UNIVERSITÀ DEGLI STUDI FIRENZE

Ph.D. in
Informatics, Systems and Telecommunications

CYCLE XXVII

Curriculum: Telematics and Information Society

Architecture and Knowledge Modelling for Smart City

ING-INF/05

Ph.D. Student:	Nadia Rauch
Tutor:	Prof. Paolo Nesi
Coordinator:	Prof. Luigi Chisci

Years 2012 / 2014

Acknowledgment

I would like to sincerely thank my tutor Prof. Paolo Nesi, for his guidance and support during these three years, for his expertise and enthusiasm: most of all, thanks for believing in me and for encouraging my research and for allowing me to grow professionally and personally.

I would also like to thank my colleague Pierfrancesco Bellini, always available to clarify my doubts and give me a hand with my research.

I am very grateful to my colleagues Mariano Di Claudio, Giacomo Martelli, Giovanni Ortolani, Marco Serena e Riccardo Billero working intensively with them in recent months has been a pleasure to me;

A special thanks to my best friend, my sister, my soulmate Lara, always by my side since the beginning of this university career… so many times I could have lose myself without her.

Finally, thanks to Andrea for having put up with my ups and downs, for his support and for sharing with me his life.

Abstract

This thesis presents and details a smart-city ontology, called KM4City that is a knowledge model for the city and its data.
The knowledge model pursues the objective of interconnect data gathered in the city, to transform it into semantically interoperable information.
In fact, a variety of Open/Closed Data information sources are available from public administrations ranging from structural, statistical to real-time information. In most cases, this information has different formats, presents inconsistencies, incompleteness, and their semantic description is not sufficient to automatically compose them to have integrated global information of the area.
Smart City ontology is not yet standardized, and a lot of research work is needed to identify models that can easily support the data reconciliation (essential in order to effectively interconnected data to each other), the management of the complexity, to allow the data reasoning. In this thesis, a system for data ingestion and reconciliation of smart cities related aspects as road graph, services available on the roads, traffic sensors etc., is also proposed. The system allows managing a big data volume of data coming from a variety of sources considering both static and dynamic data. These data are mapped to the presented KM4City ontology, and stored into an RDF-Store where they are available for applications via SPARQL queries, to provide new services to the citizens via specific applications of public administration and enterprises.
In this thesis, the results that could be obtained by applying the ontology created, are also shown, which allowed to combine all data provided by the city of Florence and the Tuscany region including: maps, traffic status, weather conditions and forecast, parking status, real time sensors on public and private vehicles, point of interests in the city as museums, monuments, restaurants, hotels, hospitals, etc. but also statistical data like travel accidents, per street per year. Finally, an application that take advantage of the created repository and ontology, will be shown, which implement new integrated services related to mobility.
The dissertation also presented the work performed about reconciliation algorithms and their comparative assessment and selection.
The KM4City ontology realized in this thesis, has also been involved in the activity of DISIT lab mainly related to a number of smart city projects, especially among them Sii-Mobility which aims at collecting and exploiting data by solving the above mentioned problems and providing integrated data to be used for implementing smart city services for citizens mobility, public administrations, and SMEs.

Index

Acronyms and Abbreviations

AmI Ambient Intelligence
API Application Programming Interface
AVM Automatic Vehicle Monitoring
CAP Common Alerting Protocol
CC Creative Common
CC0 Creative Commons 0
CC - By Creative Common Attribution
CC - By SA Creative Common by Share Alike
CSV A comma-separated values file stores tabular data (numbers and text) in plain-text form.
DBF dBASE Table File Format. File format used to store tables of data
DBMS Database Management System
DC Dublin Core Ontology, a light weight RDFS vocabulary for describing generic metadata
DISIT Distributed Systems and Internet Tech lab & Distributed Data Intelligence Lab of UNIFI
DPA Data Pack Accessory
DPR Data Protection Rights
DRM Digital Rights Management
DUL DOLCE Ultra Lite
EPR Electronic Patient Record
ETL Extraction, Transformation, and Loading
ESRI Environmental Systems Research Institute
FOAF Friend of a friend
FTS Full Type Search
GDF Geographic Data Files
GIS Geographic Information System
GPS Global Positioning System
GT Groundtruth
GUI Graphical User Interface
HBP Human Brain Project
HTML Hypertext Markup Language
HTS High throughput sequencing
HTTP Hypertext Transfer Protocol

ICT Information and Communication Technology
IEC International Electrotechnical Commission
IMS IP Multimedia Systems
IODL Italian Open Data License, equivalent to CC-BY-SA (>3.0), and to ODC-ODbL.
IoT Internet of Things
ISO International Standard Organization
ISR Intelligence, Surveillance, and Reconnaissance
IT information technology
ITS Intelligent Transport Systems
JS JavaScript
JSON JavaScript Object Notation, a lightweight data-interchange format easy for humans to read and write and, for machines, to parse and generate
KB Knowledge Base
KLM Keyhole Markup Language, XML notation, expressing geographic annotation and visualization, two-dimensional maps and three-dimensional Earth browsers
KPI Key Performance Indicators
LD Linked Data
LOD Linked open Data
LOG Linked Open Graph, a service and tool of UNIFI DISIT for accessing graphically to SPARQL entry point of LOD
LPT Local Public Transport (in Italian Trasporto Pubblico Locale, TPL)
LSL Link Specification Language
LTE Long Term Evolution
MIIC Mobility Information Integration Center of the Tuscany region
NIEM National Information Exchange Model
N3 Notation3 language, a compact and readable alternative to RDF's XML syntax.
NL Natural Language
NLP Natural Language Processing
NoSQL No Structured Query Language,
OD Open Data
ODbL Open Database License
ODP Ontology Design Pattern
OSIM Open Space Innovative Mind, tools of UNIFI DISIT
OTN Ontology on Transportation Network
OWL Web Ontology Language
PA Public Administration, such as municipality, city administration
PDF Portable Document Format is a file format used to present documents in a manner independent of application software, hardware, and operating system
POS predicate-object-subject index
PSO predicate-subject-object
RDF Resource Description Framework
RDFS Resource Description Framework Schema

RFC Request for Comments

RTZ Restricted Traffic Zone (in Italian Zona Traffico Limitato,ZTL)

SHP The Esri shapefile, or simply a shapefile, is a popular geospatial vector data format

SKOS Simple Knowledge Organisation System

SME Small or Medium Enterprise

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol

SPARQL SPARQL Protocol and RDF Query SQL Structured Query Language

SSN SemanticSensorNetwork Ontology

SSO Stimulate-Sensor-Observation Pattern

TTL Filename extension of a Turtle (Terse RDF Triple Language) file

UI User Interface

UN United Nations

UML Unified Modelling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

UTC Urban Traffic Control

VM Virtual Machine on cloud

W3C World Wide Web Consortium

Wod Web of Documents

WOD Web of Data

WS Web Service

WSDL Web Services Description Language

WSN Wireless Sensors Network

XML Extensible Markup Language

List of Figures

List of Tables

List of Diagrams

Chapter 1

1. Introduction

In recent years we have witnessed a revolution of "data". From the point of view of the government, in fact, these years were marked by the opening of data, a topic that is still the focus of debate between leading experts in innovation. This theme, nowadays, is emerging in public opinion: journalism, information visualization, territory mapping, see an opportunity in the public data recovery, quality and savings.

The Open Government can be considered as the natural evolution of e-government, a process that has characterized the late 90s and has allowed the introduction of Information and Communication Technology (ICT) in Public Administrations (PA). One of the most characteristic aspects of Open Government, is the Open Data, i.e. the publication on the Internet, in an open format, of the government agencies data, which can then be reused by citizens, businesses and other Public Administrations.
Thanks to the Open Data and their creative reuse for the development of applications, an acceleration of the economic development can be obtained: this process allows to see the Public Administrations also as facilitators of the innovation process.

A further contribution to this revolution there was due to enabling technologies such as pervasive networks, which have radically changed the human life, the acquisition of knowledge, the way in which works are performed and people learn. The availability of newer technology reflect on how the relevant processes should be performed in the current fast changing digital era. This leads to the need of adoption of a variety of smart solutions in large part of environments to enhance the quality of life and improve the performance of the citizens.

Due to these facts modern world is indeed full of devices, including sensors, actuators, and processors of data [Zaslavsky et al, 2013]: such concentration of computational resources allows sensing, capturing, collecting and process real-time data from billions of connected devices, serving many different applications including environmental monitoring, industrial applications, business applications, and pervasive applications human-centric.

Furthermore, the advancement of sciences, engineering and technologies, and the social and economic activities have collectively created a torrent of data in digital form.

In 2010, the total amount of data on earth exceeded one zettabyte (ZB) [Zikopoulos, 2012]; by end of 2011, this number grew up to 1.8 ZB [Reed et al, 2012]. Further, it is expected that it will reach 35 ZB in 2020.

So, this enormous volumes of data created every day, placed a great interest on the theoretical and practical aspects of extracting knowledge from this data sets [Dean and Ghemawat 2008], and, how to turn this big data phenomenon into a positive force for good, has drawn tremendous and intensified interest from an ever increasing set of big data stakeholders.

No coincidence that in recent years Big Data has become a popular buzzword, thanks also to the fact that research issues behind big data and big data analysis are embedded in multi-dimensional scientific and technological spaces.
In fact the issues related to this new concept has been also widely discussed, especially those relating to the management of this huge volume of data and what benefits they can provide in various application fields. For instance data streams, coming from ubiquitous sensors introduced above, can be collected and aggregated in existing/newly social networks creating a new generation of networked media. This information can be used to take decisions based on the data itself, rather than based on artificially constructed models of reality.

In addition, the recent availability of datasets on transportation networks, with high spatial and temporal resolution, is enabling new research activities in the fields of Territorial Intelligence and Smart Cities, which aim to improving mobility, a paramount issue due to the big environmental and social impact of vehicular traffic, and to billions of dollars that traffic congestions cost to the society every year [Manyika et al, 2011].
As a result, to the recent availability of high quality spatiotemporal datasets, coming from Floating Car Data and other data sources, thanks to the advances in sensor technologies (like Smartphones, GPS handhelds, etc.), new research opportunities are arising in the direction of smarter solutions for mobility.
Moreover, the local transport system has high social costs connected to the inconvenience of citizens towards mobility solutions, because the low and/or absent interoperability among transport management and monitoring systems; mobility services; services and systems for the transport of goods; ordinances and services (works, hospitals, shopping centers, museums, ..); private transport, rail transport, parking lots, and the people that move, and due to the limited capacity of the system to receive and respond to changes in the city and citizens.
Urban data, in fact, represent a large portion of the data collected daily in the city, and they can be seen as data for cities, that are invariably tagged to space and time.

The concept of Big Data and Smart City are in fact closely related: a report of the United Nations (UN) has in fact shown that, by 2050, 70% of the global population will live in urban areas; in spite of occupying only 2% of the Earth, these cities are responsible for 70% of global energy consumption and greenhouse gas emissions [Liguria, 2014].
Therefore it is important that cities embark on a path that will lead them to increase their "intelligence" or Smartness. A city is "intelligence" or smart, if it intelligently manages economic activities, mobility, environmental resources, relationships between people, policies of housing and method of administration.
For the Spanish economist Gildo Seisdedos Domínguez, the concept of Smart City is based essentially on the efficiency which in turn is based on business management, ICT integration and active participation of citizens [Caragliu et al, 2009], [Komninos, 2002]. This implies a new kind of governance with the involvement of citizens in authentic public policy.
Historically, the Smart City project was born in the world over 5 years ago, with the city of Rio de Janeiro, which acted as a pioneer of the first examples of “smart” implementation of technology in order to improve community life and reduce waste in many different areas, ranging from the energy sector to that of waste management [Jenny, 2014].
Europe, instead, started in 2009 to talk about "smart" and to devote the first investments in projects to "smart" implementations. Projects mainly concern the eco-sustainability, the reduction of environmental waste and improvement of public transport planning; the European Union provides for a total expenditure that is between 10 and 12 billion euros over a period of time that extends until 2020.
Cities must therefore take charge of the great problem of managing scarce resources and providing critical public services such as security, transportation, energy and water.
Here are a few examples of how Big Data technology could improve our cities:

- **Public Safety:** the efficiency of police and fire services could improve, by capturing and correlating all the data coming from different systems installed in the city including surveillance cameras, emergency vehicle GPS tracking and fire and smoke sensors.
- **Urban transportation:** through real time data capture and the management of signals from video cameras and magnetic sensors installed in the road network, GPS systems could be used to track the location of public buses. Equally, social media monitoring systems could enable us to flag a protest organized on social networks and therefore, facilitate the management of potential traffic jams by changing bus routes, modifying traffic light sequences and delivering information to drivers via mobile apps indicating approximate driving times and giving alternative routes.
- **Water management:** by analyzing the data coming from metering systems, pressure or PH sensors installed in water supply networks and video cameras situated in water treatment plants, it would be possible to optimize water

management detecting leaks, reducing water consumption and mitigating sewer overflow.

- **Energy management**: with all data coming from smart electric meters installed in customer’s homes, as well as meteorological open data platforms, it would be possible to optimize energy production, depending on demand, which would help us to maximize the integration of renewable energy resources like wind and solar energy.
- **Urban waste management:** by gathering data in real time from sensors, that detect the container filling level and comparing it to the historical data and usage trends, it would be possible to forecast the ideal time for emptying each individual container and optimize waste collection routes.
- **Public Sentiment Analysis:** by analyzing social media networks and blogs and then using Big Data technologies, cities would be able to measure public opinion on key issues and services, such as public transportation, waste management or public safety allowing them to priorities and shape policy.

But the results that can be achieved with a widespread use of Big Data analysis, are varied and more extensive than those presented above: these can translate into creating values in healthcare, improving the productivity in manufacturing, accelerating the pace of scientific discoveries for life and physical sciences, developing a competitive edge for business, retail, or service industries, and innovating in education, media, transportation, or government.

To reduce the social costs and to make cities smarter, services must be optimized, for example using integrated information, greater interoperability and integration among systems, and establishing a dialogue with the communities and individual citizens.

In fact, most of the public and private data sets, from which a city can take advantage, are not semantically interoperable: this means that even if they are ingested within a single Knowledge base, identical concepts may be not connected, and thus leading to the impossibility of making global inference and deductions on the knowledge base.

To better clarify, the ingestion of multiple open data to obtain a semantically interoperable model grounded on a common ontological model is a well-known complexity. In fact, differences in semantics can be due to differences in formats, coding elements (different ID formats), naming, languages and concepts. Concepts may come from different context, sectors and languages as: PA, mobility and transport, commercial, cultural heritage, social media. This problem has to be addressed at the first ingestion and update: during data linking the access and exploitation of well-known data/references (e.g., dpPedia, Europeana, Geonames, etc.) can be of help in disambiguating, but also unified thesaurus and models can be of help, such as Babelnet [BABELNET], OSIM [OSIM] and EuroVoc [EUROVOC].

It is clear that, if data collected are not made semantically interconnected, the "city" ecosystem will not be able to begin its "smarteness" process, and therefore will not become a real Smart City.

The purpose of this thesis is therefore to provide a contribution to the resolution of the problems just seen, through the implementation of an architecture that allows to integrate and especially interconnect within a single repository, all the information coming mainly from the various sensors installed in cities, from public administrations, and from the public transport companies.

The research project is conducted in the context of the “Telematics and Information Society” PhD programme at the University of Florence.

The thesis is divided into two parts, state of the art and discussion of the work.
The first part of this report describes the technologies relevant to the topics discussed, that is Big Data, Open Data and Linked Data.
In Chapter 2, in fact, the work done in the Big Data field is presented, along with a review of existing solutions. In the same chapter, Open Data and Linked Data are discussed, because they allow the definition of a well-defined model for the data on the Web, such as RDF. RDF is in fact the framework on which is based the storage component of the presented architecture.
In Chapter 3 will present the most important datasets that have been analyzed and processed during the research period and in Chapter 4, the concept of ontology is explained, along with the state of the art of the existing ontologies and knowledge models relating to Smart City. After that, in the same chapter, the *Km4City* ontology [Bellini et al., 2014] is presented in detail: it represent the focus of work and research carried out for over a year. In the same chapter is also provided a hint to the ISO standard on Smart City, which is still being finalized.
Chapter 5 is the heart of the second part of the thesis: it is the chapter where the whole architecture built, is presented; at each stage of its life cycle a section of the chapter is dedicated, in order to clarify for each, what is its purpose and why it was developed.
In Chapter 6 are then presented the experiments conducted on the architecture developed, and it is also explained how the *Km4City* ontology will be used in the project *SiiMobility*, a project cofounded by Italian Ministry of Instruction, University and Research. The consequent results are presented and evaluated with regard to their significance and quality, also in Chapter 6.
Finally, Chapter 7 presents some future projects made possible by the exploitation of the interconnected triplestore that was created.

Part I
State of the Art

Chapter 2

2. Enabling Technology

In the previous chapter some concepts have been introduced, which are fundamental for the research carried out in these years, and which are therefore key argoments to the proper understanding of the work performed.
This chapter is then dedicated to these arguments, and it has the purpose of providing a sufficient description to facilitate understanding of the remaining part of this thesis.
Initially, issues related to *Big Data* will be discussed as well as issues related thereto; subsequently current issues as *Open Government* and *Open Data* will be introduced, up to reach a wider concept as *Linked Data*.

2.1 Big Data

The management of huge and growing volumes of data is a challenge since many years, whereas nowadays no long term solutions have been found [Bellini et al., 2013A]. The term “Big Data” initially referred to huge volumes of data that have size beyond the capabilities of current database technologies, successively for “Big Data” problems one referred to the problems that present a combination of large volume of data to be treated in short time. When one establish that data have to be collected and stored at an impressive rate, it is clear that the biggest challenge is not only about the storage and management; their analysis and the extraction of meaningful values, deductions and actions is in reality the main challenge. Big data problems were mostly related to the presence of unstructured data, i.e. information that either do not have a default schema/template or do not adapt well to relational tables; it is therefore necessary to turn to analysis techniques for unstructured data, to address these problems.
Recently the big data problems are characterized by a combination of the so called 3V’s: *volume*, *velocity*, *variety*; and then a forth V has been added: *variability*. In substance, every day a large *volume* of information is produced and this data need a sustainable access, process and preservation according to the *velocity* of their arrival, therefore the management of large volume of data is not the only problem. Moreover, the *variety* of data, metadata, access rights and associating computing, formats, semantics and software tools for visualization, and the *variability* in structure and data models, significantly increase the level of complexity of these problems. The first V,

volume, describes the large amount of data generated by individuals, groups and organizations. The volume of data being stored today is exploding. For example in the year 2000 were generated and stored about 800.000 Petabytes of data in the world [Eaton et al., 2012] and experts estimated that in the year 2020, about 35 ZettaByte of data will be produced. The second V, *velocity*, refers to speed at which Big data are collected, processed and elaborated, may be handle constant flow of massive data that are impossible to be processed with traditional solutions. For this reason, it not important only to consider “where” the data are stored, but also “how” they are stored. The third V, *variety*, is concerned to the proliferation of data types from social, mobile sources, machine-to-machine and traditional data that are part of it. With the explosion of Social Networks, smart devices, sensors, data has become complex, because it includes raw, semi-structured and unstructured data from log files, web pages, search indexes, cross media, emails, documents, forums and so on. *Variety* represents all type of data and usually the enterprises must be able to analyze all them, if they want to gain advantages. Finally, *Variability*, the last V, refers to data unpredictability and to how these may change in the years, following the implementation of the architecture. Moreover, the concept of variability can be connected to assigning a variable interpretation to the data and to the confusions created in big data Analysis, referring for example, to different meanings in Natural Language that some data may have. These four properties can be considered orthogonal aspects of data storage, processing and analysis and it is also interesting that increasing variety and variability, also increases the attractiveness of data and their potentiality in providing hidden and unexpected information/meanings.

Especially in science, the need of new "*infrastructures for global research data*" that can achieve interoperability, and to overcome the limitations related to language, methodology and guidelines (policy), would be needed in short time. To cope with these types of complexities, several different techniques and tools may be needed, they have to be composed and new specific algorithms and solutions defined and implemented. The wide range of problems and the specifics needs make almost impossible to identify unique architectures and solutions adaptable to all possible applicative areas. Moreover, not only the number of application areas, so different from each other, but also the different channels through which data are daily collected, increases the difficulties of companies and developers to identify which is the right way to achieve relevant results from the accessible data. Therefore, this chapter can be a useful tool for supporting the researchers and technicians in making decisions about setting up some big data infrastructure and solutions. To this end, it can be very helpful to have an overview about big data techniques; it can be used as a sort of guideline to better understand possible differences and relevant best features among the many needed and proposed by the product as the key aspects of big data solutions. These can be regarded as requirements and needs according of which the different solutions can be compared and assessed, in accordance with the case study and/or application domain.

