coordinate che identificano il luogo in cui i beacons sono posizionati. Le informazioni che invece un beacon è in grado di restituire sono la potenza del segnale, le coordinate GPS di dove si è messo in contatto con un utente e data ed ora del contatto.

Infine, l'ultima macroclasse, cioè Temporale è stata soltanto "abbozzata" all'interno dell'ontologia, basandosi sulla Time ontology (http://www.w3.org/TR/owl-time/) e sull'esperienze fatta in altri progetti come OSIM. Si rende necessaria l'integrazione del concetto del tempo in quanto sarà di fondamentale importanza riuscire a calcolare differenze tra istanti di tempo, e l'ontologia Time ci viene in aiuto in questo.



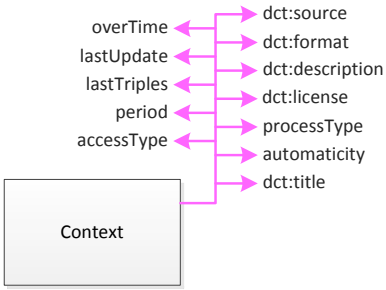
Si definisce un URI fittizio #instantForecast, #instantAVM, #instantParking, #instantWreport, #instantObserv da associare in seguito all'URI identificativo della risorsa a cui si riferisce il parametro temporale, cioè rispettivamente di km4c:BusStopForecast, di km4c:AVMRecord, di km4c:SituationRecord, di km4c:WheatherReport e infine di km4c:Observation.

l'URI fittizio #instantXXXX, sarà dato dalla concatenazione di due stringhe: ad esempio, nel caso delle istanze di km4c:BusStopForecast, si andranno a concatenare il codice fermata (che ci permette di individuarle univocamente) e l'istante temporale nel formato desiderato. È necessario creare un URI fittizio che leghi istante temporale a ciascuna risorsa, per non creare ambiguità, in quanto potrebbero essere presenti istanti temporali identici associati a differenti risorse (anche se il formato con cui è espresso l'istante temporale ha una scala piuttosto fine).

L'ontologia Time permette di definire istanti come informazioni temporali puntuali, e di usarli come estremi per la definizione di intervalli e durate, caratteristica molto utili per aumentare un po' l'espressività.

Sono state inoltre definite delle coppie di ObjectProperties per ciascuna classe che necessita di esser collegata alla classe time:Instant: tra la classe km4c:SituationRecord e time:Instant sono state definite le ObjectProperties inverse km4c:instantParking e km4c:observationTime, tra le classi km4c:WeatherReport e km4c:Instant, sono state definite le ObjectProperties inverse km4c:instantWReport e km4c:updateTime, tra km4c:Observation e time:Instant si hanno le ObjectProperties inverse km4c:measuredTime e km4c:instantObserv, tra km4c:BusStopForecast e time:Instant abbiamo km4c:hasExpectedTime e km4c:instantForecast; tra km4c:AVMRecord e time:Instant, si hanno le ObjectProperties inverse km4c:hasLastStopTime e km4c:instantAVM; infine tra km4c:BeaconObservation e time:Instant sono state definite le proprieta' inverse km4c:instantBObserv e km4c:measuredDate.

Il dominio di tutte le ObjectProperties con nome instantXXXX è definito da elementi di classe time:temporalEntity, in modo tale da poter espandere tali proprietà non solo ad istanti, ma anche a possibili intervalli di tempo.



La settima macroclasse, come già anticipato in precedenza, riguarda i metadati associati ai vari dataset. Sesame [http://rdf4j.org/] permette di definire all'interno dell'ontologia, dei Named Graph, che non sono altro che dei grafi a cui si va ad associare un nome, detto anche contesto. Il contesto è in pratica un ulteriore campo che permette di espandere il modello a triple in un modello a quadruple; Owlim [http://www.ontotext.com/products/ontotext-graphdb-owlim-new-2/] ci permette in fase di caricamento triple, di associare i vari contesti a differenti set di triple. In questa macroclasse abbiamo quindi definito delle DataProperty che ci permettano di memorizzare tutte le info utili legate ad un certo dataset, ad esempio: data di creazione, sorgente dati, formato del file originario, descrizione del dataset, licenza associata al dataset, tipo di processo di ingestion, quanto è automatizzato tutto il processo di ingestion, tipo di accesso al dataset, overtime, periodo di riferimento, parametri associati, data di aggiornamento, data di creazione triple.

DataProperties delle principali classi

All'interno dell'ontologia sono state definite alcune DataProperties per le quali c'era la certezza di non poter utilizzare dei valori equivalenti definiti su altre ontologie.

Analizziamo di seguito, per le principali classi in cui sono stati definiti, queste proprietà.

La DataProperty km4c:typeLabel è stata definita per tutte le classi esistenti all'interno dell'ontologia, e rappresenta il campo testo nel quale è salvato il tipo a cui appartiene l'istanza; grazie a tale proprietà è possibile attivare una ricerca di tipo fullText che comprenda anche il tipo di classe delle istanze.

Al'interno della classe km4c:PA sono state definite soltanto 3 DataProperties, cioè foaf:name che rappresenta il nome della pubblica amministrazione rappresentata dall'istanza, il suo identificatore univoco presente nel sistema regionale, cioè dct:identified, della DublinCore ontology, ed il dct:alternative in cui sarà memorizzato il codice comunale presente nel codice fiscale. Altre informazioni relative alla PA possono essere individuate attraverso il legame con la classe Service, dove appunto sono presenti più informazioni dettagliate che altrimenti risulterebbero ridondanti. La classe km4c:Resolution, le cui istanza come visto in precedenza sono le delibere, presenta DataProperties per la definizione dell'Id, cioè dct:identified, dell'anno di approvazione, cioè km4c:year e un'altra proprietà proveniente dall'ontologia DublinCore cioè dct:subject (l'oggetto della delibera).

Ciascuna istanza della classe km4c:Road è univocamente individuata utilizzando la DataProperty dct:identified dove viene memorizzato l'identificativo del toponimo per tutta la rete regionale, formato da 15 caratteri e definito secondo la seguente regola: RT seguito dal codice ISTAT del comune a cui appartiene il toponimo (6 caratteri), seguito da 5 caratteri che rappresentano il progressivo a partire dal valore dei

caratteri del codice ISTAT , e infine le lettere TO. La km4c:roadType invece, rappresenta la tipologia di toponimo stradale, cioè ad esempio:

- Chiasso, Corso, Largo, Località, Piazza, Piazzale, Piazzetta, Via, Viale, Vicolo, Viottolo, Viuzzo etc.

All'interno del grafo strade sono inoltre presenti 2 nomi per ciascun toponimo: il nome ed il nome esteso, cioè comprensivo anche del tipo di toponimo. Il nome senza tipo, sarà di tipo stringa ed è associato alla DataProperty km4c:roadName, mentre per il nome esteso, è stata definita un'altra dataProperty cioèkm4c:extendexName, ed infine, per tutti gli eventuali alias che si possono formare (come ad esempio per Via S. Marta un eventuale alternative potrebbe essere Via Santa Marta etc.) è utilizzata la dct:alternative, facendo in modo che possano essere più di uno i vari nomi alternativi, così da facilitare le successive riconciliazioni.

Per quanto riguarda la classe km4c:AdministrativeRoad, oltre alle dataProperty dct:alternative e km4c:adRoadName che rispettivamente contengono i possibili nomi alternativi della strada e il suo nome ufficiale, è stata definita la dataProperty km4c:amminClass rappresenta la classificazione amministrativa, cioè se si tratta di

- strada statale
- strada regionale
- strada provinciale
- strada comunale
- strada militare
- strada privata

Infine sarà presente un identificativo all'interno del campo dct:identifier, il quale segue la seguente regola, definita a livello regionale: 15 caratteri, inizia con i caratteri RT seguito dal codice ISTAT del comune a cui appartiene l'estesa amministrativa (6 caratteri), seguito da 5 caratteri che rappresentano il progressivo a partire dal valore dei caratteri del codice ISTAT , e infine le lettere PA.

Le istanze della classe km4c:RoadElement sono univocamente individuate dalla DataProperty dct:identified, campo di 15 caratteri formato come segue: i caratteri RT seguiti da 6 caratteri per il codice ISTAT del comune di appartenenza, seguiti da 5 caratteri che rappresentano il progressivo a partire dal codice ISTAT, ed infine i caratteri ES. Anche la DataProperty km4c:elementType è stata definita per la classe km4c:RoadElement, e può assumere i seguenti valori:

- di tronco carreggiata
- di area a traffico strutturato
- di casello/barriera autostradale
- di passaggio a livello
- di piazza
- di rotatoria
- di incrocio
- di parcheggio strutturato
- di area a traffico non strutturato
- di parcheggio
- in area di pertinenza

- pedonale
- raccordo, bretella, svincolo
- controviale
- traghetto (elemento fittizio)

Nel grafo strade della regione toscana, associato a questa classe troviamo anche la sua classificazione funzionale, definita all'interno dell'ontologia come DataProperty km4c:elementClass, i cui possibili valori sono:

- autostrada
- extraurbana principale
- extraurbana secondaria
- urbana di scorrimento
- urbana di quartiere
- locale/vicinale/privata ad uso privato

La DataProperty km4c:composition invece, è stata definita per indicare la composizione della strada a cui appartiene l'elemento stradale e per la precisione, sono possibili i valori "carreggiata unica" oppure "carreggiate separate". Km4c:elemLocation rappresenta invece la sede dell'elemento, ed i valori che può assumere sono:

- a raso
- ponte
- rampa
- galleria
- ponte e galleria
- ponte e rampa
- galleria e rampa
- galleria, ponte e rampa

Per quanto riguarda la classe di larghezza dell'elemento stradale, si fa riferimento alla DataProperty km4c:width, che permette di individuare la fascia di appartenenza: "minore di 3,5 mt", "tra 3,5 e 7,0 mt", "maggiore di 7,0 mt" oppure "non rilevato"; per la lunghezza invece la DataProperty di riferimento è km4c:length, valore liberamente inseribile che non fa riferimento a nessuna fascia. Altro dato disponibile sul grafo strade, e di fondamentale importanza per la definizione delle manovre consentite, è la direzione di percorrenza dell'elemento stradale, indicato dalla DataProperty km4c:trafficDir che può assumere uno dei seguenti 4 valori:

- tratto stradale aperto in entrambe le direzioni (default)
- tratto stradale aperto nella direzione positiva (da nodo iniziale a nodo finale)
- tratto stradale chiuso in entrambe le direzioni
- tratto stradale aperto nella direzione negativa (da nodo finale a nodo iniziale)

La DataProperty km4c:operatingStatus invece serve per tener traccia dello stato d'esercizio dei differenti elementi stradali e può quindi assumere i valori "in esercizio", "in costruzione" oppure "in disuso". Infine è presente anche una DataProperty che tiene conto dei limiti di velocità su ogni elemento stradale, cioè km4c:speedLimit.

Nella classe StatisticalData sono state definite DataProperty che permettano di associare al valore effettivo (memorizzato in km4c:value) informazioni necessarie a mantenere intatto il suo significato, come ad esempio il campo dct:identifier, dct:description, dct:created e dct:subject.

Un nodo o giunzione è un punto di intersezione degli assi di due elementi stradali, ed è sempre un'entità puntuale, rappresentata in termini geometrici, da una coppia di coordinate. Le istanze della classe km4c:Node possono essere univocamente individuate grazie al dct:identified, costituito anch'esso, come i precedenti codici, da 15 caratteri, secondo le seguenti regole: i primi due caratteri RT, seguiti dai 6 caratteri di codice ISTAT del comune in cui è localizzato il nodo, seguiti da 5 caratteri di progressivo a partire dal valore dei caratteri del codice ISTAT, ed infine GZ. Ciascun nodo è inoltre caratterizzato da un tipo, rappresentato dalla DataProperty km4c:nodeType, che può assumere i seguenti valori:

- intersezione a raso / biforcazione
- casello autostradale
- minirotatoria (raggio di curvatura < 10m)
- cambio sede
- terminale (inizio o fine elemento stradale)
- cambio toponimo / titolarità / gestore
- variazione classe di larghezza
- area di traffico non strutturato
- passaggio a livello
- nodo di supporto (per definire i loop)
- variazione classificazione tecnico – funzionale
- variazione stato di esercizio
- variazione composizione
- nodo intermodale per ferrovia
- nodo intermodale per aeroporto
- nodo intermodale per porto
- limite di regione
- nodo fittizio

In questo caso saranno presenti anche le DataProperty per la localizzazione, cioè geo:lat e geo:long.

Le regole di accesso sono descritte attraverso le istanze della classe km4c:EntryRule, identificabili univocamente attraverso il dct:identified di 15 caratteri così formato: i caratteri RT seguiti da 6 caratteri che rappresentano il codice ISTAT del comune, altri 5 caratteri che rappresentano il progressivo per quel codice ISTAT, e infine i caratteri PL. Le regole di accesso sono poi caratterizzate da un tipo, rappresentato dalla DataProperty km4c:restrictionType, che può assumere i seguenti valori:

- Blank (Solo in caso di manovre)
- Direzione flusso di traffico
- Passaggio bloccato
- Restrizioni speciali
- In costruzione
- Informazioni relative ai pedaggi
- Biforcazione

- Manovre proibite
- Restrizioni su veicoli

Oltre al tipo, le regole di accesso hanno anche una descrizione, detta anche valore della restrizione e rappresentata dalla DataProperty km4c:restrictionValue, che può assumere differenti range di valori, in base al tipo di restrizione con la quale abbiamo a che fare:

- Valori possibili per Blank:
    - Default Value = “-1”
- Valori per Direzione flusso di traffico & Restrizioni su veicoli:
    - Chiusa in direzione positiva
    - Chiusa in direzione negativa
    - Chiusa in entrambe le direzioni
- Valori per Passaggio bloccato:
    - Accessibile solo per veicoli di emergenza
    - Accessibile con chiave
    - Accessibile con Guardiano
- Valori possibili per Restrizioni speciali:
    - Nessuna restrizione(Default)
    - Restrizione generica
    - Solo Residenti
    - Solo impiegati
    - Solo per personale autorizzato
    - Solo per il personale
- Valori per In costruzione:
    - In costruzione in entrambe le direzioni
    - In costruzione in direzione di Marcia della corsia
    - In costruzione in direzione opposta alla direzione di Marcia della corsia
- Valori per Informazioni relative ai pedaggi:
    - Strada a pedaggio in entrambe le direzioni
    - Strada a pedaggio in direzione negativa
    - Strada a pedaggio in direzione positiva
- Valori per Biforcazione:
    - biforcazione multi corsia
    - biforcazione semplice
    - biforcazione di uscita
- Valori per Manovre proibite:
    - Manovra proibita
    - Svolta Implicita

La classe km4c:Maneuver presenta una sostanziale differenza dalle altre viste finora: ciascuna manovra è infatti identificata univocamente da un ID composto da 17 cifre numeriche. Una manovra è descritta dalla successione degli elementi stradali che interessa, che variano da 2 a 3, e dal nodo che interessa, quindi sarà quasi completamente descritta attraverso le ObjectProperties che rappresentano questa situazione. All'interno del Grafo Strade, troviamo dati relativi al tipo di manovra, tipo di biforcazione e tipo di manovra

proibita, ma visto che gli ultimi due tipi sono quasi sempre "non definiti", è stata associata una DataProperty solo al tipo di manovra, cioè km4c:maneuverType, che può assumere i seguenti valori:

- biforcazione
- manovra proibita calcolata (Calculated Maneuver)
- manovra obbligatoria (Restricted Maneuver)
- manovra proibita (Prohibited Maneuver)
- manovra prioritaria (Priority Maneuver)

La classe km4c:StreetNumber, presenta anch'essa un codice, dct:identified, per individuare univocamente le istanza di questa classe, stesso formato dei precedenti: i caratteri RT seguiti da 6 caratteri per il codice ISTAT del comune, altri 5 caratteri per il progressivo a partire dal codice comune e infine i caratteri CV. A Firenze sono presenti 2 numerazioni, l'attuale in colore nero, e l'originaria, di colore rosso; è stato quindi inserita una DataProperty, km4c:classCode, che tiene appunto conto del colore del civico e può assumere i seguenti valori: rosso, nero, nessun colore. Ciascun numero civico può inoltre essere formato, oltre che dalla parte numerica sempre presente, da una parte letterale, rappresentate rispettivamente dalle DataProperties km4c:number ed km4c:exponent. È stata inoltre inserita un ulteriore valore, km4c:extendNumber, all'interno del quale sarà memorizzato l'insieme del numero e dell'esponente, in modo da garantire una maggior compatibilità' con i differenti formati in cui potrebbero essere scritte/ricercate le istanze di questa classe.

La classe km4c:Milestone, come visto in precedenza, identifica il valore della chilometrica progressiva del tracciato, rispetto al punto d'inizio. Anche questa classe, per le sue istanza, presenta un codice univoco di riconoscimento formato da 15 caratteri, il dct:identified: RT, seguito da 6 caratteri per il codice ISTAT del comune, altri 5 caratteri per la progressiva di quel codice ISTAT, ed infine i caratteri CC. Su ciascun cippo chilometrico, solitamente viene scritto il km che corrisponde a quel punto e il contenuto di questa scritta è memorizzato attraverso la DataProperty km4c:text; grazie alle informazioni contenute nel grafo strade, risulta banale recuperare il nome della via, dell'autostrada etc. dove è posizionato il cippo; potrebbe quindi essere definita un'ulteriore ObjectProperty che colleghi la classe km4c:Milestone alla classe Road; questa informazione è tutt'ora recuperabile con un passaggio in più attraverso la classe km4c:RoadElement, ma può comunque essere inserita con facilità se lo si ritenesse opportuno in futuro. Anche in questo caso sono presenti le DataProperty per la localizzazione, cioè geo:lat e geo:long.