To this end, and to better understand the impact of big data science and solutions, in the following, a number of ***examples*** *describing major applicative domains taking advantage from the big data technologies and solutions are reported: education and training, cultural heritage, social media and social networking, health care, research on brain, financial and business, marketing and social marketing, security, smart cities and mobility, etc.*

Big data technologies have the potential to revolutionize education. **Educational data** like students' performance, mechanics of learning and answers to different pedagogical strategies, can provide an improved understanding of students' knowledge and accurate assessments of their progress. These data can also help identify clusters of students with similar learning style or difficulties, thus defining a new form of customized education based on sharing resources and supported by computational models. The proposed new models of teaching in [Woolf, Baker and Gianchandani, 2010] are trying to take into account, student profile and performance, pedagogical and psychological and learning mechanisms, to define personalized instruction courses and activities that meets the different needs of different individual students and/or groups. In fact, in recent years, it has been affirmed in the educational the approach to collect, mine and analyze large datasets, in order to provide new tools and information to the key stakeholders in education. This data analysis can provide an increasing understanding of students' knowledge, improve the assessments of their progress, and can help focus questions in education and psychology; such as the method of learning, or how different students respond to different pedagogical strategies. The collected data can also be used to define models to understand what students actually know and understand how to enrich this knowledge, and assess which of the adopted techniques can be effective in whose cases, and finally produce a case by case action plan. In terms of big data, a large variety and variability of data is present to take into account all events in the students' career; the data volume is also an additional factor. Another sector of interest, in this field, is the e-learning domain, where are defined two mainly kinds of users: the learners and the learning providers [Hanna, 2004]. All personal details of learners and the online learning providers' information are stored in specific database, so applying data mining with e-learning can be able to realize teaching programs targeted to particular interests and needs through an efficient decision making.

For the management of large amounts of **cultural heritage** information data, Europeana has been created with over then 20 millions of content indexed which can be retrieve in real time. Each of them was early modelled with a simple metadata model, ESE, while a new and more complete models called EDM (Europeana Data Model) with a set of semantic relationships is going to be adopted in the 2013 [Europeana]. A number of projects and activities are connected to Europeana network to aggregate content and tools. Among them ECLAP [Belllini, Nesi, 2014] is a best practice network, that collected not only content metadata for Europeana but real content files from over then 35 different institutions having different metadata sets and over than 500 file formats. A total of more than 1 million of cross media items is going to be collected with an average of some hundreds of metadata each, thus resulting in billions of information elements and multiple relationships among them to be queried, navigated and accessed in real time by a large community of users [ECLAP], [Bellini, Cenni and Nesi, 2012].

The volume of data generated by **social network** is great and with a highly variability in the data flow over time and space, due to human factor; e.g., Facebook receives 3 billion uploads per month, which corresponds to approximately 3600TB/year. Search engines companies like Google and Yahoo! collect every day trillions of bytes of data, around which real new business is developed, offering useful services to its users and companies in real time [Mislove, Gummandi and Druschel, 2006]. From these large amounts of data collected through social networks (e.g., Facebook, Twitter, MySpace), social media and big data solutions may estimate the user collective profiles and behavior, analyze product acceptance, evaluate the market trend, keep trace of user movements, extract unexpected correlations, evaluate the models of influence, and perform different kinds of predictions [Domingos, 2005]. Social media data can be exploited by considering geo-referenced information and Natural Language Processing for analyzing and interpreting urban living: massive folk movements, activities of the different communities in the city, movements due to large public events, assessment of the city infrastructures, etc. [Iaconesi and Persico, 2012]. In a broader sense by this information is possible to extract knowledge and data relationships, by improving the activity of query answering.

For example in **Healthcare/Medical field** large amount of information about patients' medical histories, symptomatology, diagnoses and responses to treatments and therapies is collected. Data mining techniques might be implemented to derive knowledge from this data in order to either identify new interesting patterns in infection control data or to examine reporting practices [Obenshain, 2004]. Moreover, predictive models can be used as detection tools exploiting **Electronic Patient Record** (EPR) accumulated for each person of the area, and taking into account the statistical data. Similar solutions can be adopted as decision support for specific triage and diagnosis or to produce effective plans for chronic disease management, enhancing the quality of healthcare and lower its cost. This activity may allow detecting the inception of critical conditions for the observed people over the whole population. In [Mans et al., 2009], techniques to the fast access and extraction of information from event's log from medical processes, to produce easily interpretable models, using partitioning, clustering and pre-processing techniques have been investigated. In medical field, especially hospital, run time data are used to support the analysis of existing processes. Moreover, to take into account genomic aspects and EPR for millions of patients leads to cope with big data problems. For genome sequencing activities (HTS, high throughput sequencing) that produce several hundreds of millions of small sequences, a new data structure for indexing called Gkarrays [Rivals et al., 2012], has been proposed, with the aim of improving classical indexing system such as hash table. The adoption of sparse hash tables is not enough to index huge collections of k-mer (sub-word of a given length k in a DNA sequence, which represent the minimum unit accessed). Therefore, new data structure have been proposed, that based on three arrays: the first for storing the start position of each k-mer, the second as an inverted array allows finding any k-mer from a position in a read, and the last records the interval of position of each distinct k-mer, in sorted order. This structure allowed obtaining in constant time, the number of reads that contain a k-mer. A project of the University of Salzburg with the National Institute of sick of Salzburg studies how to apply machine learning techniques to the evaluation of large amounts of tomographic images generated by computer [Zinterhof, 2012]. The idea is to apply proven techniques of machine learning for image segmentation, in the field of computer tomography.

In several areas of **science and research** such as astronomy (automated sky survey), sociology (web log analysis of behavioural data) and neuroscience (genetic and neuro-imaging data analysis) the aim of big data analysis is to extract meaning from data and determine what actions take. To cope with the large amount of experimental data produced by research experiments, the University Montpellier started the ZENITH project [Zenith], that adopts a hybrid architecture p2p/cloud [Valduriez, Pacitti, 2005]. The idea of Zenith is to exploit p2p to facilitate the collaborative nature of scientific data, centralized control, and use the potentialities of computing, storage and network resources in the Cloud model, to manage and analyse this large amount of data. The storage infrastructure used in [De Witt et al., 2012] is called CASTOR, and allows the management of metadata related to scientific files of experiments at CERN. For example, the database of RAL (Rutherford Appleton Laboratory) uses a single table for storing 20GB (which reproduces the hierarchical structure of the file), that runs about 500 transactions per second on 6 clusters. With the increasing number of digital scientific data, one of the most important challenges is the digital preservation and for this purpose is in progress the SCAPE (SCAlable Preservation Environment) project [SCAPE Project]. The platform provides an extensible infrastructure to achieve the conservation of workflow information of large volume of data. The AzureBrain project [Antoniu et al., 2010] aims to explore cloud computing techniques for the analysis of data from genetic and neuroimaging domains, both characterized by a large number of variables. The Projectome project, connected with the **Human Brain** Project, HBP, aims to set up a high performance infrastructure for processing and visualizing neuro-anatomical information obtained by using co focal ultra-microscopy techniques [Silvestri et al., 2012], the solution is connected with the modelling of knowledge of and information related to rat brains. Here, the single image scan of a mouse is more than 1Tbyte and it is 1000 smaller than a human brain.

The task of finding patterns in **business data** in not new, today is getting a larger relevance because enterprises are collecting and producing huge amount of data including massive contextual information, thus taking into account a larger number of variables. Using data to understand and improve business operations, profitability and growth is a great opportunity and a challenge in evolving. The continuous collection of large amounts of data(business transaction, sales transaction, user behaviour), widespread use of networking technologies and computers, and design of big data warehouse and data mart have created enormously valuable assets. An interesting possibility to extract from these data meaningful information, could be the use of machine learning techniques in the context of mining business data [Bose et al., 2001], or also to use an alternative approach of structured data mining to model classes of customers in client databases using fuzzy clustering and fuzzy decision making [Setnes et al., 2001]. These data can be analyzed in order to define prediction about the behaviour of users, to identify buying pattern of individual/group customers and to provide new custom services [Bose et al., 2001]. Moreover, in recent years, the major market analysts conduct their business investigations with data that are not stored within the classic RDBMS (Relational DataBase Management System), due to the increase of various and new types of information. Analysis of web users behaviour, customer loyalty programs, the technology of remote sensors, comments into blogs and opinions shared on the network are contributing to create a new business model called *social media marketing* and the companies must properly manage these information, with the corresponding potential for new understanding, to maximize the business value of the data [Domingos, 2005]. In financial field, instead, investment and business plans may be created thanks to predictive models derived using techniques of reasoning and used to discover meaningful and interesting patterns in business data.

Big data technologies have been adopted to find solutions to **logistic and mobility management** and optimization of multimodal transport networks in the context of Smart Cities. A data-centric approach can also help for enhancing the efficiency and the dependability of a transportation system. In fact, through the analysis and visualization of detailed road network data and the use of a predictive model it is possible to achieve an intelligent transportation environment. Furthermore, through the merging of high-fidelity geographical stored data and real-time sensor networks scattered data, it can be made an efficient urban planning system that mix public and private transportation, offering people more flexible solutions. This new way of travelling has interesting implications for energy and environment. The analysis of the huge amount of data collected from the metropolitan multimodal transportation infrastructure, augmented with data coming from sensors, GPS positions, etc., can be used to facilitate the movements of people via local public transportation solutions and private vehicles [Liu, Biderman and Ratti, 2009]. The idea is to provide intelligent real time information to improve traveller experience and operational efficiencies (see for example the solutions for the cities of Amsterdam, Berlin, Copenhagen, and Ghent). In this way, in fact, is possible in order to use the big-data both as historical and real-time data for the applications of machine learning algorithms aimed to traffic state estimation/planning and also to detect unpredicted phenomena in a sufficiently accurate way to support near real-time decisions.

In security field, **Intelligence, Surveillance, and Reconnaissance** (ISR) define topics that are well suited for data-centric computational analyses. Using analysis tools for video and image retrieval, it is possible to establish alert for activity and event of

interest. Moreover, intelligence services can use these data to detect and combine special patterns and trends, in order to recognize threats and to assess the capabilities and vulnerabilities with the aim to increase the security level of a nation [Bryant et al., 2010].
In the field of **energy resources optimization** and environmental monitoring, very important are the data related to the consumption of electricity. The analysis of a set of load profiles and geo-referenced information, with appropriate data mining techniques [Figueireido, Rodrigues and Vale, 2005], and the construction of predictive models from that data, could define intelligent distribution strategies in order to lower costs and improve the quality of life in this field, another possible solution is an approach that provides for the adoption of a conceptual model for a smart grid data management based on the main features of a cloud computing platform, such as collection and real-time management of distributed data, parallel processing for research and interpretation of information, multiple and ubiquitous access [Rusitschka, Eger and Gerdes, 2010].
In the above overview about some of the application domains for big data technologies, it is evident that to cope with those problems several different kinds of solutions and specific products have been developed. Moreover, the complexity and the variability of the problems have been addressed with a combination of different open source or proprietary solutions, since presently there is not an ultimate solution to the big data problem that includes in an integrated manner data gathering, mining, analysis, processing, accessing, publication and rendering. It would be therefore extremely useful a “map” of the hot spots to be taken into account, during the design process and the creation of these architectures, that helps the technical staff to orient themselves in the wide range of products accessible on internet and/or offered by the market. To this aim, we have tried to identify the main features that can characterize architectures for solving a big data problem, depending on the source of data, on the type of processing required, and on the application context in which should be to operate.

2.2 Open Data

A term that often goes hand in hand with Big Data is Open Data, in fact, looking around is now clear that the data published daily on the web, with the goal of interoperability are growing increasingly; often however the commitment to openness remains implicit. In fact, despite the strong effort, the data are rarely available in such a way as to facilitate its use by third parties.
Within small groups of data, the exchange can be regulated by existing social norms, but in a much larger scale like the web, licenses are required which make explicit the basic conditions for the use of these data.

In the figure below, the possible licenses attributable to Open Data are listed. One thing to remember is that there is a substantial difference between "public" data and "open" data, so it is always good practice to check the type of license associated with the single dataset, before using them.

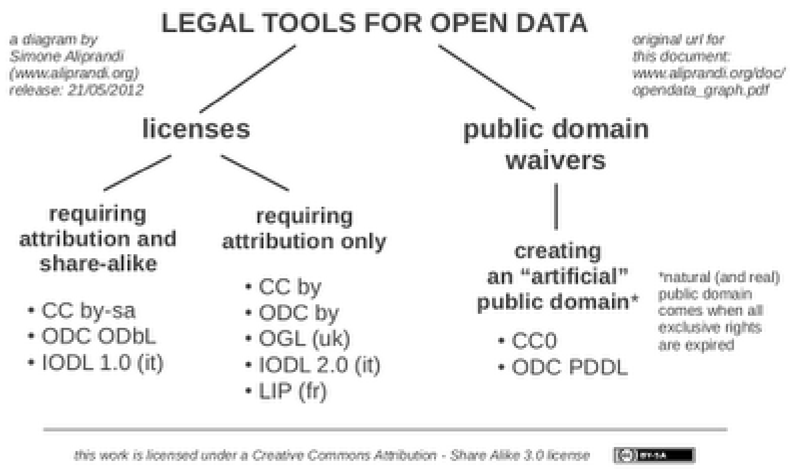

LEGAL TOOLS FOR OPEN DATA
a diagram by Simone Aliprandi (www.aliprandi.org) release: 21/05/2012
original url for this document: www.aliprandi.org/doc/opendata_graph.pdf
licenses
public domain waivers
requiring attribution and share-alike
• CC by-sa
• ODC ODbL
• IODL 1.0 (it)
requiring attribution only
• CC by
• ODC by
• OGL (uk)
• IODL 2.0 (it)
• LIP (fr)
creating an "artificial" public domain*
• CC0
• ODC PDDL
*natural (and real) public domain comes when all exclusive rights are expired
this work is licensed under a Creative Commons Attribution - Share Alike 3.0 license

Figure 1 - Legal tool for Open Data

The data are public when there is an actual public domain waivers, that is a waiver that can be precisely indicated by the specification lack of copyright or the Creative Commons 0 (CC0) [CC0] exactly where it is specified that:

> …"*the person who associated a work with this deed has dedicated the work to the public domain by waiving all of his or her rights to the work worldwide under copyright law, including all related and neighboring rights, to the extent allowed by law.*"

With regard to licensing, the most frequently used for the Open Data can be divided into two groups:

- Licenses that require the attribution of the work to the author and sharing as it is
- Licenses that require only the attribution of the work to the author.

Among the most commonly used licenses belonging to the first group, there is the Creative Common by Share Alike (CC - By SA), with its latest version, i.e. 4.0 [CCSA] that allows to share, that is copy and redistribute the material in any medium or format, and to adapt, to be precise, remix, transform, and build upon the material, per any purpose, even commercially. The most important thing is that appropriate credit must be given, providing a link to the license, and indicated if some changes were made.

The ODbL (Open Database License) is instead a rather complex license but well done, and can effectively implement the copyleft model in the field of databases. It contains a number of contract terms reflecting the model of the licenses Attribution Share Alike proposed by Creative Common. It only license the rights to the database; therefore, in the case of a database containing creative works, in order to ensure a free use of the entire work, it is appropriate to apply another license on the works contained in the database itself.

Finally, one of the most interesting Italian pilot projects in the field of licensing, is the one initiated by the Piedmont Region, which has licensed the IODL - Italian Open Data License, which is also inspired by the model Attribution - Share Alike [ShareAlike].
The most frequently used license among those belonging to the second group, is certainly the Creative Common Attribution (CC By) [CCA] with which the data sharing is permitted, that is copy and redistribute the material in any medium or format and Adapt it through operations such as remix, transform, and build upon the material for any purpose, even commercially; in any case it give appropriate credit is mandatory as well as to provide a link to the license, and indicate if changes were made.

Open Data is a concept that is taking field in various sectors. Certainly in the area of Government is a hot topic, in fact, in recent years, nearly all nations of the world have developed their own "Open Data" strategy in order to increase the transparency and efficiency of the governments themselves,
but above all to have the opportunity to define a new relationship between citizens and public administration.
In addition, the Open Data is a central element in the strategies of e-Government, which is essential to encourage greater transparency in administrative act (thanks to which liability is promoted by providing citizens with information about the activities of public administration),the active participation of citizens in decision-making processes of government to make available online public data, to stakeholders (citizens, organizations, businesses), with a steady increase of datasets exposed to the end of promotion and economic development.
Even though the expected impact is still limited, it is clear that the strategy to publish public data in reusable format, can produce new innovative enterprises.

The Helsinki Region Infoshare is the metropolitan area in the Finnish capital and it is a perfect example of how government can support local innovation through Open Data. The datasets released in the last four years by fourteen governments in the region of Helsinki, are in total 1100 which provide data to researchers and developers across a large number of urban phenomena, from the labor force to transport, up to public and private housing. Making transparent and open part of the daily work of the administrations involved, is a priority of the project, which is gradually changing the way the different departments and municipal services work, by acting as a true "Google Data Helsinki", as stated by the director Asta Manninen.
The Helsinki Region Infoshare allowed through his actions, the birth of a number of applications related primarily to public transport and land management, such as that made by HSY [HSY] that has crossed date related to buildings construction and age of residents to structure policies most effective in environmental terms of energy saving and especially in the suburbs or where there are lower average incomers.
The action of local governments in favor to inclusion and quality of urban life, however, is not confined to encourage the use of new technologies, but to integrate them into its

administrative activities in an innovative way, combining it with an approach to dealing with citizens tense to facilitate the participation of the latter in public life [D'Antonio and Tanskanen, 2014].

Also in the definition process of how scientific data can be published and reused freely, the term Open Data is making strong field. In the area of scientific research there was a great effort to get to define Open data more accurately and certainly this effort will be intensified in the coming years. In this context, the Open Data are essential especially for reuse, that is, the ability to reuse data from other searches, without explicit authorization, in order to aggregate them in other databases, use them during simulations, or for so-called "mashups", i.e. the combination of data from different sources to derive new insights.

Undoubtedly Open Data lead to many benefits. One of the most important is related to interoperability and to maximize efficiency in data usege. Assuming, in fact, a cost associated with the collection of data, this will potentially divided for each application that use it. Especially in the public sector these costs are covered by taxes that citizens pay, and it is therefore proper to give them evidence of the work done by contributions but also give them the opportunity to know and use their data.
The open data also generate innovation, because they are the first step towards the realization of applications not foreseen; innovative ideas are unexpected and open data can unleash their proliferation.
The control over data quality and data correctness is another advantage very attractive: more eyes and more users can locate errors and correct data, even after years of their functional testing.
It is evident that especially in scientific field, where the data contained in articles are considered facts and therefore they are not copyrightable, and where the most important research involving in most cases "Date That belong to the human race", such as, for example, genomes, data on organisms, medical science, environmental data, the definition of Open Data and their use assumes a greater significance. The scientific data are a good of the community and as such should be freely available to those who want to contribute to their improvement. Also in the scientific research, the rate of discovery is accelerated by better access to data [Kauppinen et al., 2011].

2.3 Linked Data

The large publication of Open Data has opened the way of the information sharing. As seen above, most of the Open data are published by governmental organizations, in file formats such as: html, xml, csv, shp, etc., and typically provide information that may present links to web resources. These links are typically coded as un-typed hyperlinks, URLs (Uniform Resource Locators).

Interoperability is one of the most important advantages of open data model, but the data, if isolated, have little value; vice versa, their value increases significantly when different data set, independently produced and published by various parties, can be crossed by third parties. To enable reuse of data, datasets must be able to combine and mix freely. Data must then be linked together, establishing a direct link when the data refer to identical objects (but they come from different sources) or otherwise related to each other.

The World Wide Web has dramatically changed the way we share knowledge, eliminating the barriers to publishing and providing access to documents as part of a global information space.
Despite the indisputable benefits that the web offers, until recently the same principles that have enabled the Web of documents to flourish, have not been applied to the data. Traditionally, data published on the Web are made available as dumps in formats such as CSV or XML, or marked as HTML tables, sacrificing much of its structure and semantics. In traditional Web, the nature of the relationship between two linked documents is implicit, and the data format, or HTML is not expressive enough to allow individual entities described in a paper to be connected via links to related entities.
However, in recent years the web has evolved from a global information space of linked documents to one in which both documents and data are connected. At the base of this evolution, there is a set of best practices for publishing and connecting structured data on the Web known as Linked Data.