L'Accesso o km4c:Entry è l'elemento puntuale che identifica sul territorio l'accesso esterno, diretto o indiretto, principale o secondario ad uno specifico luogo di residenza/attività; l'accesso in pratica materializza la "targhetta" del numero civico. Come detto in precedenza, ogni accesso è connesso logicamente ad almeno un numero civico. Ciascun istanza della classe Entry è identificata univocamente dalla DataProperty dct:identified, formato da 15 caratteri, come tutti gli altri codici visti: RT seguito da 6 caratteri di codice ISTAT del comune, poi altri 5 caratteri del progressivo dal codice ISTAT ed infine i caratteri AC. Esistono solo tre tipi di accessi, il cui valore è memorizzato nella DataProperty km4c:entryType e sono l'"accesso esterno diretto", l'"accesso esterno indiretto" e l'"accesso interno"; oltre al tipo di accesso, per questa classe può essere utile conoscere se è presente un passo carrabile o no (DataProperty km4c:porteCochere). Sono inoltre presenti anche in questo caso le DataProperty per la localizzazione, cioè geo:lat e geo:long.

La classe km4c:Service è stata corredata di una contact card, cioè di una serie di DataProperty prese in prestito dall'ontologia schema.org (http://schema.org) per rendere più standardizzata la descrizione delle

varie aziende: schema:name, schema:telephone, schema:email, schema:faxNumber, schema:url, schema:streetAddress, schema:addressLocality, schema:postalCode, schema:addressRegion alle quali abbiamo aggiunto skos:note per eventuali aggiunte come gli orari di apertura che a volte sono presenti nei dati. Oltre a queste, sono state definite altre DataProperty come km4c:houseNumber, per isolare il numero civico dall'indirizzo e, quando possibile, la geo:lat e geo:long per la localizzazione.

Con l'introduzione delle DigitalLocation la classe Service è stata corredata delle seguenti proprietà:

- km4c:hasGeometry: proprietà definita per memorizzare l'insieme di coordinate associate alla digitalLocation, che può essere un POINT, un LINESTRING ed un POLYGON;
- km4c:routeLength: per memorizzare la lunghezza dei percorsi (ad esempio di Jogging);
- km4c:stopNumber: proprietà che indica il numero di fermata all'interno dell'intero percorso della Linea 1 del Tram;
- km4c:lineNumber: proprietà che indica la linea di appartenenza di una fermata.
- km4c:managingBy: proprietà che indica il gestore della DigitalLocation;
- dct:description: proprietà in cui è riportata la descrizione della DigitalLocation;
- km4c:owner: proprietà che indica il proprietario della DigitalLocation;
- km4c:abbreviation: proprietà che riporta eventuali abbreviazioni di nome della DigitalLocation;
- km4c:type: proprietà che indica la categoria a cui appartiene il gestore di una DigitalLocation, oppure il tipo di ZTL o di museo o dell'area di riferimento.
- km4c:time: proprietà che indica l'orario di apertura di un esercizio;
- km4c:firenzeCard: proprietà che indica se la DigitalLocation è affiliata con la FirenzeCard;
- km4c:multimediaResouce: proprietà per associare immagini ed mp3 alle singole DigitalLocation;
- km4c:districtCode: proprietà che specifica il codice del quartiere in cui si trova la DigitalLocation;
- km4c:routePosition: proprietà che indica la posizione della DigitalLocation all'interno di un percorso tematico;
- km4c:areaCode: proprietà che indica un codice comunale per l'area di riferimento;
- km4c:routeCode: proprietà che specifica il codice del percorso tematico a cui fa riferimento la DigitalLocation.
- schema:price: proprietà che indica l'eventuale costo di ingresso;

La classe km4c:Event invece possiede tutte le proprietà appartenenti alla schema:Event e altre proprietà aggiunte in base alle informazioni fornite dal Comune di Firenze. Per comodità di seguito tutte queste proprietà saranno riportate in forma di elenco.

- schema:startDate: proprietà che indica la data di inizio dell'evento;
- schema:endDate: proprietà che indica la data di fine dell'evento;
- schema:description: contiene la descrizione dell'evento;
- schema:image: contiene, se presente, url di un immagine di riferimento per l'evento;
- schema:name: è il nome dell'evento;
- schema:url: sito web dell'evento;
- schema:telephone: numero di telefono di riferimento per l'evento;
- schema:streetAddress: indirizzo del luogo in cui si terrà l'evento;
- schema:addressLocality: città in cui si terrà l'evento;
- schema:addressregion: provincia in cui si terrà l'evento;
- schema:postalCode: CAP della città in cui si terrà l'evento;

- km4c:houseNumber: numero civico del luogo in cui si terrà l'evento;
- skos:note: proprietà che conterrà eventuali note o altre informazioni presenti per l'evento;
- schema:price: proprietà che indica l'eventuale costo di ingresso;
- dct:identifier: contiene l'identificativo dell'evento assegnatogli dal Comune di Firenze;
- km4c:eventCategory: proprietà per indicare il tipo di evento;
- km4c:placeName: contiene il nome del luogo in cui si svolgerà l'evento;
- km4c:freeEvent: variabile float che indica se l'evento è gratuito o meno;
- km4c:eventTime: proprietà che contiene l'orario di inizio dell'evento;

Oltre a queste proprietà, un evento è corredato anche di una coppia di coordinate che lo geo localizzano, cioè geo:lat e geo:long.

La classe gis:Geometry presenta invece soltanto la proprietà gis:asWKT, la quale permette di definire una unica stringa rappresentante l'insieme di coordinate che definiscono la forma geometrica dell'oggetto, cioè ad esempio:

LINESTRING ((11.21503989 43.77879298, 11.21403307 43.77936126, 11.21385829 43.77947115))

La classe km4c:CarParkSensor presenta 2 DataProperty specifiche per i parcheggi: il dct:identified, definito a livello regionale sempre attraverso un codice di 15 caratteri iniziante per RT e terminante con la sigla della provincia di appartenenza, km4c:capacity, cioè la capacità in numero di posti del parcheggio; a tale classe è connessa la classe km4c:SituationRecord dove sono state definite come DataProperty i valori km4c:fillrate ed km4c:exitrate, cioè rispettivamente il numero di veicoli in entrata/uscita per ora, km4c:carParkStatus, cioè una stringa che descrive l'attuale stato del parcheggio (possibili valori sono "enoughSpacesAvailable", "carParkFull", "noParkingInformationAvailable", etc), l'id univoco, cioè dct:identified, allo stato di validità del record (DataProperty km4c:validityStatus, che può essere solo "active" nel caso di parcheggi), la km4c:parkOccupancy, cioè il numero di posti occupati, o il corrispondente in percentuale che è invece chiamato km4c:occupied, ed infine i posti liberi, per i quali si utilizza la DataProperty km4c:free.

La classe km4c:WeatherReport è caratterizzata dalle DataProperty dct:identified, contenente l'id univoco che permette di identificare i differenti report, km4c:timestamp che indica il tempo in cui è stato creato il report espresso in millisecondi; sono state poi definite DataProperty per esprimere la fase lunare (km4c:lunarPhase) e per l'orario di tramonto/sorgere del sole e della luna, cioè km4c:sunrise, km4c:sunset, km4c:moonrise e km4c:moonset. Sono inoltre presenti le DataProperty km4c:heightHour e km4c:sunHeight che rappresentano rispettivamente l'orario in cui il sole raggiunge la massima altezza e tale altezza. Ciascuna istanza di WeatherPrediction è caratterizzata dalle DataProperties km4c:day, che rappresenta il giorno a cui fa riferimento la previsione (che insieme all'id del report, costituisce un modo univoco per identificare le singole previsioni) i valori di temperatura minimi e massimi o reali e percepiti (rispettivamente rappresentati dalle DataProperties km4c:minTemp, km4c:maxTemp, km4c:recTemp, km4c:perTemp), la direzione del vento, km4c:wind, km4c:humidity cioè la percentuale di umidità, la quota a cui è presente la neve km4c:snow, la parte della giornata a cui fa riferimento la singola previsione contenuta in un report indicata con km4c:hour e l'indice ultravioletto della giornata da km4c:uv.

Le classi km4c:SensorSiteTable e km4c:SensorSite hanno soltanto una DataProperty ciascuna, relativa all'id, rappresentato per entrambe con dct:identified. La classe km4c:Observation invece è completata dalle DataProperties dct:identified, dct:date, provenienti dalla DublinCore ontology e dalle DataProperty km4c:averageDistance e km4c:averageTime (rappresentanti distanza e tempo medio tra due auto),

km4c:occupancy e km4c:concentration (relative alla percentuale di occupazione della strada e al numero di auto e alla concentrazione di auto), km4c:vehicleFlow (flusso di veicoli rilevato dai sensori) e i dati relativi alla velocità media e al percentile calcolato su tali velocità: km4c:averageSpeed, km4c:thresholdPerc e km4c:speedPercentile.

La classe km4c:PublicTransportLine e la classe km4c:Lot presentano entrambe come DataProperties la dct:identifier e la dct:description dalla ontologia DublinCore e rappresentano rispettivamente il numero della linea/del lotto e la descrizione del percorso che effettua o del lotto.