In 2006, Tim Berners-Lee published the Linked Data principles [Berners-Lee, 2006], as a model to stimulate the process of making accessible and sharing data as digital resources on the web and from them establishing links with semantically connected sources via URI (Uniform Resource Identifiers) [Bizer et al., 2009]. In other words, Linked Data expresses a method of exposing, sharing, connecting data, via dereferentiable URI, where the term dereferentiable indicates that they provide access to a resource, exactly identified by the URI. The Linked Data are basically data published on the web that machines are able to read and interpret, and the meaning of which is explicitly defined by a string consisting of words and markers.

On this wave, the data publication moved towards the diffusion of Linked Data, opening to the construction of Linked Data repositories and thus for creating a globally connected and distributed data space with integrated semantics. The result of this process is then a network of linked data, exactly, that belong to a domain or initial context, which is in turn connected to other set of external data, of a different domain, within a context of relations more and more extended.

To give you an idea of how this network has evolved over the recent years, an interesting project was realized: the LOD cloud diagram. The image in Figure 2 shows the diagram of the data sets that have been published in Linked Data format, by employees of the Linking Open Data community project and other individuals and organizations. It is based on metadata collected and edited by employees of the Data Hub, as well as on the metadata extracted from a network scan data Linked conducted in April 2014 [DataHub].

During this last inventory, the Linked Data Cloud contained 570 datasets which are connected by 2909 link-sets; the count of the triple has not been updated since 2012, when there were over 52 Billion of Triples. To make a comparison, in 2011 the Linked Data Cloud was formed by 295 data sources that containing over 31 Billion RDF triples, among which 504 million of these triples were links connecting entity described by different data sources [LODCloud]. Thanks to this numbers, it is easy to understand how in the last 3 years, the interest in the Linked Data has grown.

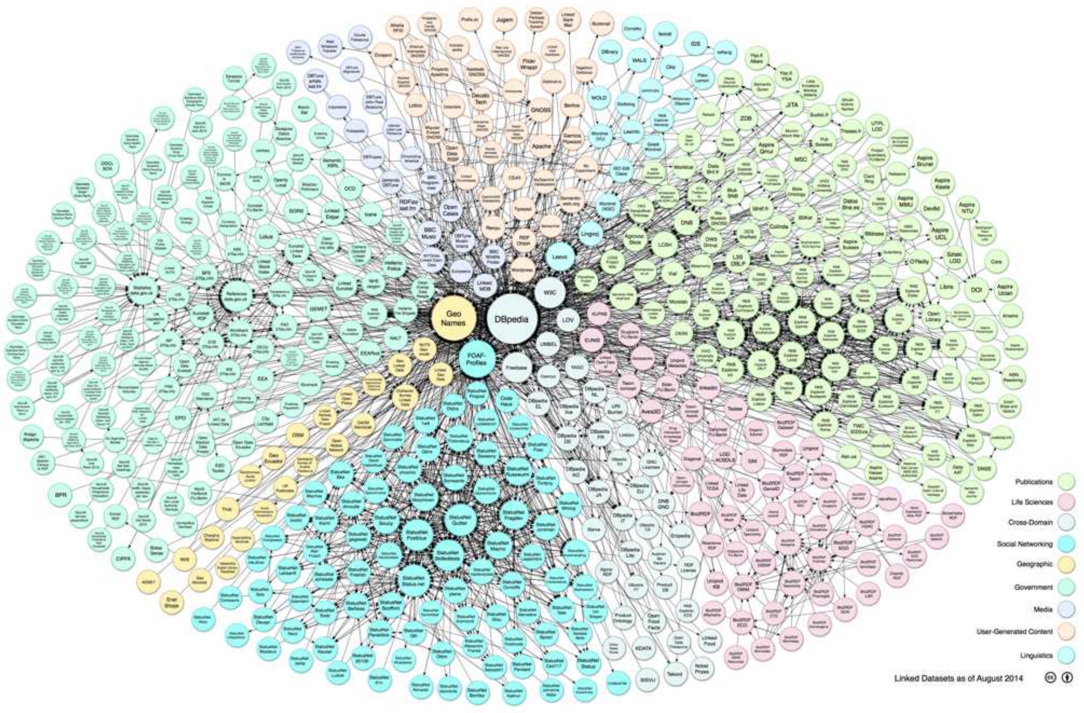


Figure 2 - Linked Data Cloud

Linked Data are based on documents formalized in RDF (Resource Description Framework) [Klyne and Carrol, 2006].
The RDF format is a relative simple but powerful formalism for representing information in triple statements consisting of a subject, predicate and object, where again each of them can be a URI (or an atomic value). As result, the World Wide Web of documents (and Intranets) is complemented with a Web of Linked Data, where

everybody can publish, interlink and enrich data. The more interesting characteristic of this Linked Data Web are the following:

- URIs serve two purpose: identifying “things” ad serving as locators and access paths for information about these things;
- Web of Linked Data is as distributed and democratic as the Web itself;
- Identifiers defined by different people or organizations can be mixed and meshed;
- Linked Data published in various locations can be easily integrated by merging the sets of RDF triple statements , simplifying the data integration;
- The same triple statement formalism is used for defining structure and data.

The adoption of linked data best practices, led to the extension of the Web towards a global data space connecting data from different sectors such as scientific publications, film, music, television and radio, genes, proteins, drugs and clinical trials, online communities, statistical and scientific data.

According to what said so far, it is easy to understand that a concept very close to that of Linked Data, is the Semantic Web, although it cannot be solved with the application of best practices but, for its construction, it requires compliance with relevant rules for the creation of content accessible to automated processes.
Adding semantics to the web involves two things: allowing documents which have information in machine readable formats and allowing links to be created with relationship values. Only when we have this extra level of semantics will we be able to use computer power to help us exploit the information to a greater extent than our own reading.

Semantic Web technologies can be used in various application fields; for example, in data integration thanks to which data in different locations and different formats can be integrated into a single application; in cataloging for describing the content and content relationships available at a particular Web site, page, or digital library; in describing collections of pages that represent a single logical “document”; in resource discovery and classification to provide better, domain specific search engine capabilities; by intelligent software agents to facilitate knowledge sharing and exchange and in many others.
A further contribution of the Semantic Web is having allowed to open the web of data to artificial intelligence processes , to encourage companies, organizations and individuals to publish their data freely, in an open standard format and to encourage businesses to use data already available on the web.

The successful of Linked Data initiative has driven the publication of big, heterogeneous semantic datasets. Data from the governments, social networks, and relating to bioinformatics, are public exposed and interlinking within the web of data.

This can be seen as part of the more general trend of Big Data, addressed in the Paragraph 2.1, where the actual value of data deepens on the knowledge which can be inferred from it, in order to support the decision making process.

Linked Data is the next step for the web. There are multiple benefits over traditional approaches and methodologies of science, such as Business Intelligence, a field in which the topic discussed in this chapter is gaining importance because it is cheaper and faster to implement.
If knowledge is power, with Linked Data's capability, there is the potential to pull information to provide unrivalled context on Business Strategy and not only.

Furthermore, Linked Data will become an integral part of the development of Data Driven Systems architectures that will revolutionize the way we build and maintain information management systems over the next few years.

For example, in the context of Linked Data driven Question Answering systems [Cimiano et al.,2013], which have captured most attention recently, these systems allow users, even with a limited familiarity of technical systems databases, to pose questions in a natural way and gain knowledge of the available data. In addition, there has been a renewed interest from industry in having computer systems not only to analyze the vast amounts of information [Ferrucci et al., 2010] but also in providing intuitive user interface to pose questions in natural language (NL) [Waltinger et al 2013] in an interactive dialog manner [Sonntag 2009, Waltinger et al 2011]: this is only one of the possible applications of semantic web.

An examination of the reference context, both in terms of regulation, as the level of the state of the art and trends at the international level, the LOD represent a necessary and effective tool to enable the development of a concrete semantic interoperability between PA, both nationally and internationally level.
Below a general methodology for the opening of interoperable public data through the LOD is proposed. The proposed methodology takes into account the information obtained during the experience of the PhD, and the "best practices" at the international level.
The proposed approach seeks to maintain a sufficient level of flexibility in order to be adapted to the specific needs of any data "producer". The proposed objective is to generate datasets usable as Linked Data, or dataset in the RDF format containing connections between data themselves and with external dataset. This methodology can be divided into 7 phases:

- Data Identification and selection
- Data Quality Improvement
- Data analysis and data modeling
- Data enrichment

- external linking (interlinking)
- Data validation
- Data publication

The data identification process is definitely the starting point of the process to opening up and linking data. The information assets available can be very large (as in case of PA) or more limited (as in case of scientific research, where often data is only related to a single topic) and then a massive publication of the data in an open format may not be feasible. In this case, select a subset of data becomes necessary, provided that open and transform into Linked Data only selected data, makes sense. These data should be selected taking into account the demand by third parties, any license to which the data relates, and the constraints of privacy.
Data within information systems or archives by an Institution are often "dirty" (sometimes have been designed to be functional, through computer applications, to an internal business processes) and not immediately ready for publication or for the appropriate processing. Therefore, the quality of the data is particularly important because a dirty dataset can make some operations by comparison, similarity and aggregation on the data inefficient or even impractical. Most likely a phase of Quality Improvement must be made which allows the data to acquire the required quality. Some of the most common problems that can be addressed with this process are: incompleteness of the data, different formats, ambiguous meanings, inconsistent data type, lack of correspondence between the names used in the physical patterns and actual data.
The objectives of the third phase, i.e. data analysis and modeling , are precisely the formalization of the conceptual model and the coherent representation of the reference dataset, on the formalized model. This phase represents a logical and conceptual restructuring of the data; in analogy with the methods of software analysis and design, it is clear that a process of re-engineering and refactoring of the base information is triggered.
Both the conceptual model that the dataset at the end of this phase will be represented in RDF.

During the enrichment phase the data, previously reclaimed and molded, are enriched through explicating boundary information (metadata) that simplify reuse and also through the derivation of additional content information, thanks to techniques for automatic information extraction or automated reasoning (inference).
The critical step in the process is the Linking phase, where the information content is linked to other information, which may be other datasets produced by the same source, or data sets already in the Web of Data (WoD). This helps ensure an easier navigation and access to a broader set of data, and provide data at a good level according to the classification of the quality of LOD. [Linked Data Stars].

The next phase is validation: there are different types of validation applicable, but inside the work related to this thesis, the conceptual validation was mainly used, which allows to verify the adequacy of the ontology to the initial and domain requirements.
The last phase is then publication: the fundamental point of this phase is the choice of the publication platform to use, which must be able to integrate themselves, in a flexible way, with any information and organizational system, but above all it must allow punctual access to data with standards such as SPARQL.

SPARQL (*Simple Protocol and RDF Query Language*) is a RDF query language made standard by the *Data Access Working Group*, part of the *W3C* consortium, which has made it an official recommendation on January 2008.
SPARQL is a key element of the semantic web, and it allows to extract information from distributed knowledge bases on the web; practically it search for sub-graphs corresponding to the user's requests, which are made through a query.
A SPARQL query consists of the following parts:

- Prefix declarations, for abbreviating URIs;
- A result clause, identifying what information to return from the query;
- Dataset definition, stating what RDF graph(s) are being queried;
- The query pattern, specifying what to query for in the underlying dataset;
- Query modifiers, slicing, ordering, and otherwise rearranging query results.

The prefix declarations are identified by the PREFIX statement, the result clause is instead identified by the SELECT statement, the dataset definition is identified by the FROM statement, the query pattern is identified by the WHERE statement and finally, at the bottom, there are the query modifiers.
SPARQL queries are based on the "pattern matching" mechanism and in particular on the "triple pattern" construct. Most SPARQL query forms contain a series of triple patterns called *basic graph pattern*. A *basic graph pattern* corresponds to a sub-graph of RDF data where the terms of this RDF sub-graph, can be replaced with variables.
The SPARQL query output, instead, can be of several types: usually their results are results sets or RDF graphs, which mainly come from queries of existential type ("exists or not, the researched sub-graph?"), or tabular ("lists the possible results"); the results can be of various formats and the most common are XML, JSON, CSV, HTML and RDF.
SPARQL adopts the syntax Turtle, an extension of N-Triple, a very concise and intuitive alternative to traditional RDF / XML.
In the following exemple the realized RDF triplestore is queried asking to find all subjects *(?roadEle*) and objects (*?length*) put in relation through the *km4c:length* predicate.

```
Select ?roadEle ?length WHERE {
```

```
GRAPH
      <http://www.disit.org/km4city/resource/GrafoStradale/Grafo_strad
ale_Firenze> {
      ?roadEle  a km4c:RoadElement .
}
      ?roadEle km4c:length ?length .
}
ORDER BY ASC (?length)
```

SPARQL variables start with a *?* and can match any node (resource or literal) in the RDF dataset. The *WHERE* clause define the triple pattern to which the variables must match. The GRAPH clause, instead, restricts the search to the specified node graph and the *SELECT* clause returns a table of variables and values that satisfy the query. Finally, the ORDER BY clause allows to display the results sorted according to the specified variables. To retrieve multiple properties about a particular resource is possible to use multiple triple patterns.

Part II
Discussion of the Work

Chapter 3

3. Private & Public Data available

In this chapter some of the available datasets will be analyzed, in order to make very clear what is the data heterogeneity on which the research project is based, and the problems that each one presents.
With regard to Open Data, the description will be limited to only a dataset per type, given the high number of available data sets, but it will be shown a list of all used datasets, from each different source.

3.1 Street Guide

Through the portal *Osservatorio dei Trasporti* of Tuscany, the data belonging to the Tuscany road graph were downloaded. These data constitute the backbone of the entire project and provide a representation of the entire regional road network, which is organized as a graph, i.e., formed by arcs (the so called road elements) and nodes (or junctions). To this basic structure a variety of information is then associated such as street names (which are nothing more than set of road elements grouped within individual municipalities), addresses (using information related to access and house numbers), etc.
Since data on transport infrastructure (roads, railways, house numbers) do not change frequently, the *Osservatorio dei Trasporti* does not provide the ability to download the data via web services, but only via a web interface, after authentication using a certificate issued by the same office. The downloading process of these data will therefore not be automated within the structure that will be built.
The data contained into the archives recovered from the portal of the *Osservatorio dei Trasporti* are manifold; in particular, in the vision released during the development of the project, there are two main categories of data: the Data Pack Address Book (DPI) and the Data Pack Accessory (DPA). The DPI contains all the information necessary to constitute the road graph; among these are:

- **Road elements**: linear entities bounded by two junctions. Road Elements are the basic component of the entire Road Graph, and consist of an ordered set of points.

- **Junctions**: points of intersection between the axes of two Road Elements. From the geometric point of view, a junction is a punctual entities (also called Node) represented by a pair of coordinates.
- **Toponyms**: from the definition recovered in the manual provided by the Tuscany Region, a Toponym corresponds to a portion of the mobility road network, to which a name is assigned from a municipality, i.e. the name of the street that is part of the municipality. In practice, the Toponym entity is defined as a set of Road Elements of the same city, aggregated according to the name assigned by the municipality.
- **Administrative Roads**: like a toponym, an Administrative Road is a set of road elements. However, in the specific case of Administrative Road, the grouping is performed based on administrative criteria. For more details on the difference between the Toponyms and the Administrative Roads, see Section 4.4.1.
- **Street numbers**: as easy to understand, the element Street Number is used to define a specific address, within a given street. In fact, contrary to what one might imagine, the Street Number entity is not directly associated with any geolocating component; in fact, to this aim, at least an Entry element is connected to a single Street Address, with a pair of coordinates associated.
- **Entry**: punctual elements (i.e. with geographic coordinates) that identify access to a specific place of residence or business located in a certain number Civico.
- **Milestone**: point element that contains the value of the mileage of an Administrative Road, calculated from a starting point.

The DPA contains instead the following information:

- **Limitations of Access** to Elements Road;
- **Forbidden** (or allowed) **turns** .

Inside archives provided by Osservatorio dei Trasporti, files containing geographical information are encoded in *ESRI Shapefile* or file with the extension *.shp*, which are in turn necessarily linked to file *.shr* and *.dbf*. The shapefiles are vector files containing information mainly used for geographic information systems: the*.shp* file preserves the geometry, the *.shx* file instead stores the geometry index, while inside the *.dbf* file, the list of objects, together with all their attributes, is present.
Data types that can be stored within a Shapefile are **Points** (used for example for the class Junction), **Poly-lines** (set of connected and ordered broken lines, such as the Road Elements) and **Polygons** (used for example to representing the extension of a municipality or a Province).
All other data, i.e. those who do not have geographic information shall be issued only with *.dbf* extension, a format via which xBase databases are saved. These files are easily readable by spreadsheet software such as *Microsoft Excel* or *OpenOffice Calc*.
With regard to the data composition, there are two distinct types of tables: tables with the actual data that are indicated by the initials GIA (e.g. *GIA_ACCESSO.dbf*,

GIA_EL_STRADALE.shp, etc.), while files whose initials are DOM contain domain tables, within which all the possible values that can have a particular attribute, are stored. For example, in the table *GIA_GIUNZIONE.dbf*, the attribute TIP_GNZ is in each record, and indicates the type of junction in numeric format.

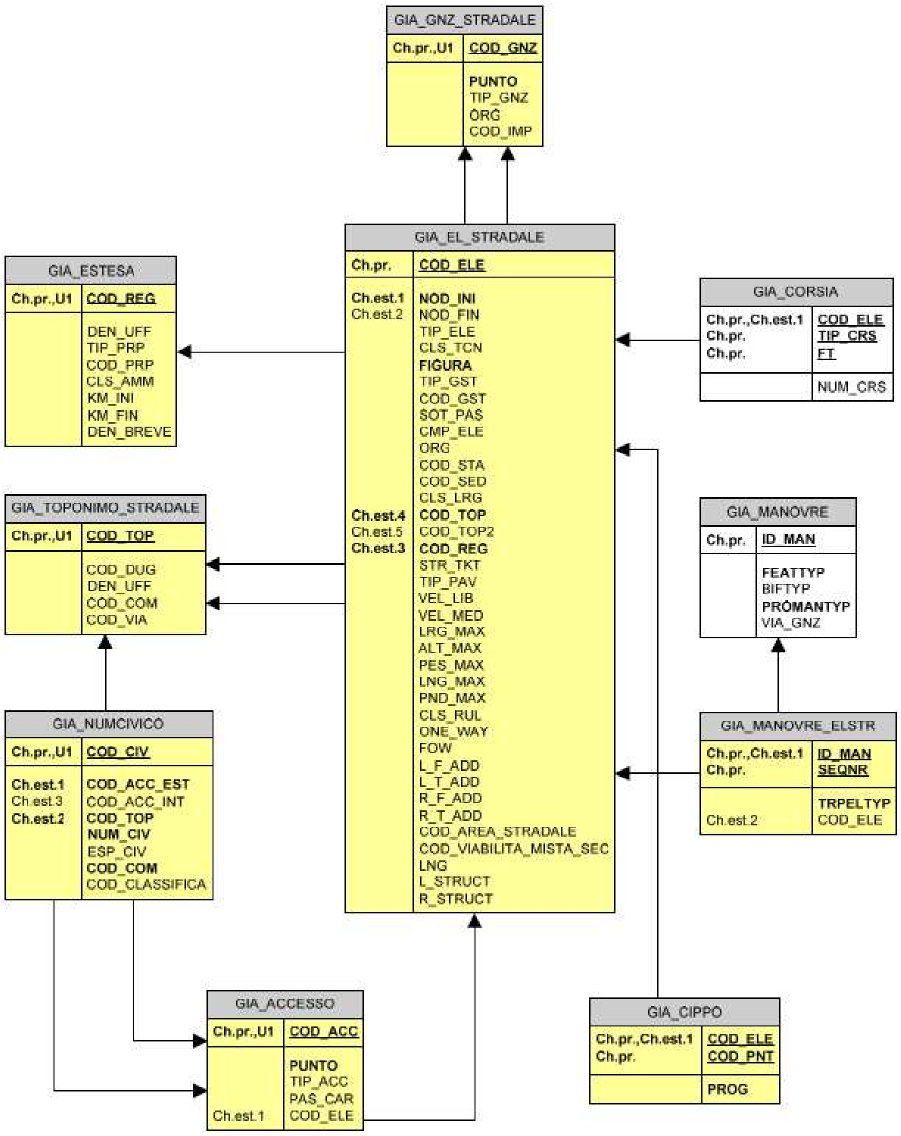


Figure 3 - Street Guide logical model

Inside the corresponding file *DOM_TIP_GNZ.dbf*, there are all the values that can take this field, with the its text description. Below in Table 1 the content of the latter table is shown.