La classe km4c:Route invece oltre alle DataProperties dct:identifier, foaf:name e dct:description, presenta il campo km4c:routeLenght, cioè la lunghezza del percorso espresso in metri e la direzione del percorso (km4c:direction).

La classe km4c:BusStop presenta le DataProperty dct:identifier, foaf:name per il nome della fermata, e le classi geo:lat e geo:long appartenenti alla Geo Ontology. Queste ultime due DataProperties sono anche le uniche presenti nella classe RouteJunction, oltre al dct:identifier.

La classe km4c:BusStopForecast contiene soltanto le DataProperty per il tempo di arrivo previsto e per l'identificativo, rispettivamente km4c:expectedTime e dct:identifier.

La classe km4c:AVMRecord necessita invece delle DataProperty per identificare il mezzo a cui fa riferimento il record (km4c:vehicle), il tempo di arrivo all'ultima fermata (km4c:lastStopTime), lo stato della corsa, cioè se è in anticipo o in ritardo o puntuale (km4c:rideState), l'ente gestore e l'ente proprietario del sistema AVM (km4c:managingBy e km4c:owner), l'identificativo univoco del record (dct:identifier), e le coordinate geo:lat e geo:long, che indicato la posizione del mezzo al momento dell'invio del report.

Infine la classe km4c:Ride presenta solo la DataProperty dct:identifier, come la classe km4c:RouteLink. La classe RouteSection, invece presenta oltre all'id anche la DataProperty km4c:distance, dove viene salvato la distanza tra due fermate successive all'interno di un percorso.

Analizziamo adesso le DataProperty relative alle classi km4c:Beacon e km4c:BeaconObservation: la prima classe, oltre alla DataProperty dct:identifier, presenta le proprietà km4c:owner, nella quale è memorizzato il proprietario del singolo beacon, schema:name che contiene l'eventuale nome associato ad un beacon, km4c:uuid, km4c:minor, km4c:major, km4c:public (definisce se il beacon è pubblico oppure no), ed infine le DataProperty per la memorizzazione della posizione, cioè geo:lat e geo:long. La seconda classe, cioè BeaconObservation, presenta invece, oltre al dct:identifier che rende univoca ciascuna lettura, le coordinate dei dove ha effettuato una connessione, memorizzate nelle DataProperty geo:lat e geo:long, la dct:date che memorizza giorno e ora della osservazione ed infine possiede la DataProperty km4c:power, che indica la potenza con cui ha rilevato la connessione con un utente, che permette appunto di stabilirne la distanza.

Sono state inoltre definite delle proprietà da associare al Context, relative ad informazioni provenienti dalle tabelle che descrivono i singoli processi ETL che elaborano i differenti dati; per ciascun processo infatti,il cui nome è memorizzato nel campo dct:title, viene definito un tipo di sorgente, memorizzato all'interno del campo dct:source, il formato originale dei dati (csv, dbf etc) memorizzato nel campo dct:format, la descrizione del dataset dct:description, la licenza associata al dataset è invece salvata nel campo dct:license, il tipo di processo a cui si fa riferimento è invece salvato nella DataProperty km4c:processType, la DataProperty km4c:automaticity dice se il processo è automatizzato oppure no perché, ad esempio,

dataset come il grafo stradale non possono essere completamente automatizzati a causa del processo di ottenimento dati che necessita di inviare e ricevere email, km4c:accessType informa su come sono recuperati i dati (chiamate HTTP, Rest, ecc.), km4c:period contiene il periodo di tempo (in secondi) che intercorre tra 2 chiamate di uno stesso processo, km4c:overTime indica dopo quanto tempo un processo deve essere killato, km4c:lastUpdate rappresenta la data di aggiornamento dei dati, mentre km4c:lastTriples la data di generazione triple.

La classe km4c:RailwayLine ha soltanto 3 DataProperty, la dct:identifier che contiene l'identificativo della linea ferroviaria (codice di 12 char iniziante con le lettere RT, seguite da 3 caratteri che identificano la regione, T09 per la Toscana 5 char per il numero progressivo ed infine le lettere PF), la foaf:name in cui è salvata la denominazione convenzionale e la dct:alternative in cui è memorizzata invece la denominazione ufficiale se presente.

La classe km4c:RailwayDirection invece ha soltanto le prime due DataProperty specificate per km4c:RailwayLine, con lo stesso utilizzo: dct:identifier, in cui viene memorizzato il codice formato da 12 char iniziante con le lettere RT, seguite da 3 caratteri che identificano la regione, T09 per la Toscana 5 char per il numero progressivo ed infine le lettere ED e foaf:name in cui viene memorizzata la denominazione convenzionale.