VALUE	DESCRIPTION
0100	intersezione a raso / biforcazione
0200	casello autostradale
0300	minirotatoria (raggio di curvatura < 10m)
0400	variazione di sede/sottopasso (SOT_PAS, COD_SED)
0900	passaggio a livello
0500	terminale (inizio o fine elemento stradale)
0600	cambio toponimo / titolaritÓ / gestore
0700	variazione classe di larghezza (CLS_LRG)
0800	area di traffico non strutturato
5106	variazione composizione
5201	nodo intermodale per ferrovia
5203	nodo intermodale per aeroporto
5202	nodo intermodale per porto
5301	limite di regione
99	nodo fittizio (limite di ArcInfo)
1000	nodo di supporto(loop)
1100	variazione classifica tecnico-funzionale
1200	variazione stato di esercizio (COD_STA)

Table 1 - DOM_TIP_GNZ content

In Figure 3 is possible to see part of the *Street Guide* composition.

3.2 Railway Graph

The Tuscany Region provides details of the rail network that extends throughout its territory, through the portal of the *Osservatorio dei Trasporti*. The rail graph has the following basic entities:

- **Rail element**: a linear entity delimited by two nodes, identified by an ordered set of points. Generally represents the axis of the railway line which takes place on the movement of trains. Contains the structural characteristics of railway infrastructure such as the power type, gauge, number of tracks, etc.
- **Railway junction**: it represents a point of intersection of two or more elements rail. A junction is always a punctual entity (node), represented in geometric terms by a pair of coordinates. The junction can be described as well as a fork (fork/confluence), as the presence of a station/good yard/stop/rail toll, the beginning/end of a line and the variation of the characteristics of the railway infrastructure.

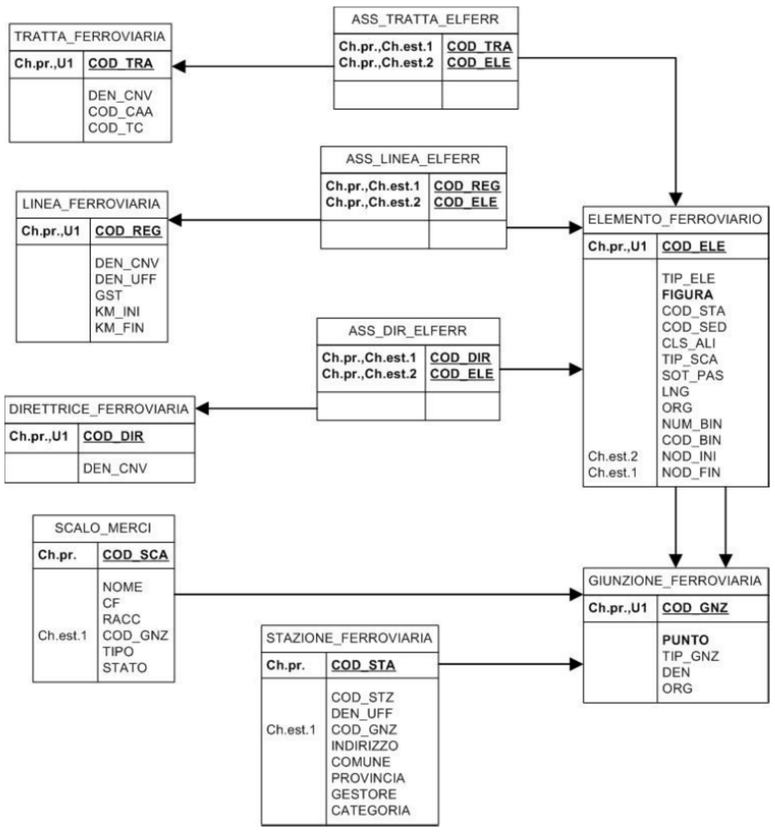


Figure 4 - Railway Graph logical model

In addition to the basic entities there are also:

- **Railway Direction**: a railway line is obtained as a set of rail elements through the association Ass_Direttrice_ElFerr.
- **Railway line**: even a railway line is obtained as a set of rail elements, but thanks to the association Ass_Linea_ElFerr.
- **Railway Section**: as the previous ones, a railway section corresponds to a set of rail elements through the association Ass_Tratta_ElFerr.
- **Train station**: this entity contains all information about a train station. The railway stations refer to a railway junction.
- **Good yard**: it corresponds to a cargo terminal. The good yard refer also to a railway junction.
- **Ass_Direttrice_ElFerr**: it represents the relationship table (n:m) between the Railway element and the Railway direction.
- **Ass_Linea_ElFerr**: it is the relationship table (n:m) between the Railway element and the Railway Line.
- **Ass_Tratta_ElFerr**: it is the relationship table (n:m) between the Railway element and the Railway section.

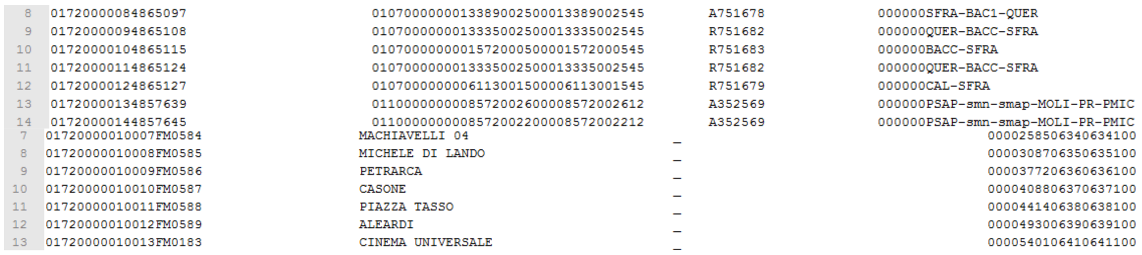
In Figure 4 is possible to observe the logical model of the Railway Graph.

3.3 Bus network

Through the portal of the *Osservatorio dei Trasporti*, data relating to the entire bus network are available, which interconnect all of Tuscany. Through this portal it is possible to get a list of all bus routes and all stops that make them, the paths followed for each line, as CSV files. In addition to these files, thanks to the ATAF contribution, data related to the scheduled rides that including departure and arrivals times, re also available, of course, limited to the part of the network managed by ATAF itself; this data can be downloaded in *.rar* format, always within the portal of the *Osservatorio dei Trasporti*. Within the *rar* package there are many files, among which the most intreresting are the following:

- RT_DTORA.txt
- RT_HDORA.txt
- RT_NODI.shp
- RT_ITIN.shp

The two *shape* files contain information relating respectively to the nodes, i.e. ATAF bus stops, which also include the coordinates that allow to geolocalize each stop, and paths information, which instead including a set of coordinates that identifies the whole path. Particularly interesting are the other two *TXT* file that is *RT_DTORA* and *RT_HDORA*, within which are encoded data about ATAF scheduled time, along with all stops made by each individual ride. Since these files all active race codes, can then be recovered, which will then be used to query the MIIC web server, which provides real-time information on bus transits.
All this information is encoded in the two *TXT* files, as is possible to observe from Figure 5, where a portion of data extracted from each file, is shown.



```
01720000084865097           010700000001338900250001338900254 5   A751678        000000SFRA-BAC1-QUER
01720000094865108           0107000000013335002500013335002545    R751682        000000QUER-BACC-SFRA
01720000104865115           0107000000015720005000015720005 45    R751683        000000BACC-SFRA
01720000114865124           0107000000013335002500013335002545    R751682        000000QUER-BACC-SFRA
01720000124865127           0107000000006113001500006113001545    R751679        000000CAL-SFRA
01720000134857639           0110000000008572002600008572002612    A352569        000000PSAP-smn-smap-MOLI-PR-PMIC
01720000144857645           0110000000008572002200008572002212    A352569        000000PSAP-smn-smap-MOLI-PR-PMIC
0172000010007FM0584        MACHIAVELLI 04              _                     00002585063406341 00
0172000010008FM0585        MICHELE DI LANDO            _                     0000308706350635100
0172000010009FM0586        PETRARCA                    _                     0000377206360636100
0172000010010FM0587        CASONE                      _                     0000408806370637100
0172000010011FM0588        PIAZZA TASSO                _                     0000441406380638100
0172000010012FM0589        ALEARDI                     _                     0000493006390639100
0172000010013FM0183        CINEMA UNIVERSALE           _                     0000540106410641100
```

Figure 5 - RT_DTORA and RT_HDORA content

Analyzing the first part of the figure, relative to *RT_HDORA.txt* file, the following information can be retrieved (the numbers at the beginning of each line indicate the starting and ending position of each fields inside the file):

- 000-003: It contains the company code (regional coding);
- 004-009: It represents the ride unique progressive, within the flow;
- 010-029: Ride company code;
- 030-039: Not used; must contain blank space;
- 040-043: It is the managing body code (regional coding);
- 044-047: Not used; must contain “0000”;
- 048-055: It is the Length of the entire path (expressed in meters);
- 056-059: It corresponds to the total ride duration (in minutes);
- 060-067: Accounting length (in meters);
- 068-071: Ride accounting duration (in minutes);
- 072-081: It is the line holding code;
- 082-082: A=Outbound R=Return;
- 083-102: It is the company unique code of the path;
- 103-108: Not used; must contain “000000”;
- 109-228: it defines the brief path description.

From the second file corresponding to the second part of the above figure, extracted from *RT_DTORA.txt* file, the following information can be retrieved:

- 000-003: It contains the holding code (regional coding);
- 004-009: It represents the ride unique progressive, within the flow; the *COMPANY* and *PROG_CORSA* fields establish a join with the *RT_HDORA* flow;
- 010-013: It is the stop progressive number within the path;
- 014-023: It represent the stop company code;
- 024-029: Not used; must contain “000000”;
- 030-035: Not used; must contain “000000”;
- 036-039: Not used; must contain “0000”;
- 040-079: It contains the stop name;
- 080-119: It specifies the stop location;
- 120-127: It corresponds to the progressive distnace from the ride start terminus in meters;
- 128-131: It is the arrival time at the bus stop; for starting terminal it is "9999"
- 132-135: It is the departure time from the bus stop; for arrival terminal it is "9999";
- 136: It indicates the main stops of the path;
- 137: It indicates the optional stops of the path;

- 138: It indicatesthe bus stops from where the bus passes without stopping.

It is evident that the information relating to the bus network managed by ATAF, need to be handled differently, compared to information from other public transport companies, through the use of two different ingestion processes [Bellini, Nesi, Rauch, 2014B], that will map information on the same macroclass of the ontology designed for this thesis.

3.4 Open Data

The available data during the design phase of the architecture, come from different sources, and this implies different formats and different methods of access to this data. Below, the main datasets and their fundamental characteristics will be analyzed.
All the Open Data set coming from two different sources, the Tuscany Region's Open Data Portal (http://dati.toscana.it/), which provides over 160 open data set, and the Municipality of Florence's Open Data Portal (http://opendata.comune.fi.it/) through which is possible to access to over 650 open data set.
The resources accessible through these two portals belong to different categories, and in particular there are files that contain information related to public administration, infrastructure and transport, environmental data, weather, city planning, statistical data, data regarding social life and so on, and the formats, in which these elements are provided, are various, in particular, there are file of type CSV, XML, KMZ, JSON, PDF, RDF and SHP.
Given the high number of datasets available as Open Data, only part of them were selected, in the initial stage: from the Tuscany Region Portal, 24 files in CSV format were examined, containing information about various places of interest (POI) like public offices, pharmacies, restaurants, schools, hotels, universities, each of them provided with an address formed by street name and house number. A further set of examined files, always coming from the portal of the Tuscany Region, includes 286 XML files relating to the weather forecast for each municipality in Tuscany, provided by the consortium LaMMA (http://www.lamma.rete.toscana.it/); in addition to forecasts, these files contain some physical nature's information such as temperature, intensity of UV rays and humidity.
From the City of Florence Portal, 6 CSV files were initially analyzed, containing different statistics on the area, which contain purely static information. These datasets are in fact related to a specific year, and then most likely in the future they will not be updated, but they will only expanded with the data for the new year. Moreover, from the portal of the city of Florence, an additional resource, provided in KMZ format, has been analyzed, covering the only tram line currently on the territory of Florence.

3.4.1 *Services dataset*

The Tuscany region offers a number of files in CSV format, that contain information to geographically locate a range of services (public offices, businesses and other services) through the use of the address. The delimiter used in the CSV file is a comma, while the addresses are enclosed in quotation marks in order to maintain together street name and street number. As mentioned previously, for reasons of priority, at the time only 24 dataset have been analyzed:

- **Art and culture**: this dataset contains information about museums, galleries, monuments, theaters and libraries;
- **Banks**: it contains information about banks in the Tuscan area;
- **Express Couriers**: the dataset contains information about offices of courier services;
- **Emergencies**: it contains information on police stations, carabinieri, fire brigade, civil protection, police and financial police;
- **Food and Wine**: this dataset contains information on a selection of restaurants, pizzerias, pubs, bars and catering companies;
- **Pharmacies**: it contains information relating to pharmacies located in the Tuscan territory;
- **Companies Trade**: a dataset that contains information on shopping centers, large non-food distributions, clothing outlets and hypermarkets;
- **Airlines Infrastructure**: it contains information on civil airports, airfields and helipads;
- **Kindergarten**: the dataset contains information about private preschools, private and public infant schools;
- **Elementary School**: it contains information on public and private elementary schools;
- **Middle School**: the dataset contains information on public and private junior high schools;
- **High School**: it contains information on public and private high schools;
- **Language courses and training schools**: a dataset that contains information about language courses and training schools;
- **Sport in Tuscany**: it contains information on sports facilities, gyms, racetracks, sports schools, sailing schools, ski schools, equipment for the golf, climbing associations;
- **Health and Healthcare**: the dataset contains information about hospitals, emergency room, medical guards, public assistance, private clinics, local health authority offices, health reservation centers, and dentists;
- **Road transport services**: it contains information on rest stops, camper service, vehicle rental and repair shops;
- **Various services**: a dataset that contains information about income revenue authority, youth information centers, employment centers, social security service offices and department of motor vehicles;

- **Accommodations**: it contains information about cottages, bed and breakfast, camping, guest houses, hotels and residences;
- **Accommodations with geolocation**: it contains very similar information to the previous dataset with the addition of coordinates for geolocalize each structure;
- **Leisure**: the dataset contains information about cinema, nightclubs, parks natural, game rooms and recreation rooms;
- **Various offices**: it contains information about consulates, civil registry, prefectures and other public offices;
- **Universities and conservatories**: the dataset contains information about universities and conservatories;
- **Guided Tours**: it contains information on associations that organize guided tours, tour operators and private tour guides;
- **Welfare**: this dataset contains information on social workers, nursing homes, senior centers, rehabilitation centers, dining hall and community.

The structure of all these datasets is very similar, in fact, only 2 of the 24 datasets have additional fields in addition to the 12 listed below:

- *Id*: is an integer of 6 digits which uniquely identifies each row of the file, and then each object of interest. A peculiarity of this field is in the fact that it is unique, and it is not possible to find 2 equal id also on different files, so it could be used as a key, for a database table.
- *Url*: contains a web url that points to an information page on the site http://www.intoscana.it, related to the each object.
- *Name*: contains the legal name of the place of interest. Unfortunately there is no standard syntax, so the name can be uppercase, lowercase, enclosed in single quotes and any other combination of styles.
- *Category*: at regional level, all services were classified into different categories; this field specifies the category of each service.
- *Phone* and fax: these 2 fields contain respectively the telephone number and the fax number; unfortunately also in this case, there is a lack of writing convention: often the prefix is missing, sometimes, if there are two similar telephone numbers, only the last different digit are reported (especially when referring to an office).
- *Address*: the field contains the address of the referred service. Unfortunately also this field presents a high number of imperfections: almost in all services, the postal code is missing, or the dug associated to the street name is written in a lot of different ways like "Piazza", "piazza", "p.za" or "p.zza". As we shall see in later chapters, to overcome these lacks of uniformity in writing, a reconciliation phase will be carried out.
- *City*: this field indicates the city, or the municipality in which the service is located.

- *Provinces*: string formed by 2 uppercase letters that indicate the abbreviation of the province.
- *Website* and email: this 2 fields contain respectively, the website url and the email address of each services. They are often empty.
- *Notes*: a field that contains various type of information; it is a text field variable in format and length, which is often empty.

In addition to these fields, the *Enogastronomia.csv* file, contains an additional field (*closing days*) that, as the name suggests, indicates the restaurant's day of closing, and the file *Turismo-accoglienza-georeferenziazione.csv* contains the pair of coordinates that identify the location of each property:

- Closing days: this field indicates the service weekly day of closure. To express this information a unique notation is used: 3 capital letters to indicate each day, enclosed by brackets, eg. *(LUN).*
- Latitude and longitude: this field contains the geographical coordinates in WGS84format.

3.4.2 *Weather Forecast Dataset*

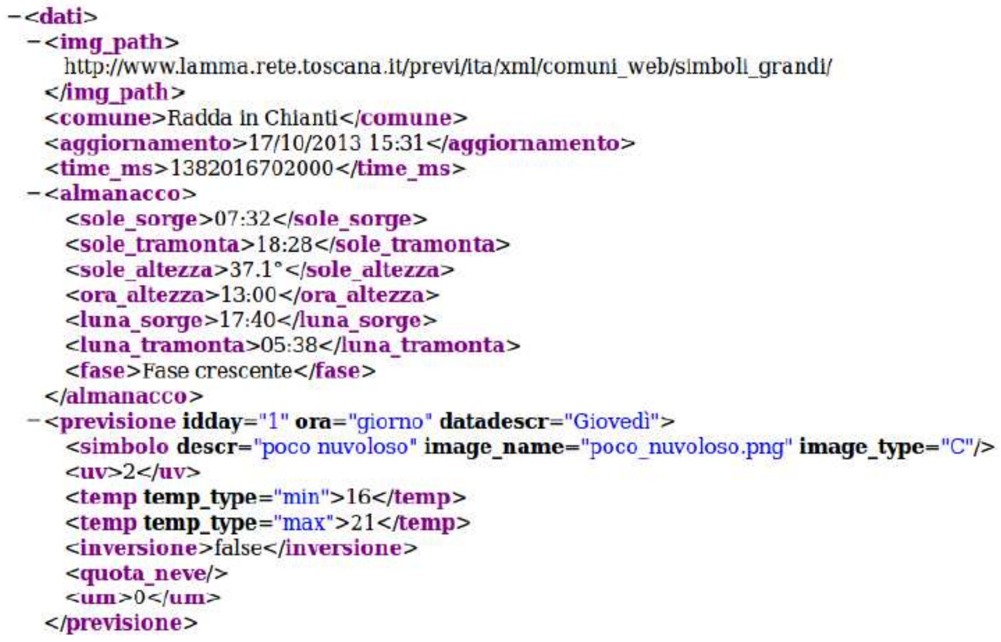
The Tuscany region provides information about the weather forecast throughout the region. In particular provides one XML file, for each of the 286 municipalities which shows information of various nature, updated twice a day.
Each file is structured in two sections, a first section containing a variety of information related to the today's date and the main section containing instead a series of predictions regarding also the next four days.
In particular, for the current date there are some fields that describe the characteristics relative to the sun and the moon as the time of sunrise, the time of sunset, the maximum height of the sun and the time of its achievement.
The forecasts section contains different information for each day, based on the proximity with today's date
In the following table is possible to observe which information are provided for each days of the five interested by forecast.

Timing	Involved Days	Fields
Day	Day 0, Day 1, Day 2, Day 3, Day 4	• Minimum Temperature • Maximum Temperature • Humidity • UV rays • Rain • Wind direction
Night	Day 1, Day 2	• Average Temperature • Perceived Temperature • Humidity • Rain

		• Wind direction • Snow level
Morning	Day 0, Day 1, Day 2	• Average Temperature • Perceived Temperature • Humidity • Rain • Wind direction • Snow level
Afternoon	Day 0, Day 1, Day 2	• Average Temperature • Perceived Temperature • Humidity • Rain • Wind direction • Snow level
Evening	Day 0, Day 1, Day 2	• Average Temperature • Perceived Temperature • Humidity • Rain • Wind direction • Snow level

Table 2 - Information provided per day inside weather forecast

Note: The fourth and fifth day from the present one, only contain rain and winds forecasts. The forecast for each days of the week are provided in two ways: first each day is divided into two section, night and day, and for the day most closet to taday, is used also the division in Morning Evening and Night.



```
-<dati>
  -<img_path>
     http://www.lamma.rete.toscana.it/previ/ita/xml/comuni_web/simboli_grandi/
   </img_path>
   <comune>Radda in Chianti</comune>
   <aggiornamento>17/10/2013 15:31</aggiornamento>
   <time_ms>1382016702000</time_ms>
  -<almanacco>
     <sole_sorge>07:32</sole_sorge>
     <sole_tramonta>18:28</sole_tramonta>
     <sole_altezza>37.1°</sole_altezza>
     <ora_altezza>13:00</ora_altezza>
     <luna_sorge>17:40</luna_sorge>
     <luna_tramonta>05:38</luna_tramonta>
     <fase>Fase crescente</fase>
   </almanacco>
  -<previsione idday="1" ora="giorno" datadescr="Giovedì">
     <simbolo descr="poco nuvoloso" image_name="poco_nuvoloso.png" image_type="C"/>
     <uv>2</uv>
     <temp temp_type="min">16</temp>
     <temp temp_type="max">21</temp>
     <inversione>false</inversione>
     <quota_neve/>
     <um>0</um>
   </previsione>
```

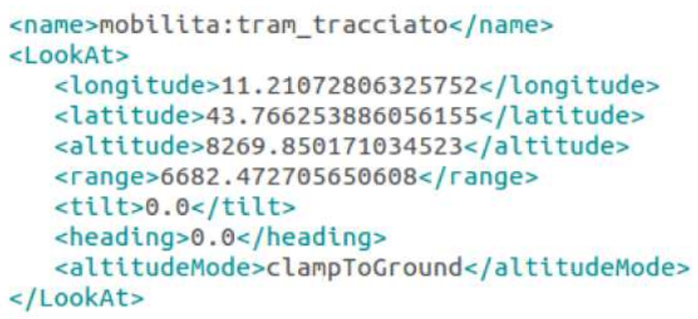
Figure 6 - Web service response for weather forecast

Inside the starting section of each XML file, the following XML tag can be find:

- *Comune*: this tag indicates the municipality to which the forecast refers;
- *Aggiornamento*: this tag saves the creation time of the XML file;
- *time_ms*: this tag contains a timestamp, i.e. the number of milliseconds from 1970 to the time contained into the tag *Aggiornamento*;
- *Almanacco*:
    - *sole_sorge*: this tag contains the sunrise time;
    - *sole_tramonta*: it contains the sunset time;
    - *sole_altezza*: the tag indicates the maximum height of the sun in degrees;
    - *ora_altezza*: this tag provides time when the sun is at its greatest height;
    - *luna_sorge*: it corresponds to the time when the moon rises;
    - *luna_tramonta*: the tag indicates the time when the moon sets;
    - *fase*: the tag specifies whether the moon is in ascending or descending phase;
- *Previsione*:
    - *idday*: the contents of the tag is a number from 1 to 5 that represents the day of the week to which the forecast refers (1 and 5 is to present the fifth starting today);
    - *ora*: the contents of the tag indicates one of the 5 times of the day provided (day, morning, afternoon, evening or night);
    - *datadescr*: this tag contains the name of the weekday (Monday - Sunday);
    - *simbolo-descr*: this tag contains the description of the symbol used for the prediction of rain and winds.;
    - *simbolo-image-type*: the tag indicates the type of information expressed in the previous tag. If equal to "C", the information concerning the rain forecast, and if it is equal to "W" it refers to the wind.;
    - *uv*: the tag contains an integer that indicates the power of ultraviolet rays;
    - *temp_type="min"*: the tag contains information about the minimum temperature in Celsius;
    - *temp_type="max"*: the tag contains information about the maximum temperature in Celsius;
    - *temp_type=""*: this tag contains information about the average temperature in Celsius;
    - *temp_type="perc"*: the tag contains information about the perceived temperature in Celsius;
    - *quota_neve*: the content of the tag is the height above the sea level (in meters) at which it is possible to find snow.;
    - *um*: the content of the tag is an integer that indicates the percentage of humidity in the air.

3.4.3 *Tram Line Dataset*

The City of Florence's Open Data Portal provides a resource that describes the path covered by the unique tram line in Florence. The file is provided in KMZ format, which is a compressed version of a KML (keyhole markup language) file. KML (*Keyhole Markup Language*) is a lightweight XML-based language schema used by *Google Earth* and *Google Maps*. KML specifies only a basic set of features commonly used in 3D GIS with the possibility to call data from network resources or to point to network resources. In addiction it can call geometry described by a COLLADA (.dae) file and offers ways to specify custom schema features. The KML file specifies a set of elements (geographical bookmarks, images, polygons, 3D models, textual descriptions and labels) to be displayed in *Google Earth*, *Maps* and mobile. Each location must have a longitude and a latitude. The coordinates provided in these files comply with the standard WGS84 (World Geodetic System of 1984) [W3C geo].



```
<name>mobilita:tram_tracciato</name>
<LookAt>
   <longitude>11.21072806325752</longitude>
   <latitude>43.766253886056155</latitude>
   <altitude>8269.850171034523</altitude>
   <range>6682.472705650608</range>
   <tilt>0.0</tilt>
   <heading>0.0</heading>
   <altitudeMode>clampToGround</altitudeMode>
</LookAt>
```

Figure 7 - Tram line KML file

The files in KMZ format always begin with a series of information used by Google Earth to determine the location of the initial view; this information are closed into the LookAt tag.

Further this first section, is possible to find, into the file, the section containing the geographical reference of the element concerned. This section changes with the type of element that represents, among the 4 possible:

- Punctual element, i.e. each resource is identified by a point;
- Linear element, i.e. element composed by a set of coordinates;
- Areal element, i.e. a string of coordinates where the first and the last position coincide, and it can also be formed starting from disjoint areas.
- Mixed element, that may contain a variable number of positions of any previous types.

Following is the list of top tags contained in the KMZ file and their meaning:

- Name: this tag contains an identifier of the file. Usually corresponds to the file name and it does not have a standard form;

- Placemark: it is the tag that contains fields that indicate information about the placeholder that will be drawn on the map;
- Description: this tag is inside the Placemark tag and it contains a number of variable information. This information is written in HTML, and to avoid interference with XML, the CDATA command is used, which allows to write a text in any format, without the XML parser try to interpret it.
- LookAt: the LookAt tag contains a number of fields that inform Google Earth on what is the position to be taken; these parameters indicate the user's position relative to the object.
    - Longitude and latitude: this 2 tags indicate latitude and longitude expressed according to the WGS84 notation;
    - Altitude: height in meters. The reference point for this measurement depends on altitudeMode tag described below;
    - Range: a tag that contains the distance between the point specified by latitude, longitude, altitude and the position of the observer (in meters, Figure 7);
    - Tilt: it contains the angle between the direction of the viewpoint (LookAt) and the normal to the Earth's surface (see Figure 7). This field can take values from 0 to 90 degrees and it cannot have negative values;
    - Heading: this tag identifies the direction of the point of view expressed in degrees (default value is equal to zero, which corresponds to the north) and its values ranging from 0 to 360 degrees;
    - AltitudeMode: a tag that specifies how the altitude must be interpreted: it can assume 3 values, i.e. *clampToGround* that specifies to not consider the altitude value contained in LookAt, *relativeToGround* that interprets the altitude as a value in meters above ground level and, finally, *absolute* that interprets the altitude value in meters from sea level.
- IconStyle:
    - ColorMode: can take only two values, *random* thank to which the color of the placemark is randomly assigned or *normal* that has no effect on the indicator;
    - Icon/href: this tag includes an HTTP address or a local path used to load an icon;
- LabelStyle: the value contains in this tag specifies how the name of a feature is drawn on the 3D viewer. It can indicate the color, the *colorMode* or the scale of use;
- Point: it indicates a geographic location defined by latitude, longitude and altitude (optional);
- MultiGeometry:
    - LineStrings: this tag defines a set of segments connected with one another. The set is formed by two or more coordinate separated by space;

   - Polygon: a tag to defines an exterior boundary and zero or more interior boundaries. Each border is composed in turn of a *LinearRing*. All coordinates are expressed in a clockwise direction;
   - LinearRing: with this tag is possible to represents a closed line; it is represented by a set of coordinates where the first corresponds with the last.

3.4.4 *Statistics Dataset*

The city of Florence offers a long series of files in CSV format, through its Open Data Portal, among which 6 files with different content from each other, were selected and analyzed during the initial phase of this research project. Below, a brief description of each file, is provided:

- Public door access and taxi: a dataset that contains the number of access to public door and taxi from 2009 to 2011, split per month;
- Resolutions: this dataset contains all the Resolution of the Regional Council;
- Tourist arrivals: a dataset that contains the number of tourist arrivals to the city of Florence’s accommodation, by year and nationality of origin, from the year 2006 to 2010;
- Afaf: it contains the number of tickets sold, season tickets sold, number of passengers and the ATAF network length in Km inherent to the public transport service, per year; this dataset contains data from 2001 to 2010;
- Car accidents: this dataset contains, for each streets of the city of Florence, the number of car accidents happens on these streets, and the comparison with previous years. The data shall include: car accidents on the observation date and total car accidents of the previous two years.
- Circulating vehicles: the dataset contains the number of vehicles registered in the municipal area, by ACI (Italian automobile club) per year. Data is related to years from 2001 to 2010.

For each dataset analyzed, the main fields and their meanings are listed below:

- Public door access and taxi:
   - Year: this field contains the year to which the statistics refer;
   - January to December: it contains an integer that indicates the number of accesses in the specified month;
- Resolutions:
   - Year: fields that contains the year of the decision;
   - Date: it contains the issue date of the resolution in year/month/day hour:minute:seconds.milliseconds format;
   - Note: this field contains an integer that uniquely identifies each acting within the file;
   - Subject: a text field to give a brief description of each resolution;

    - Link_Delibera: it contains an URL that allows to access to the resolution file online, in PDF format.
- Tourist Arrivals:
    - Year: it contains the reference year of the statistics;
    - Arrivi_italiani: this field contains an integer that indicates the total number of Italian tourists who visited the city in the specific year;
    - Arrivi_stranieri: an integer field that indicates the total number of foreign tourists who visited the city in the specific year.
- Afaf:
    - Year: fields that contains the year to which the statistics refer;
    - Numero_biglietti_venduti: an integer field containing the number of tickets sold in a specific year;
    - Numero_abbonamenti_venduti: an integer field containing the number of seasonal tickets sold in a specific year;
    - Numero_passeggeri: this field contains an integer indicating the total number of passengers in the specified year; L
    - Lunghezza_della_rete_in_km: it contains the length of the entire road network covered by all lines ATAF, in kilometers.
- Car accidents per road:
    - Address: it contains the street name to which the statistics refer;
    - Sinistri2009: this field contains the total number of car accidents incurred in 2009 on the specified road;
    - Sinistri2010: this field contains the total number of car accidents incurred in 2010 on the specified road;
    - Sinistri2011: this field contains the total number of car accidents incurred in 2011 on the specified road;
    - Deaths: it contains the total number of deaths that occurred in the last 3 years on the indicated street;
    - Injuries: the field contains the total number of injuries occurred in the last three years on the indicated street;
    - Bruised: a field that indicates the total number of bruised in the last 3 years on the indicated street;
    - Damage: this field contains the total number of accidents that have occurred in the last 3 years on the indicated street.
- Vehicles circulating:
    - Year: fields that contains the year to which the statistics refer;
    - Autovetture_registrate_circolanti: integer field that contains the number of cars recorded and circulating in the Florence area in the specific year;
    - Motocicli_e_ciclomotori_circolanti: integer field that indicates the number of motorcycles and mopeds registered and circulating in the Florence area, in the specific year;

- altre_tipologie_di_veicoli_circolanti: integer field that contains the number of vehicles that not belong to the previous categories, circulating Florentine area, in the specific year;
- totale_veicoli_circolanti: this field contains an integer indicating the total number of vehicles circulating in the Florence area in the specific year.

3.5 Real Time Data

In the initial phase of the project, thanks to the Tuscany Region, it was possible to access real-time data provided by the Mobility Information Integration Center (MIIC) that is defined in [MIIC-DateX] as a project of the Tuscany Region that deals with the real-time collection of information relating to public transport, traffic, emergency services and general services, provided by federated entities to MIIC, and their distribution by web service. In particular, the data produced by sensors on the traffic of Tuscany, by car park operators of the major cities of Tuscany and devices AVM (Automatic Vehicle Monitoring) installed on part of public transportation in the metropolitan area of Florence, were analyzed.

3.5.1 *MIIC Client Pull Service*

Measurements of traffic sensors and data relating to the state of the parking are exposed through the MIIC web service that provide the *clientPullService* that have only the *getDatex2Data* method. This method returns an object of D2LogicalModel type that contains structured data in XML format according to the standard DATEXII, that is a standard supported by the European Commission for the exchange of information between centers of road traffic management, traffic information centers and service providers and that has become the standard reference for all applications that access dynamic traffic information in Europe [DateX].
The web service is available in two different forms, the "open" one also called "Standard Catalog" that exposes a limited amount of data, and the "authenticated" one, also called "catalog in Publish/Subscribe", that performs an authentication checks. The authentication process consists of a credentials check provided by the MIIC during a registration phase and that has resulted in the creation of a new MIIC's Users, which can be identified by a Username and a Password. Users are then assigned to one or more entity catalog; the latter is a logical container that groups together one or more entities (i.e. D2LogicalModel) subjected to extraction of data according to a given criterion for classification [MIIC-DateX]. For example, in a catalog of the various Parking entities, each Parking can be grouped according to the criterion of geographic identity (the same City), proximity (the same geographical area), of importance (the main car parks, or the largest car parks, etc.).
According to what has been seen so far, for each type of data, two types of web service are made available; below the list of web service endPoint, and the corresponding link

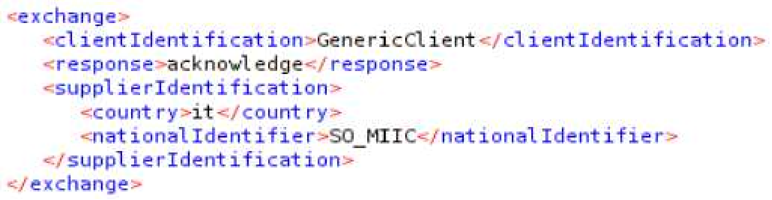
to get the WSDL files (Web Services Description Language), analyzed during the initial phase of the project:

- **Parking Standard Catalog**:
  http://www501.regione.toscana.it/MIICWebServices/parcheggiClientPullService
  http://www501.regione.toscana.it/MIICWebServices/parcheggiClientPullService?wsdl
- **Parking catalog in Publish/Subscribe**:
  http://www501.regione.toscana.it/MIICWebServices/parcheggiClientSecurePullService
  http://www501.regione.toscana.it/MIICWebServices/parcheggiClientSecurePullService?wsdl
- **Sensors Standard Catalog**:
  http://www501.regione.toscana.it/MIICWebServices/sensoriClientSecurePullService
  http://www501.regione.toscana.it/MIICWebServices/sensoriClientSecurePullService?wsdl
- **Sensors Catalog in Publish/Subscribe**:
  http://www501.regione.toscana.it/MIICWebServices/parcheggiClientSecurePullService
  http://www501.regione.toscana.it/MIICWebServices/parcheggiClientSecurePullService?wsdl

The information exchange with the web services is realized thanks to the SOAP (Simple Object Access Protocol) protocol. The invocation occurs through the request method HTTP Post which addressing the service endpoint and the following HTTP headers: Username and Password (i.e. the credentials provided by the MIIC), RequestCatalog (a number, also provided by the MIIC, that identifies the Catalog from which the data will be received) and SOAPAction. The last header is instead enhanced with the URI identifying the method to invoke, in this case it is as follows: *http://datex2.eu/wsdl/clientPull/1_0/getDatex2Data* where the reference to the getDatex2Data method is explicit. WSDL does not require the setting of parameters, in fact the body of the HTTP message consists of the opening and closing tag of a SOAP message, as shown in Figure 8.

The MIIC provides data also through the portal's graphical interface of the *Osservatorio Regionale per la Mobilità ed i Trasporti* [Osservatorio Trasporti] accessible by a certificate provided during a registration process. This portal has been used to integrate the data downloaded as previous description, with information that is not available in messages of D2LogicalModel type.

The result of a call to one of the web service previously indicated is formed by two elements: *Exchange* and *PayloadPublication*. The first element is the same for both variants, while the second contains the actual data of the measurements and so its internal structure varies according to the type of data, with exception for *publicationTime* and *publicationCreator* fields, that contain respectively the creation time of the given and the name of the institution that providing the data. In *Exchange* there are mainly generic information about the data exchange such as the data provider and the Client identity and the result of the transfer.



```
<exchange>
   <clientIdentification>GenericClient</clientIdentification>
   <response>acknowledge</response>
   <supplierIdentification>
      <country>it</country>
      <nationalIdentifier>SO_MIIC</nationalIdentifier>
   </supplierIdentification>
</exchange>
```

Figure 8 - Exchange tag content

In Figure 9 a map of Tuscany, generated by the *Osservatorio Regionale* is shown, where the position of the sensors, which produce the measurements transmitted from the web service, can be seen.

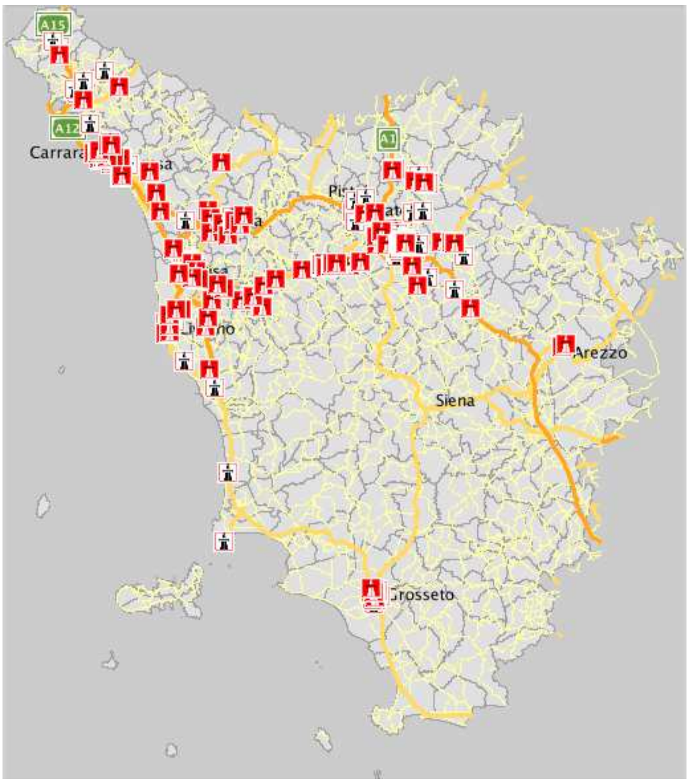


Figure 9 - Sensors position map

All sensors are located in the biggest city of the provinces of Florence, Grosseto, Livorno, Massa and Pisa; there are also active sensors in the municipality of Arezzo, not visible on the map. There are 158 catalogs, of which only 71 are currently active. A catalog or *siteTable* corresponds to a set of sensors usually located on the same street, and it is identified with a code such as *FI055ZTL001*. Each sensor of the group has a unique code which consists of the *siteTable* identifier with an extension of two-digit progressively numbers, for example FI055ZTL00101, FI055ZTL00102, etc.

The element *PayloadPublication* of each sensors contains the *measurementSiteTableReference* field that provides the siteTable identifier and for each sensor of the examined siteTable, there are one or more *siteMeasurements* elements. This tag contains the fields *measurementSiteReference* that identifies the sensor, *measurementTimeDefault* that contains date and time of the measurements according to the DateTime W3C format [DateTime] and a series of elements, which implement the *TrafficValue* interface. Below the field list of this interface, is provided, for a more in-depth description, see [MIIC-DateX]:

- AverageDistanceHeadway: this tag contains the average distance between vehicles transiting, expressed in meters;
- AverageTimeHeadway: a tag that contains the average time interval between two transits, in seconds;
- VehicleFlow: this tag indicates the number of transits per hour;
- AverageVehicleSpeed: it contains the average speed of transiting vehicles in km/h;
- Concentration: a tag that contains the number of vehicles per kilometer;
- Occupancy: in this tag, the percentage of occupation of the road is saved;
- Threshold: this tag contains the percentage of vehicles whose speed is less than the value defined in *value*;
- Value: tag that indicates the reference value for the previous tag *Threshold*;
- Period: a tag contains the value of the measurement intervals, in seconds;
- Time: it contains the measurement time, it is equivalent to *measurementTimeDefault*.

Each sensors, according to the catalog, making measurements every 5 or 10 minutes and a call to the web service returns the latest measurements of each sensor, up to a maximum of 6.

For example, sensors of catalog 15, measure the traffic every 10 minutes (period of 600 seconds) and each sensor contributes to payload with its last 2 *siteMeasurements*.