La classe km4c:RailwayElement, oltre al solito campo dct:identifier, formato da 12 caratteri che seguono le seguenti regole: I caratteri RT seguiti da 3 caratteri di codice regionale, T09 per la Toscana, seguito da 5 numeri progressivi ed infine le lettere EF, necessita della DataProeprty km4c:elementType (che può assumere i seguenti tre valori "ferrovia ordinaria" "ferrovia AC/AV" e "altro"), della km4c:operatingStatus che può assumere solo i valori "ferrovia in costruzione", "ferrovia in esercizio" e "ferrovia dismessa", del campo km4c:elemLocation che indica la sede dell'elemento ferroviario (e può assumere i valori "a raso", "su ponte/viadotto" e "in galleria"), della DataProperty km4c:supply che specifica se si tratta di una "linea elettrificata" oppure di una "linea non elettrificata", del campo km4c:gauge cioè il tipo di scartamento (può assumere soltanto 2 valori "ridotto" o "standard"), della DataProeprty km4c:underpass che può assumere i seguenti valori:

- l’elemento non è in sottopasso di nessun altro oggetto
- l’elemento è in sottopasso di un altro oggetto
- l’elemento è contemporaneamente in sovrappasso e sottopasso di altri oggetti

Altre DataProperty che sono state definite sempre per la classe km4c:RailwayElement sono la km4c:lenght cioè la lunghezza in metri dell'elemento, km4c:numTrack cioè il numero di binari (se 0 la linea è in costruzione o dismessa), ed infine il km4c:trackType, che specifica se l'elemento è composto da "binario singolo" o "binario doppio".

La classe km4c:RailwaySection necessita delle DataProperty km4c:axialMass, cioè la classificazione della linea rispetto alla massa assiale, la quale può assumere i seguenti valori:

- D4 - corrispondente a massa per asse pari a 22.5 t
- C3 - corrispondente a massa per asse pari a 20.0 t
- B2 - corrispondente a massa per asse pari a 18.0 t
- A - corrispondente a massa per asse pari a 16.0 t
- Non definita

Sono poi state definite le DataProeprty km4c:combinedTraffic (che può assumere i valori "PC80","PC60","PC50","PC45","PC32","PC30","PC25","PC22", "linee con il profilo limite di carico FS" e "non definito"), dct:identifier (il solito codice di 12 char che inizia con i caratteri RT, seguito dal codice regionale e da 5 char di numero progressivo e termina con i caratteri TR), e la foaf:name cioè la denominazione convenzionale della tratta.

Per la classe km4c:RailwayJunction sono state definite solo 3 DataProeprty: dct:identifier, cioè il codice identificativo di 12 char formato come nei precedenti casi ma terminante per GK, foaf:name, cioè la denominazione ufficiale per bivi, stazioni e scali ed infine km4c:juncType, che può assumere uno dei seguenti valori:

- passaggio a livello
- terminale (inizio o fine)
- bivio (confluenza o diramazione)
- stazione/fermata/casello ferroviario
- scalo merci
- interporto
- variazione di stato (COD_STA)
- variazione di sede (COD_SED)
- variazione numero dei binari (Num_bin)
- variazione alimentazione (COD_ALI)
- limite amministrativo

La classe km4c:TrainStation presenta invece il solito campo di 12 cifre dct:identifier (comporto da RT seguito da 3 char di identificativo personale, T09 per la Toscana, 5 char di progressivo e infine i caratteri SF), la DataProperty foaf:name in cui è memorizzata la denominazione ufficiale riportata per esteso, l'indirizzo reperito dall'elenco pubblicato sul sito RFI e memorizzato nei campi schema:addressRegion, schema:addressLocality, schema:postalCode, schema:streetAddress, la DataProperty km4c:managingAuth, cioè l'ente gestore riportato sul sito RFI, la categoria a cui appartiene la stazione (DataProperty km4c:category), e infine il campo km4c:state, cioè lo stato della stazione che può assumere i valori "Attiva", "Non Attiva" e "Facoltativa con fermata a richiesta".

La classe km4c:GoodYard presenta, oltre al codice di 12 char (formato come tutti i precedenti ma terminante in SM) memorizzato nel dct:identifier, la DataProperty foaf:name in cui è memorizzato il nome dell'impianto merci, la km4c:railDepartment che mantiene la denominazione del compartimento ferroviario, km4c:railwaySiding cioè la definizione della caratteristica fisica del numero di raccordi, km4c:yardType che indica se gli scali sono pubblici (valore "scali pubblici") o se i raccordi linea sono ad uso privato (valore "raccordi in linea"), ed infine la DataProeprty km4c:state che indica se lo scalo è "attivo" o "in costruzione".