In Catalog 48, instead, sensors make a measurement every 5 minutes and the Payload are shown the last 6 *siteMeasurements* of each sensor.
The main problem of this data is the lack of information to geo-localize the sensors; in fact there is no information on their position, however, on the portal of *Osservatorio dei Trasporti*, the road name in which each sensor is installed, can be recovered to manually associate it to each sensor.
Among the 71 functioning catalogs, only 64 produce at least one of the data described above. Specifically, only 14 catalogs measure the *averageDistanceHeadway*, *averageTimeHeadway* is measured by all sensors (but half of them have the measurement fixed to "-1") as *vehicleFlow*, that appears in all the payload (but in 7 *sensorTable* the detected value is fixed to - 1) and *averageVehicleSpeed*; The value of *concentration* is measured only by 26 sensors groups; *occupancy* is present in the payload of only 15 catalogs; *threshold* and *value* appear in all measurements. In summary, only 15 catalogs correctly produce the complete set of measurements described above, and they are all located in Empoli and Grosseto areas.

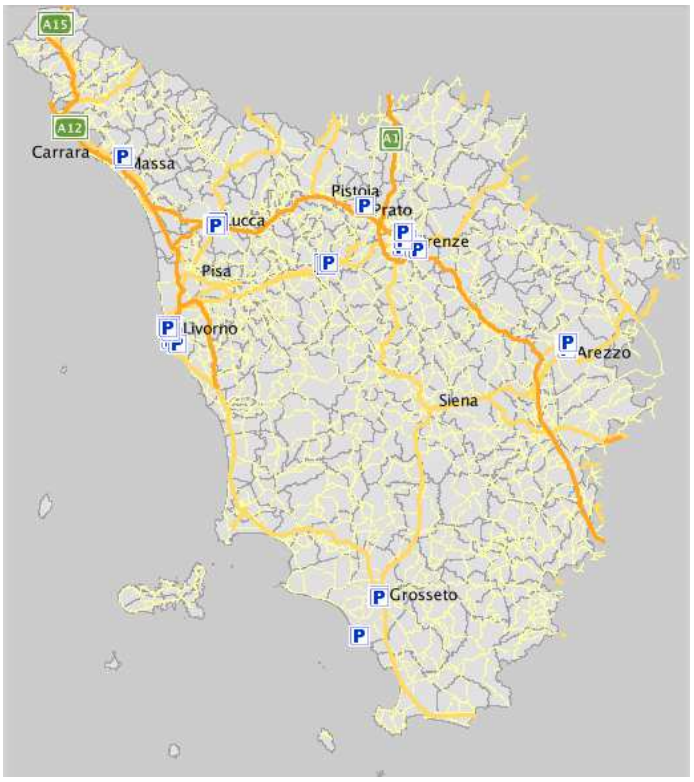


Figure 10 - Parking sensors map

Figure 10 is generated through the portal of *Osservatorio dei Trasporti*, and it shows the car parking for which the MIIC provides information on its status of employment, in the main centers city of Tuscany: Arezzo, Empoli, Livorno, Florence, Grosseto, Massa and Lucca.

Each city corresponds to a different catalog; the main element of the parking payload is *SituationRecord*, and there is one for each car parking belonging to the catalog. This interface can be implemented by different web services, and also it has a group of fields devoted to general information, common to all the possible specializations, and another group that changes depending on the data type.

The standard DATEX provides a more complex structure for the *SituationRecord* interface, compared to that actually realized by the MIIC. Below is a description of the individual fields, for which in [MIIC-DateX] are added more details:

- *situationRecordCreationTime*: The tag contains date and time of message creation, in DateTime format W3C. Equivalent to *publicationTime*;
- *situationRecordObservationTime*: it contains date and time of the survey, in DateTime format W3C;
- *situationRecordVersion*: tag set to '1' to parking payload;
- *situationRecordVersionTime*: tag equivalent to *situationRecordCreationTime* for the parking payload;

There is also a Validity class consisting of:

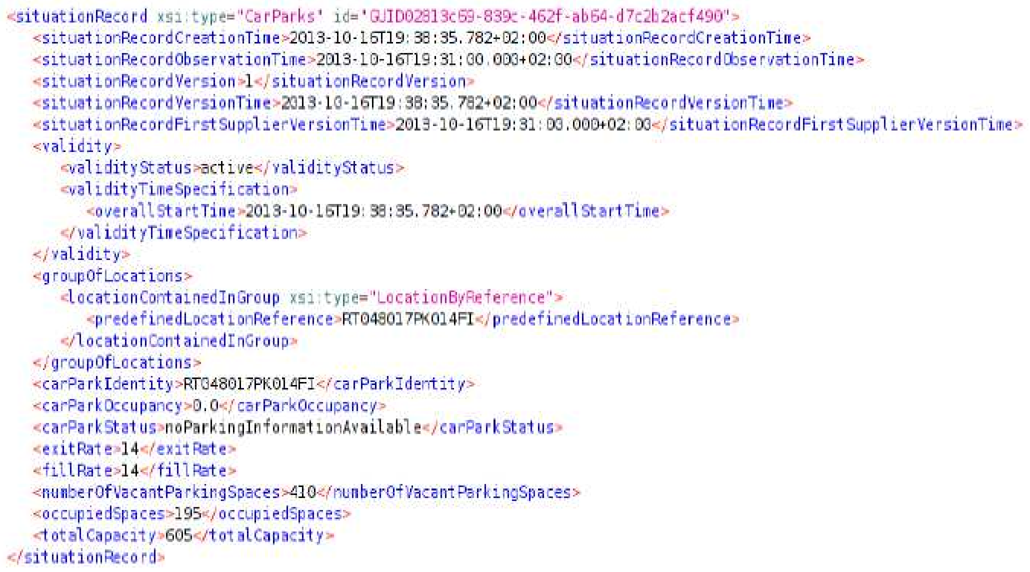
- *validityStatus*: tag sets to 'active' for the parking payload;
- *overallStartTime*: tag with content equivalent to *situationRecordCreationTime* to the parking payload.

Finally, the *groupOfLocations* class consists only by the *predefinedLocationReference* tag, enhanced with the identification code of the parking.

As regards the second part of *situationRecord*, the one specializes in description of the parking state, in Figure 11 is shown an example of *situationRecord* element that represents the fields returned by the web service during the testing phase. The following describes tags that make up this class:

- *carParkIdentity*: tag that identifies the car parking, equivalent to the content of *groupOfLocations*;
- *carParkOccupancy*: tag containing the percentage of occupied lots;
- *carParkStatus*: tag that contains a string that describes the parking state;
- *exitRate*: tag that indicates the number of output vehicles per hour;
- *fillrate*: it contains the number of inbound vehicles per hour;
- *numberOfVacantParkingSpaces*: tag containing the number of free lots;
- *occupiedSpaces*: tag indicating the number of occupied lots;

- *totalCapacity*: tag for the total number of lots.



```
<situationRecord xsi:type="CarParks' id='GUID02813c69-839c-462f-ab64-d7c2b2acf490">
   <situationRecordCreationTime>2013-10-16T19:38:35.782+02:00</situationRecordCreationTime>
   <situationRecordObservationTime>2013-10-16T19:31:00.000+02:00</situationRecordObservationTime>
   <situationRecordVersion>1</situationRecordVersion>
   <situationRecordVersionTime>2013-10-16T19:38:35.782+02:00</situationRecordVersionTime>
   <situationRecordFirstSupplierVersionTime>2013-10-16T19:31:00.000+02:00</situationRecordFirstSupplierVersionTime>
   <validity>
      <validityStatus>active</validityStatus>
      <validityTimeSpecification>
         <overallStartTime>2013-10-16T19:38:35.782+02:00</overallStartTime>
      </validityTimeSpecification>
   </validity>
   <groupOfLocations>
      <locationContainedInGroup xsi:type="LocationByReference">
         <predefinedLocationReference>RT048017PK014FI</predefinedLocationReference>
      </locationContainedInGroup>
   </groupOfLocations>
   <carParkIdentity>RT048017PK014FI</carParkIdentity>
   <carParkOccupancy>0.0</carParkOccupancy>
   <carParkStatus>noParkingInformationAvailable</carParkStatus>
   <exitRate>14</exitRate>
   <fillRate>14</fillRate>
   <numberOfVacantParkingSpaces>410</numberOfVacantParkingSpaces>
   <occupiedSpaces>195</occupiedSpaces>
   <totalCapacity>605</totalCapacity>
</situationRecord>
```

Figure 11 - SituationRecord example

The data described above were integrated with the name and code of the street where each parking is located; these information are in fact only available on the *Osservatorio dei Trasporti* portal, but they cannot be downloaded with a static link. So, a MySQL table is been created, which contains correspondence between *carParkIdentity* and additional data related to its position. In conclusion, data obtained from the web service *parcheggiClientSecurePullService* are in almost cases complete, in fact almost the surveys relating to parking census return valid data for the fields described above. At the moment only *exitRate* and *fillrate* are exceptions, that are measured by only a few parking in the Florence area.

3.5.2 *AVM client pull service*

The web service *avmClientPullService* differs from the web services previously described, mainly for the structure of data and for the methods to invoke it. The related RFC [MIIC-AVM] provided by MIIC, provides a complete description also for some features that have not yet been implemented. So, the work done, took advantage of a web service version that has limitations, which will be underlined when encountered. The endpoint of the web service is:
http://www501.regione.toscana.it:80/MIICWebServices/avmClientPullService
the communication protocol is SOAP and, also in this case, there is only one defined method: *getAvmMessage*. In contrast to what was seen previously not access credentials are required, while the body of the SOAP request requires parameters setting.

The structure of information exchanged during the communication can be ideally divided into two parts, *exchange* and *listaCorse*; this latter contains the AVM detections provided by MIIC following a well-formed request. The *exchange* item is used by the client to fill out the request and, during the response of the server, to provide information on the outcome of the communication. In detail, fields involved are the follows:

- *idConsumer*: tag that contains the client identifier, i.e. the user of data;
- *serverID*: this tag instead contains the server identifier, that corresponds to the string "SO_MIIC";
- *oraInvioMessaggio*: it contains date and time of the request, in format W3C dateTime;
- *Keepalive*: tag to be set to "True" if you just want to check the status of the web service, without having to set other request parameters;
- *pullType*: tag that specifies the manner in which data are required to the server. It can be "*catalog*" or "*filter*"; differences between the two values will be specified below;
- *pushType*: tag not significant for the service *avmClientPullService*;
- *catalogueReference*: tag that takes value in case *pullType* = "*catalog*";
- *filterReference*: tag significant in case pullType = "*filter*"; it contains a series of sub-elements, like *codiceLinea*, *codiceCorsa* and *codiceCorsaLogico*, with decreasing priority;
- *response*: tag where the server indicates the outcome of the communication; it may be "*ok*" if successful, "*notOk*" in case of error or "*Stillalive*" which indicates that the service is active, in case of a "*keepalive*" request;
- *notOkInfo*: tag that provides the details of any error, if *response* = *"notOk"*.

As anticipated, the MIIC has not completed the development of the web service, and in particular is not currently active the *pullType* = "*catalog*" mode, and *filterReference* refers only to codiceCorsaLogico, valued as *codiceCorsa*, so for simplicity will be referred to the latter value.
So in summary, to obtain data via the *getAvmMessage* method, it is necessary to set *pullType* = "*filter*" and to assign a value to the field *codiceCorsaLogico*.
In order to assign a valid value to *codiceCorsa*, so it is recognized by the server, the RFC in [MIIC-AVM] refers to a "Ride DataBase", which should be created by the MIIC in collaboration with local data providers and subsequently made available to web service users. At the moment, however, this database has not been created and, on the website of *Osservatorio dei Trasporti*, there are no links that allow to download a list of asset rides and related codes. Looking forward official data, a temporary solution has been developed based on data coming from the *avmClientPullService* web service, through the web interface mentioned above.
We clarify the concept of race: it is the path that a public transport perform from a start terminus to an end terminus, that is each time a vehicle leaves a terminus, a new ride is

enabled, associated with a specific code, which ends on arrival to the end terminus. This ride is usually repeated the next day with the same code and the same departure time, arrival time, line, stops, etc. These ride codes consist of 7 digits, they range from 4500000 to 4800000 and then, based on this information, a temporary database has been created, populated thanks to calls to the web service, which prove one by one each integer in that range and store into the database, only the codes correctly interpreted by the server.
The tags contained into each ride element are:

- *codiceCorsa*: tag valued at inquiry phase;
- *codiceCorsaLogico*: this tag contains the same value of *codiceCorsa*;
- *codiceLinea*: it contains the identification code of the line related to *codiceCorsa*;
- *codicePercorso*: the tag contains the identification code of the path;
- *enteErogante*: tag that identifies the company that provides the Local Public Transport (LPT) service;
- *enteGestore*: tag that identifies the LPT service manager;
- *Message type*: tag that can assumes the following values: "*synchronous*" to indicate a programmed transmission of data at regular intervals, "*event*" for sending important data or unplanned data, "*modification*" to report a change in the service;
- *variation*: tag present in case of *Message type* = *"modification*", which contains the description of the change;
- *info*: tag present in case of "*synchronous*" or "*Event*" Message type.

The info element contains key information about the state of race in detail:

- *statoCorsa*: this tag contains an integer set to 0 if the ride is in advance, 1 if it is late and 2 if it is in time;
- *minutes*: tag that indicates the minutes of delay or advance;
- *oraInvio*: tag containing the date and time when the message was sent, in format W3C DateTime;
- *latMezzo*: tag indicating the latitude of the vehicle in format WGS84 Datum;
- *longMezzo*: tag indicating the longitude of the vehicle in format WGS84 Datum;
- *idMezzo*: it contains the identification number of the vehicle;
- *tipoEvento*: in case of *Message type* = "*evento*", this tag can be "*partito*" means the vehicle is departed from terminal, "*intermedio*" that is, the vehicle is at an intermediate stop, "*arrivato*" i.e. it is arrived to the terminus, "*interrotto*" i.e. the service has been interrupted, "*riabilitato*" means the service is restarted after an interruption;
- *ultimaFermata*: tag forming by the couple *idFermata* and *orarioFermata*, indicating the identifier and the time of the last stop made by the vehicle;

- *listaProssimeFermate*: this tag contains an ordered list of pairs *idFermata* and *orarioFermata*, indicating the next stop forecast for the ride;
- *warning*: this tag provide a description of the interruption reason when *tipoEvento* = "*interruption*";
- *carico*: tag indicating the maximum number of passengers on the vehicle;

Thanks to the tests carried out on this web service, it was found that *listaProssimeFermate* is associated with the vehicle rather than to the race, in fact initially this tag contains some of the planned stops of the route, listed in an orderly way, but when the vehicle approaches the terminus of the ride in progress, in the tag are also added the scheduled stops for the next ride that correspond to the return path, which will start after the same vehicle reaches the terminus, which marks the end of the race.
So the message of synchronous type sending before a message of type "*arrivato*" contains the list of the upcoming stops that will do the same vehicle, under a new ride code.
A limitation of this web service is that currently monitors a small number of lines (lines 2, 4, 6, 13, 17, 23, 31, 36) compared to the total amount of those operating on the Florence area, but the procedure for access to all the lines is already in progress.

Chapter 4

4. Km4city Ontology

With the availability of the data presented in the previous chapter, the development of a unified and integrated ontology for smart city including transport, info-mobility and large set of other open data, has become necessary.
In fact, as seen in the introduction, the presence of an ontology allows guiding and automate the ingestion process, exploiting a range of algorithms for: data cleaning, verification and validation, reconciliation, enrichment with VIP names on dbPedia, enrichment with geonames, enrichment of digital location and point of interests descriptions, etc. see [DICCOF].
For these reasons, an ontology capable of mapping the different available data, has been created, trying to give it a certain abstraction degree, that allows to use it even in different scenarios compared to that in which it was developed.
In this chapter the concept of ontology will be initially clarified and then, the state of the art of ontologies in the field of Smart City, will be analyzed. Finally, the *Km4city* ontology, that is the ontology developed during the research carried out in the PhD, will be presented.

4.1 What is an ontology?

An ontology is actually one of the most efficient methods to formally represent a set of concepts.
Ontology is a term that derives from philosophy: it appears for the first time in the writings of Parmenides (about 504 BC) and then derives from the greek "eon logos", meaning "speech ENTITY": ontology is concerned, in fact, the study of being, or of what it is and its basic categories.[Gruber, 1993]
In computer science, ontologies are most used in the field of Artificial Intelligence, for the classification of the data; T.R. Gruber in fact, defines an ontology as a “specification of a conceptualization” [Gruber, 1993]. He states that a formal representation of a set of

knowledge is a conceptualization, i.e. a set of objects, concepts and relationships, between them that exist in a particular area of interest. A conceptualization is, therefore, a simplified and abstract representation of the particular field of knowledge to be represented, for any purpose.
An ontology is thus an attempt to formulate a exhaustive and rigorous conceptual scheme of a given domain, and this pattern can take several forms, in fact, there are different types of ontologies, and below, a classification based on the language expressivity and formality, is explained [Roussey et al., 2011]:

- **Information Ontology**: it is composed of diagrams and sketches used to clarify and organize the idea of collaborators during the development of a project. This type of ontologies are only used by humans and among their characteristics, there are the ease of modification, scalability, conciseness, schematization. Information ontologies focus on concepts, instances and their relationship, and they are main described by means of visual languages.
- **Linguistic Ontology**: dictionaries, vocabularies, dictionaries, taxonomies, folksonomies, thesaurii and lexical database, are some examples of this type of ontologies, that mainly focus on terms and their relationships. Their aim is to present and define the used vocabulary, which is the results of a terminology agreement between the interested users' community. SKOS (Simple Knowledge Organization System) [SKOS] is the preferred language to describe Linguistic ontology.
- **Software Ontology**: this type of ontologies provide conceptual schemata whose main focus is normally on data storage and data manipulation, and they are used during software development activities, with the aim to guaranteeing data consistency. Software ontologies are normally defined with conceptual modeling language used in software and database engineering like Entity-Relationship Model Language or Unified Model Language (UML) [Cranefild, 2001].
- **Formal Ontology**: they require a clear semantics for the used language in defining concept, and strict rules about how to define concepts and relationships. All this can be obtained by using first order logic [Smullyan, 1968] or Description Logic [descLogic] where the meaning of each concept is guaranteed by formal semantics [Borgo, 2004]. An example of this type of ontologies, is a Knowledge Base (KB), that is formal system to capture the meaning of an adopted vocabulary via logical definitions. The main purpose of formal ontology is reasoning. OWL [OWL] is the standard recommended by W3C to define this type of ontologies.

The interaction between people and software systems leads to the search of a common and shared system to communicate and understand information, so the idea is to use a shared vocabulary to describe the content of the resources, the semantics of which is described in a reasonably unambiguous format and processable by a machine.

Any information will be mapped by its own ontology and inserted in a context that creates relationships with other ontologies, in order to create logical relationships that allow, for example, to distinguish the meaning of the word "root" in the context of a "natural environment" than to "root" in a context of "arithmetic", that is understandably be different for any program semantics.
Thanks to this type of structure, all information will have a complete meaning in a certain context, according to the mechanism of information association of the human brain. So an ontology allows a conceptualization of explicit semantics of the data, with a more syntactically and semantically richer language, and with an agreed terminology so that the ontology can be reused.

4.2 State of the Art of Smart City Ontology

Before starting to develop the *Km4city Ontology* [Bellini, Nesi, Rauch, 2014], a study on the state of art of ontologies related to smart cities, was completed. From this preliminary analysis the following ontologies has resulting as the most interesting:

- The *SCRIBE* Ontology, realized by *IBM* in 2011;
- The Ontology of Transportation Network (*OTN*), realized by the Department of Computer Science, University of Munich, in 2005;
- The Semantic Sensor Network Ontology (*SSN*), realized by the W3C Semantic Sensor Network Incubator Group, in 2009;
- The *StarCity* Ontology, realized by the IBM Research (Dublin, Ireland) in 2013;
- The *SEMANCO* Ontology, realized in the homonymous European project, co-funded by the European Commission within the 7th Framework Programme;
- The *BOnSAI* Ontology, realized as part of the Smart IHU project of the International Hellenic University.

In the following paragraphs these ontologies will be briefly described.

4.2.1 *SCRIBE*

The only ontology that can be found on the web, created especially for smart city and presented as a "*help to transform cities into Smart Cities*" is *SCRIBE* [SCRIBE], made by IBM, on which not much information is available online free of charge.
SCRIBE is a semantic model and tools for "smarter" cities, based on data gathered from cities around the world, which try to solve some challenges like integration of multiple tools and modeling technologies, such as RDFS, OWL and UML. The aim of the *SCRIBE* team is to realize an authoritative information model that captures dynamic aspects of as much as possible service in the city.
The ontology is realized keeping in mind that the organizations of each city is very specific; furthermore, to model a Smarter City operation, corresponds to model the flow

of events and messages through the system, so the ontology is compatible with both standards *CAP* (Common Alerting Protocol) [CAP] e NIEM (National Information Exchange Model) [NIEM].
The first version of *SCRIBE* includes the following main sub-taxonomies:

- *CityPhysicalBase*, which includes the physical objects (geospatially and temporally based) in the city landmarks, roads, etc.;
- *EntityRoleBase*, to describe organizations, people, items (i.e. entities as in NIEM) and their roles;
- *EventAndMessageBase*, is based on CAP, and include event organized temporally and causally, such as *ExternalEvents* (like storms, road work), *Message* (a Road Work Advisory), *WorkItem* (Road Work Work Item) and a *SystemEvent* (application alert).;
- *MeasurementBase*, for measurements (height, length) and measurement units (inches, cm);
- *OrganizationBase*, which captures the city' organization and the set of service areas;
- *ProtocolBase*, to describe the city protocols as a set of protocol steps;
- *SCGeo*, that is the geospatial core sub-ontology;
- *TimeBase*, an extension of Time ontology [TimeBase].

In essence, the *SCRIBE* is not closed, in fact it consists of a Core Model that includes all common classes views above, which can be expanded with the extensions of the domain and the customization of individual cities: all building blocks (service types, city departments, KPI taxonomies, CAP messages) can be customized to define the overall operations of a city.

4.2.2 *OTN*

Among other ontologies that may be related to Smart Cities, we mention the *OTN*, that is the *Ontology of Transportation Networks*, which corresponds to a direct encoding of the *Graphic Data Files* [GDF] in OWL.
This ontology defines the entire transport system, from the single road/rail, to the type of maneuvering, that can be performed on a segment of road, or public transport routes.

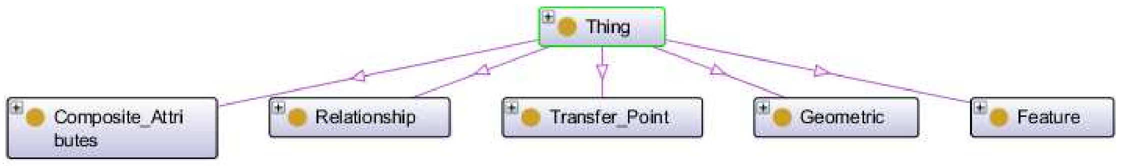


Figure 12 - OTN Ontology portion

As shown in Figure 12, the *OTN* includes, as in GDF standards, five main concepts expressed in the following five main macro classes:

- *Composite_attributes*: it represents classes consisting of composed attributes, and includes classes like *TimeTable*, *Accident*, *House_Number_Range*, *Validity_Period*, *Maximum_Height_Allowed*;
- *Features*: it contains all GDF features as OTN classes, such as *Railways*, *Service*, *Road_and_Ferry_Feature*, *Public_Transport* ;
- *Geometry*: this macroclass defines the geometric forms of features, i.e. *Edge*, *Node* and *Face* classes;
- *Transfer points*: it is a class which describes how to get from one object to another (e.g. train stations), which includes classes such as *Road*, *Road_Element*, *Building*, and others;
- *Relationships*: this macroclass describes the non-geometric relationships between features, such as the *Maneuvre*;

This ontology defines many concepts that can be easily identified in the data set used in this research. Therefore this ontology was also used as a reference vocabulary for the definition of the Knowledge Model created.

4.2.3 *SSN ontology*

On the Web there are also many ontologies related to sensor networks, such as the *SemanticSensorNetwork* Ontology, which provides elements for the description of sensors and their observations [SSN].
This ontology has been defined using *OWL 2* ontology language and it has been created starting from the review of standard and existing ontology.
The *SSN* ontology is organized, conceptually but not physically, into ten modules; the full ontology consists of 41 concepts and 39 object properties, directly inheriting from 11 *DUL* (DOLCE Ultra Light) concepts and 14 *DUL* object properties. The ontology is able to describe sensors, the accuracy and capabilities of such sensors, observations and methods used for sensing. Also concepts for operating and survival ranges can be easily represented, as these are often part of a given specification for a sensor, along with its performance within those ranges.
The SSN ontology is built around a central Ontology Design Pattern (ODP), shown in Figure 13 [Gangemi, 2005] which describe the relationships between sensors, stimulus, and observations, the Stimulus–Sensor–Observation (SSO) pattern. The ontology allows to observe the represented domain from four different perspectives:

- A sensor perspective, focused on what senses, how it senses, and what is sensed;
- An observation perspective, focused instead on observation data and related metadata;
- A system perspective, focusing on systems of sensors and deployments;
- A feature and property perspective, focusing on what senses a particular property.

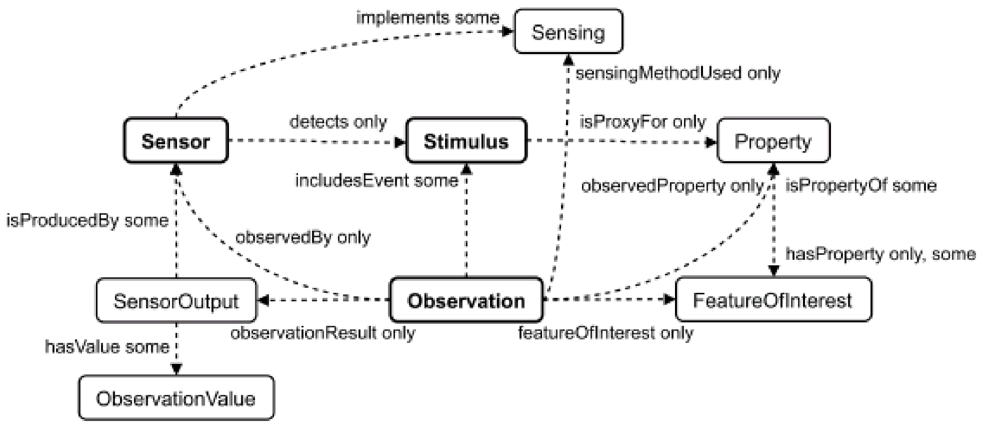


Figure 13 - SSN Ontology concept

However, there is a shortage in the ontology presented: locations of platforms, systems or sensors and temporal properties of deployments are areas where other ontologies are required to fill in the details.

Currently, inside the ontology *Km4City*, sensors are not represented, because detailed data on such devices are not available; in case there is the need to represent the sensors with greater detail, the ontology SSN will definitely be taken into account.

4.2.4 *STAR CITY ontology*

STAR-CITY is a system supporting semantic traffic analytics and reasoning for city, which integrates human and machine-based sensor data. Normally data are provided with a variety of formats, velocities and volumes, in fact *STAR-CITY* can handle structured and unstructured data, static data, stream data and large amount of historical data.
This ontology has been mainly designed to provide insight on real time traffic conditions; in fact *STAR-CITY* relies the semantic interpretation of the contextual information to derive insights like as analysis and diagnosis, contextual explorations and predictions of traffic conditions thanks to high accurate research in semantic predictive reasoning.
The novelty of *STAR-CITY* lies in the system ability to ingest highly heterogeneous real-time data and perform various types of inference, that is, analysis, diagnosis and prediction. These inferences are processed through a combination of various types of reasoning as Description Logic, machine learning based, rules-based, stream based [Starcity].
Currently this ontology has been applied to the context of the city of Bologna and Dublin but can be applied in other cities that expose all types of sensors data.

4.2.5 *SEMANCO*

SEMANCO [Semanco] is the reference ontology developed within the context of *SEMANCO* project (http://www.semanco-project.eu), which seeks to develop tools to provide data integration, required by Smart City cluster.
The *SEMANCO* ontology born as a mediator for the integration of technical and statistical data relating to buildings, which are normally measured and distributed through a set of heterogeneous data structures. Similar ideas of ontology-driven data integration can be found in [Calvanese et al., 1998] and [Wang et al.,2009].
The ontology is one of the semantic tools that the project provide to stakeholders involved in urban planning, to help them to make informed decisions about how to reduce carbon emission in cities; precisely the ontology is used as a tool for semantic data analysis and interconnection of various data sources (cadaster, census, building types, GIS). This ontology should help to interconnect all these data, according to their semantics, facilitating federated query for the entire data set and enabling semantic interoperability of tools operating on data.
The ontology building process includes six basic steps: capture of basic terminology taking into account the user perspective; construction of the initial vocabulary, based also on official classifications and standards; data source mapping on vocabulary; encoding the specific ontology using the appropriate tools developed in the project; data integration based on a model R2RML; ontology evaluation mainly based on its computation efficiency.
The most recently published *SEMANCO* ontology consists of 1042 classes, 849 properties and 7192 axioms.

4.2.6 *BONSAI*

BOnSAI is an ontology for enabling Ambient Intelligence (AmI) in a Smart Building, which, thanks to its domain-dependent characteristic, specializes the already existing domain-independent ontologies in order to model a domain-specific concepts of an AmI application. However, *BOnSAI* sets off to model many more concepts required in a Smart environment.
The *BOnSAI* ontology is designed to enable automation and energy savings at the International Hellenic University (IHU) where a Smart ambient has been setting, i.e. an environment equipped with smart devices (actuators and sensors) in large scale. The interaction with the rest of the system is provided through a web service interface. In this ontology is possible to find context-related, service-related, hardware-related and functionality-related classes.
Hardware-related concepts support both energy-awareness, capabilities and services in the system. All hardware is mainly divided into appliances and devices, which differ in their ability to provide services; the only thing in common is that they both have a location so they can be placed according to where they work.

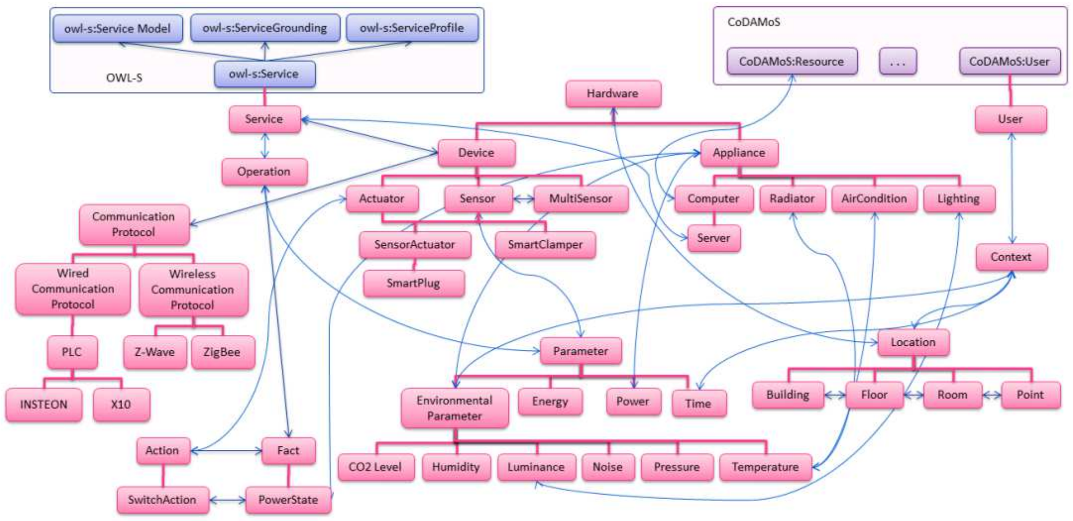


Figure 14 - BOnSAI Ontology

Appliances correspond to non-service-enabled appliances in the building such as radiator, printer, lighting, air conditioning, etc. The devices, instead correspond to enabled services smart devices, i.e. sensors and actuators. BOnSAI also includes the classes of MultiSensor and SensorActuator, which is destined for the devices of dual purposes.

4.3 The ISO/IEC JTC 1/SG 1 Standard

The definition of a standard relating to Smart Cities is currently underway, by the ISO/IEC. The analysis of this topic was requested by China, which has also submitted its contribution for the Smart City Reference Model. The Smart City Reference Model is the abstract, comprehensive, macroscopical description of Smart City. It provides guidance for Smart City construction, which will bring the city to a higher, more mature level, in other words, to provide the city with the ability to put together all its resources effectively taking advantage of opportunity, but without address challenges. Meanwhile, Smart City Reference Model also contributes to the analysis of the standardization need of Smart Cities and the formation of the Smart City standardization roadmap.

The second Plenary Meeting of ISO/IEC JTC1/SG 1 on Smart Cities, was held in London, in September 2014.
The key message of the report drawn up during the Plenary Meeting, is that the intelligence of a city refers to its ability to function as a single organism, in which all its resources are managed and distributed in an efficient way, to help the city and its citizens to "flourish".

So, for this purpose, a set of intelligent features of the city have been identified. these characteristics include smart city standard needed, but also enabling technologies that make smart city applications possible.
In the document four concepts are clarified: the *need*, described as the expression of something desired or considered necessary, which is often expressed as a general concern; a *requirement* instead is a formal statement of some functionality whose output contributes to a desired *outcome* that will satisfy a need. The *output* is unambiguous, but not necessarily produce the *outcome* that leads to the desired result.

The aim of the ISO/IEC JTC 1 is to identify the specific requirements of ITC standardization, on the basis of an understanding of the special needs of smart cities.

Within the standard there is already a chapter, in which the Enabling Technologies for Smart Cities are listed, because the future Internet domain landscape, comprises a great diversity of research streams and related topics for designing alternatives for Smart Cities. However, the following technology streams are the most connected to Smart Cities:

- **Ubiquitous Computing**: a concept in software engineering and computer science where computing is made to appear everywhere and anywhere. In contrast to desktop computing, ubiquitous computing can occur using any device, in any location, and in any format.
- **Networking**: it increases broadband capacity with 4G LTE and IP Multimedia Systems (IMS) as well as future networking technologies. Networking technologies provide the infrastructure of the smart cities to make all the devices, computers and people can have convenient, reliable and secretive communication with each other.
- **Open Data**: Open data, especially open government data, is a incredible resource that is as yet largely untapped. Open data plays an important role in the construction and operation of the smart cities. When the smart city is constructed, open data can provide large amount of data to assist the city planners and constructors. The citizens and city managers can make right decisions in city lives and managements.
- **GIS** (Geographic Information System): it is used to provide location based services. GIS and location intelligence applications can be the foundation for many location-enabled services based on analysis, visualization and dissemination of results for collaborative decision making.
- **Big Data**: a smart city can be seen as a "system of systems", and several systems offer large amounts of information. Using model smart city technologies, the amount of data increases rapidly. This allows to do many things that previously could not be done: spot business trends, prevent diseases, combat crime and so on.

- **Cloud Computing**: it is still helping private sector to reduce costs, increase efficiency, and work better. From a commercial point of view, cloud computing is a key concept to enable a global ecosystem, where organizations are able to be more competitive.
- **IOT**: Internet of Things (IoT) refers to the interconnection of uniquely identifiable embedded devices, within the existing Internet infrastructure. The interconnection of these embedded devices (including smart objects), is expected to usher in automation in many fields, while also allowing advanced applications such as Smart Grid.
- **E-Government:** the development of effective and efficient e-government is a key prerequisite for the development of Smart Cities. The lack of integration among various e-government and urban initiatives, and the relatively low level interest shown by many national authorities, limit the efforts for the development of local e-government.
- **Service Oriented Architecture** (SOA): Service-Oriented Architecture (SOA) is a software architecture design pattern based on different pieces of software that provide application functionality as services to other applications; it is also independent of any vendor, product or technology. Adopting a SOA approach for local government organizations requires a new way of thinking about IT infrastructure, not only technically, but also organizationally.
- **Embedded Network**: Embedded networks of sensors and devices into physical space of the city are expected advancing the ability create by Web 2.0 applications, social media and crowdsourcing. A real time spatial intelligence is emerging to have a direct impact on city services. Smart Cities with instrumentation and interconnection of mobile devices and sensors can collect and analyze data and improve the ability to manage and predict urban streams.

Within the document, there is a section dedicated to the knowledge model: the heterogeneous data from different sources must be aggregated, so a unified set of concepts and terminologies became essential. Furthermore, the development of new applications requires a common knowledge base on smart city. In order to support a cross-domain and cross-city interoperable knowledge, a core model that collects concepts from different stakeholders, is necessary, to support a more standardized knowledge expression. This model should include a taxonomy of different types of smart devices (mobile device, sensors, etc), the main Smart City areas (mobility, health, etc.), and the most important components of each sector (for example buses and trams for mobility).

As shown in the next chapter, the knowledge model made during these three years of research, can be considered in large part a specialization of the core model proposed within the standard. In addition, in this thesis is possible to identify references to most of the enabling technologies, identified as fundamental to a Smart City, in the document issued by ISO/IEC. The evidence found in official documents, even if not yet in an official version, incite to continue the search in the direction taken in order to improve the results obtained to date.

4.4 Km4city, the Knowledge Model for the City

In order to create a knowledge model for Smart City services, following the study carried out on existing ontologies in the field of smart city and the large number of data sets that have been analyzed, a new knowledge model has been developed, with the aim of modeling and establishing the needed relationships among element, thus making a general data set semantically interoperable (e.g., associating the street names with toponimous coding, resolving ambiguities), maximizing the compatibility with existing solutions. The results of this deep analysis phase is the *km4City Ontology*, a knowledge model for the city and its service, an ontology formed by 78 classes and 93 ObjectProperties.

Km4City is in fact designed to be the knowledge model for a generic Smart City and its services, and it allows to interconnect all data sets provided from several PA and mobile operators, with the aim of creating a single data store useful, for the same PA or third parties, to develop new innovative applications that improve and simplify the citizens life.

The work performed started from the data sets available in the Florence and Tuscany area, presented in previous chapter.

To avoid ambiguity and to increase the understanding of the concepts defined in the knowledge model and to make the ontology made more adaptable to other urban realities, a number of reference vocabularies were chosen and used during the development phase.

Following the analysis of available data relating to the Street Guide provided by the *Osservatorio dei Trasporti*, many similarities between the data and part of *OTN* Ontology, presented in Section 4.2.2, have been identified.

In order to associate a more precise semantics to the various entities of *Km4City* Ontology, an explicit reference to the corresponding entity into the *OTN* ontology, has been defined inside the created knowledge model, with the aim to facilitate connection with other datasets, also to follow the guidelines for the for the implementation of *Linked Open Data* (http://www.w3.org/DesignIssues/LinkedData.html) (see Chapter 2.3).

So, the *OTN* ontology has become one of the reference vocabulary used during the development of the Km4city model [Bellini et al., 2014B]. Among reference vocabularies used in the definition of the Knowledge Model is possible to find the *dcterms*, i.e. a set of properties and classes maintained by the Dublin Core Metadata Initiative, the *foaf* vocabulary, an ontology dedicated to the description of the relations between people or groups, *schema.org* a reference model used to provide description of people and organizations, and the *wgs84_pos* vocabulary, representing latitude and longitude, with the WGS84 Datum, of geo-objects.

Another interesting knowledge model is the *GoodRelation* Ontology [GoodRelation] 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; moreover, products and services offered can increase their visibility into latest generation search engines, or recommendations systems and similar applications.

GoodRelation is clearly the reference model for the description of e-commerce and their products; with a view to future expansion, an interconnection between *Km4city* ontology and *GoodRelation* has been created.

The *Km4city* knowledge model allow to interconnect, store and then query, a collection of data coming from many different sources, specifically various portals of the Tuscany Region (*MIIC*, *Muoversi in Toscana*, *Osservatorio dei Trasporti*) and Open Data provided by individual Tuscany municipalities (mainly Florence), as shown in Section 3.4. Taking into account the different types of data that he Knowledge Model has to map, it is clear that its size will be relevant, and therefore, to facilitate its construction and its understanding, it has been divided into the following macroclasses:

- **Administration Macroclass**: it is structured in order to represent the Italian public administration hierarchy: each region is divided into several provinces, within which the territory is divided into municipalities. Moreover each PA, during its mandate, can produce resolutions and publish statistics.
- **Street Guide and Rail Network Macroclass**: it represents the entire roads system and railway system in Tuscany; the first one, from an administrative point of view, is seen as a set of administrative extensions or administrative roads, while from the citizen' point of view, it is composed by a set of roads. The second one, from an administrative point of view is seen as a set of railway directions, railway lines, railway sections, while from the citizen' point of view, it is composed by a set of train stations interconnected together.
- **Points of Interest Macroclass**: this macroclass allows to represent services to the citizens, points of interest, businesses activities, tourist attractions, and anything else can be located thanks to a pair of coordinates on a map.
- **Local Public Transport Macroclass**: it includes information relating to public transport by road and rail; 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.

- **Sensors Macroclass**: is the macroclass created to host data collected by various sensors installed along some roads, in major car parks of Florence and in that neighborhood. In addition in this macroclass data related to Lamma's weather forecast were included.
- **Temporal Macroclass**: it 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.
- **Metadata Macroclass**: it is used to keep track of the status and descriptors associated with the various ingested dataset.

4.4.1 Administration Macroclass

The first macroclass, i.e. Administration, is composed as shown in the following Figure 15.

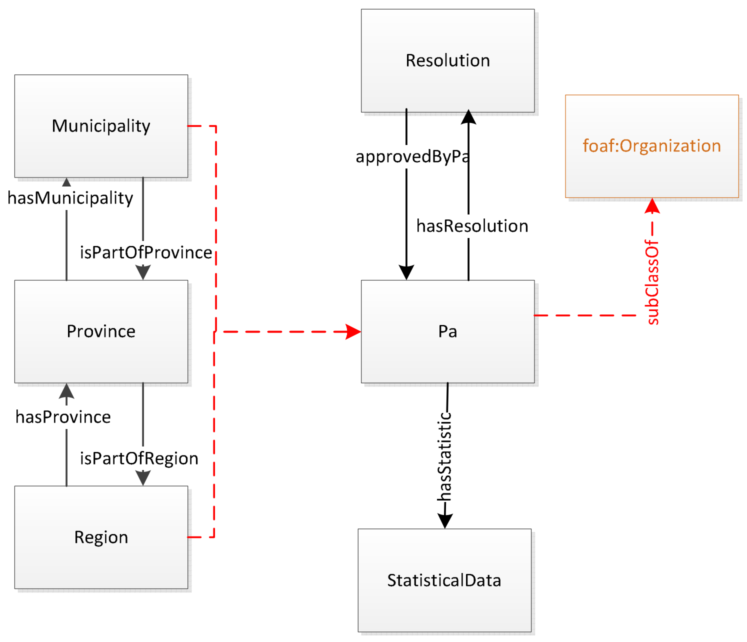


Figure 15 - Administration Macroclass of Km4City Ontology

The main class of Administration Macroclass is *PA*, which has been defined as a subclass of *foaf:Organization*, link that helps to assign a more clear meaning to this class. The three subclasses of *PA* are automatically defined according to restriction defined on ObjectProperties (represented in the figure by solid lines). For example, the *Region* class is defined as a restriction of the class *PA* on ObjectProperty "*hasProvince*", so that only the *PA* that possess provinces, can be classified as a region. Another example: to define the PA elements that make up the *Municipality* class has been instead used a restriction on ObjectProperty "*isPartOfProvince*," so if a *PA* is not assigned to a province, it cannot be considered a municipality.

To establish the hierarchy within the class *PA*, a pairs of inverse ObjectProperties were defined, for each step: "*hasProvince*" and "*isPartOfRegion*", "*hasMunicipality*" and "*isPartOfProvince*."

Connected to the class *PA*, through the ObjectProperty "*hasResolution*", is possible to find the *Resolution* class, whose instances are represented by the resolutions passed by the various PA note. The "*hasResolution*" ObjectProperty has an inverse property, that is, "*approvedByPa*".

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

4.4.2 Street Guide and Rail Network Macroclass

The *Street Guide and Rail Network* macroclass consist of two separated parts, that is the *Street Guide* and the *Railway Graph*, which will be analyzed below, respectively.

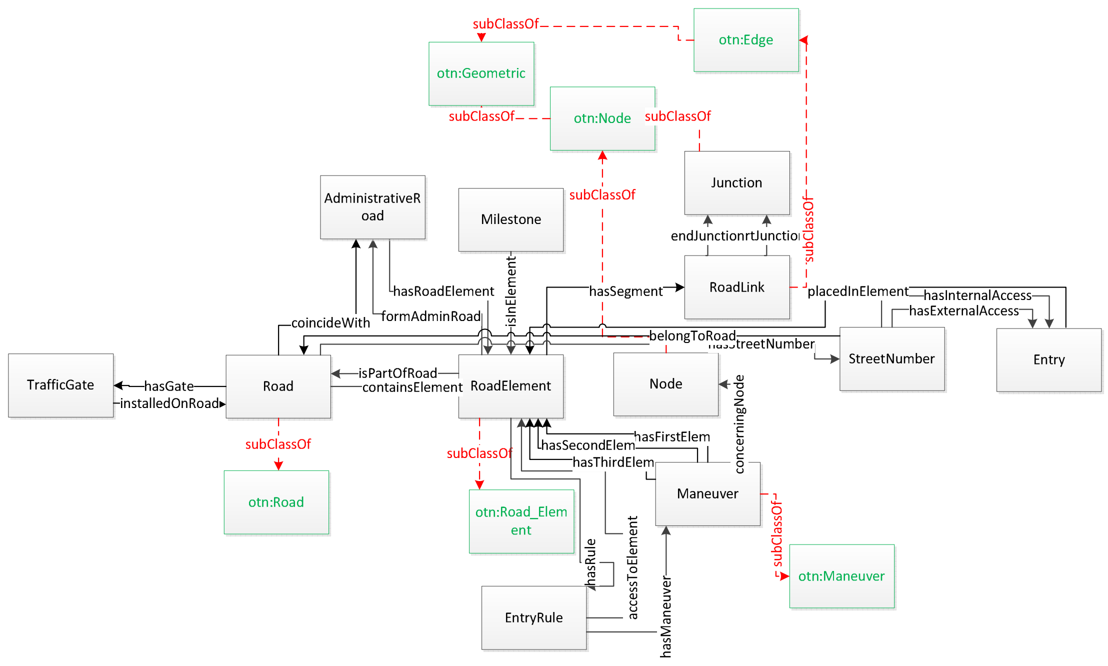


Figure 16 - Street Guide Macroclass of Km4City Ontology

In Figure 16 the Street Guide part is shown; its main class, in the middle of the figure, is *RoadElement*, which is defined as a subclass of the corresponding element in the ontology *OTN*, i.e. *Road_Element*. Each *RoadElement* is delimited by a start node and an end node, detectable by the ObjectProperties "*startsAtNode*" and "*endsAtNode*", which connect the *RoadElement* class to the class *Node*. Some restrictions have been specified in the *RoadElement* class definition, related to the *Node* class: a road element must have both "*startsAtNode*" and "*endsAtNode*" ObjectProperty, both with a cardinality exactly equal to 1. One or more road elements forming a road: the class *Road* is in fact defined as a subclass of the corresponding class in the *OTN*, i.e. the homonymous class *Road*, with a cardinality restriction on the "*containsElement*" ObjectProperty, which must be minimum equal to 1, in other words, a road that does not contain at least one road element, cannot exist. Also the *AdministrativeRoad* class, which represents the administrative division of the roads, is connected to the class *RoadElement* through two inverse ObjectProperty "*hasRoadElement*" and "*formAdminRoad*", while it is connected with only one ObjectProperty to the *Road* class (that is "*coincideWith*"). To better clarify the relationship existing among the classes *Road*, *AdministrativeRoad* and *RoadElement* it can be stated that an instance of the *Road* class can be connected to multiple instances of *AdministrativeRoad* class (e.g. if a road crosses the border between two provinces), but the opposite is also true (e.g. when a road crosses a provincial town center and assumes different names), i.e. there is a N:M relationship between this two classes. On each road element is possible to define access restrictions identified by the class *EntryRule*, which is connected to the *RoadElement* class through two inverse ObjectProperties, i.e. "*hasRule*" and "*accessToElement*". The *EntryRule* class is defined with a restriction on the minimum cardinality of ObjectProperty "*accessToElement*" (set equal to zero), because in most cases, each rule is associated to one road element, 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 RTZ), but also on maneuvers; for this reason, the class *Maneuver* has been defined and connected to the class *EntryRule* by the "*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. Thanks to a deeper analysis of Street Guide data, it has been verified that only in rare cases maneuvers involving three different road elements and then to represent the relationship between the *Maneuvre* and *RoadElement* classes in the simplest way, three ObjectProperties were defined: "*hasFirstElem*", "*hasSecondElem*" and "hasThirdElem", in addition to the ObjectProperty that binds a maneuver to the junction that is interested, that is, "*concerningNode*" (because a maneuver takes place always in the proximity of a node). Defining the *Maneuvre* class, cardinality restrictions have been specified: "*hasFirstElem*" and "*hasSecondElem*" ObjectProperty have a cardinality restriction set equal to 1 while the maximum cardinality associated to the

ObjectProperty "*hasThirdElem*", is set to 1, in fact, for the maneuvers that affect only two 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, and therefore also a *Node* class has been defined, as a subclass of the same name class belonging to ontology *OTN*. The *Node* class presents restrictions on DataProperties *geo:lat* and *geo:long*, two properties inherited from the definition of the *Node* class as subclass of *geo:SpatialThing*, a class belonging to the *Wgs84* ontology: in fact, each node can be associated only with one pair of coordinates in space, and cannot exist a node without these values, then the cardinality of these DataProperties is equal to one.

The *Milestone* class represents the kilometer stones that are placed along the administrative roads, that is, the elements that identify the precise value of the mileage at that point, i.e. the advanced of the route from the starting point. A milestone may be associated with a single instance of *AdministrativeRoad* class, and it is therefore defined a cardinality restriction equal to 1, associated to the ObjectProperty "*placedInElement*". Like the *Node* class, also the *Milestone* class is defined as subclass of *geo:SpatialThing*, but in this case the presence of a couple of coordinates, is not mandatory (a restriction on maximum cardinality is set to be equal to one, but this not exclude a possible lack of value).

A street number is used to define an address, more accurately, 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, but sometimes, it can also correspond to an internal access. Looking at this relationship from the access point of view, instead, it is possible to say that each access is logically connected to at least one street number. According to what is stated, the classes *StreetNumber* and *Entry* were defined.

With the owned data the class *StreetNumber* can be connected to the class *RoadElement* and to the class *Road*, respectively through the ObjectProperties "*placedInElement*" and "*belongToRoad*". Furthermore, for the ObjectProperty "*belongToRoad*" an inverse property "*hasStreetNumber*", has been also defined.

Within the ontology therefore, for the *StreetNumber* class a cardinality restriction on the ObjectProperty "belongToRoad", which must be equal to 1, has been defined.

The *Entry* class can be also connected to both the street number and road element where it is located. The relationship between classes *Entry* and *StreetNumber*, is identified thanks two ObjectProperties, that is "*hasInternalAccess*" and "*hasExternalAccess*", on which cardinality restrictions have been defined, because, as mentioned earlier, a street number will always have only one external access, but in some cases, could also have an internal access (this latter restriction is in fact defined by setting its maximum cardinality equal to 1, i.e. only values 0 and 1 are allowed). Like *Node* and *Milestone* classes, also the *Entry* class is defined as a subclass of *geo:SpatialThing* class, so it is possible to associate a maximum of one pair of coordinates, *geo:lat* and *geo:long*, to each its instance: this can be translated with a restriction on the maximum cardinality of this two DataProperty of the *Geo* ontology, set to one (each DataProperty may take one value, or none).

The *Street Guide and Rail Network* macroclass is connected to the Administration one, through two different ObjectProperties "*ownerAuthority*" and "*managingAuthority*", which as the name suggests, they represent respectively the public administration to which the administrative road belongs, or the public administration that manages the road element. In this representation, only private roads remain out, because information contained into datasets, related to private property, are limited.

From a cartographic point of view, however, each road element is not a straight line, but a broken line, which follows the actual course of the road. To represent this situation, the classes *RoadLink* and *Junction* have been defined: thanks to the interpretation of the KMZ file, the set of coordinates that define each *RoadElement*, is recovered and each of these points is then added to the ontology as an instance of the class *Junction*, defined as a subclass of *geo:SpatialThing*, with therefore a pair of coordinates. Each small segment between two junctions, represents, instead, an instance of the class *RoadLink*, which is defined through a restriction on the cardinality of ObjectProperty "*endJunction*" and "*startJunction*" (setting equal to 1) that connect this two classes.

Connect to *Road* class there is also the *TrafficGate* class, whose instances represent the access gates to the RTZ (Restricted Traffic Zone) or to lanes only for bus.

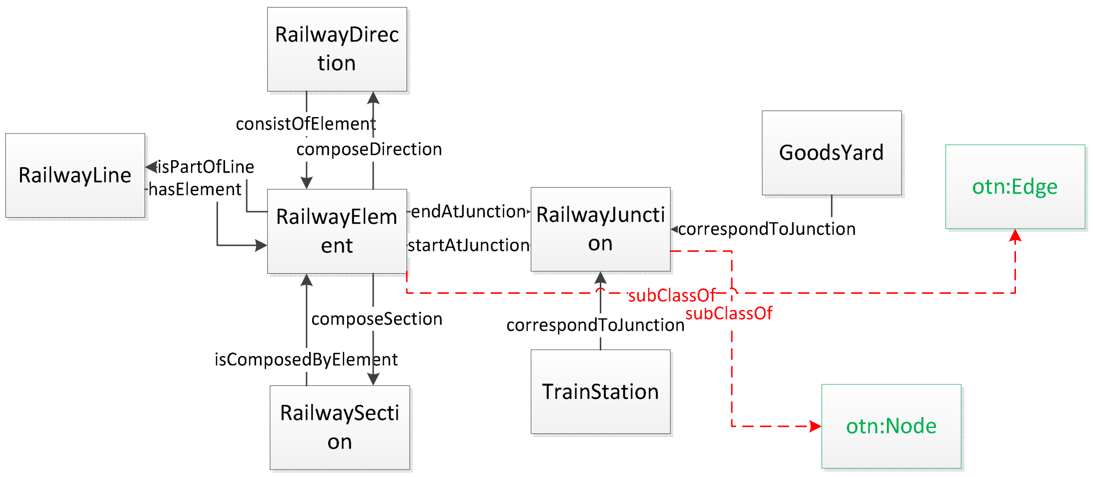


Figure 17 - Rail Network Macroclass of Km4City Ontology

The second part of this macroclass corresponds to the Railway Graph, that is shown in Figure 17. It is mainly formed by class *RailwayElement* which is defined as a subclass of the *OTN:Edge* class. The railway elements can be assembled to form the following railway components:

- *Railway direction*: 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;
- *Railway section*: a section of the line in which is possible to find only one train at the time, that is usually preceded by a "protective" or "block".
- *Railway line*: i.e. the infrastructure that allows trains or other railway convoys to travel between two places of service.

For each of this components, a class inside the Knowledge Model, have been defined, that is the classes *RailwayDirection*, *RailwaySection* and *RailwayLine*. The first one can be connected to the *RailwayElement* class thanks to two inverse ObjectProperties "*consistOfElement"* and "*composeDirection"*; the second one is instead connected to the *RailwayElement* class, through two other inverse ObjectProperties, i.e. "*isComposedByElement"* and "*composeSection"*; finally the last class, the class *RailwayLine*, is connected to railway elements using the ObjectProperty "*hasElement"* and its inverse "*isPartOfLine"*.

Each instance of class *RailwayElement* is connected to two instances of class *RailwayJunction* (defined as a subclass of the *OTN:Node*), by the ObjectProperties "*startAtJunction"* and "*endAtJunction"*.

Classes *TrainStation* and *GoodsYard* represent respectively a train station and a freight station, and both correspond only to one instance of the *RailwayJunction* class, both through the ObjectProperty "*correspondToJunction"*.

4.4.3 Points of Interest Macroclass

For the third macro class, that is, *Points of Interest*, a generic class *Service* has been defined, which can represent, as mentioned before, all services or activities, which may be useful to the citizen and who may have the need to “search-for” and to “arrive-at”. This macroclass allows to represent services to the citizens, points of interest, businesses activities, tourist attractions, and anything else can be located thanks to a pair of coordinates on a map.

The classification of individual services and activities is based on main and secondary categories planned at regional level: for this purpose the "*hasServiceCategory*" ObjectProperty and the corresponding class *ServiceCategory*, have been created. This last class can only take the exactly values, represented by individuals defined inside the Knowledge Model, that have been found on the data provided by the Tuscany Region.

The categories of the Tuscany Region are expressed in Italian, therefore a process of translation was necessary, which resulted in creating a dictionary, to which translations into other languages can easily be added. A better classification of services and activities could be implemented thanks to the ATECO code, i.e. a code representing the ISTAT classification of economic activities, but unfortunately into the data provided, a high percentage of ATECO code is missing.

Analyzing the available data and taking into account the services classification available within the OTN ontology, the following subclasses have been identified: *Accommodation*, *GovernmentOffice*, *TourismService*, *TransferService*, *CulturalActivity*, *FinancialService*, *Shopping*, *Healthcare*, *Education*, *Entertainment*, *Emergency* and *WineAndFood*. The Accommodation subclass for example, is defined as a restriction on the ObjectProperty *"hasServiceCategory"*, that must take one of the following individual values: *holiday_village*, *hotel*, *summer_residence*, *rest_home*, *religiuos_guest_house*, *bed_and_breakfast*, *hostel*, *farm_house*, *agritourism*, *vacation_resort*, *day_care_center*, *camping*, *historic_residence*, *mountain_dew*, *boarding_house*. Similarly, all other subclasses listed above, have been defined, including among the possible values of the ObjectProperty *"hasServiceCategory"*, those corresponding to that type of service.

Some classes, to identify particular services that citizens may have need to reach, but that may be offered by different types of activities, were also defined: for example, there are some POI that also offer a wireless internet connection, for which the *HotSpotWifi* class was defined, which is connected to the *Service* class via the ObjectProperty "*isInstalledOnService"*; that class is also connected to the Street Guide because some hot spot wi-fi in the city of Florence are installed along roads and not in correspondence to an activity. Another additional service is the ticket sales, for which, in fact, the class *BusTicketsRetail* has been added, connected to the *Service* class thanks to the ObjectProperty "*isATicketsRetail"*.

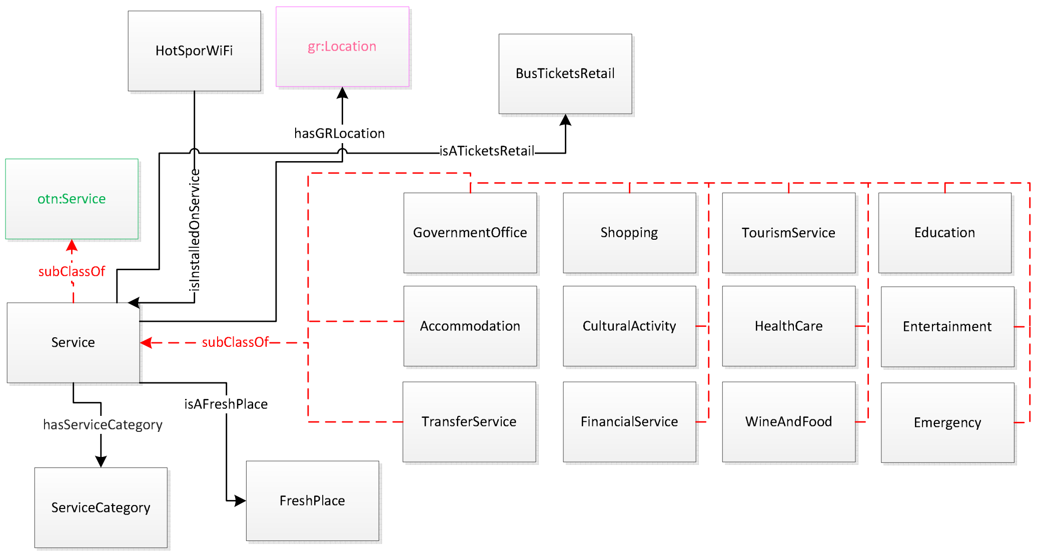


Figure 18 - Point of Interest Macroclass of Km4City Ontology

The city of Florence has also a list of cool places where citizens can stay in the summer to freshen up a bit; so, the *FreshPlace* class has been also included into the ontology, which is connected to the class *Service* using the ObjectProperty "*isAFreshPlace*".

In Figure 18 is represented a possible integration between our knowledgemodel Km4City and the *GoodRelations* ontology, achieved at class level: the classes *km4c:Service* and *GoodRelations:Locality* can be identifying as equivalents from a definition point of view and, for example, the service "*Ristorante il Pozzo*" can be connected to the respective locality, defined in the *GoodRelations* ontology, through the ObjectProperty *km4c:hasGRLocality*.

4.4.4 Local Public Transport Macroclass

The fourth macroclass, called *Local Public Transport* macroclass, includes the data related to major LPT (Local Public Transport) companies scheduled times, the rail graph, and data relating to real time passage at bus stops. This macroclass is not complete yet, it contains data related to routes and bus stops of all Tuscany Region, but it contains at the moment, only bus timetable data related to the metropolitan area of Florence, but the data model proposed can be used for all data of the rest of Tuscany.

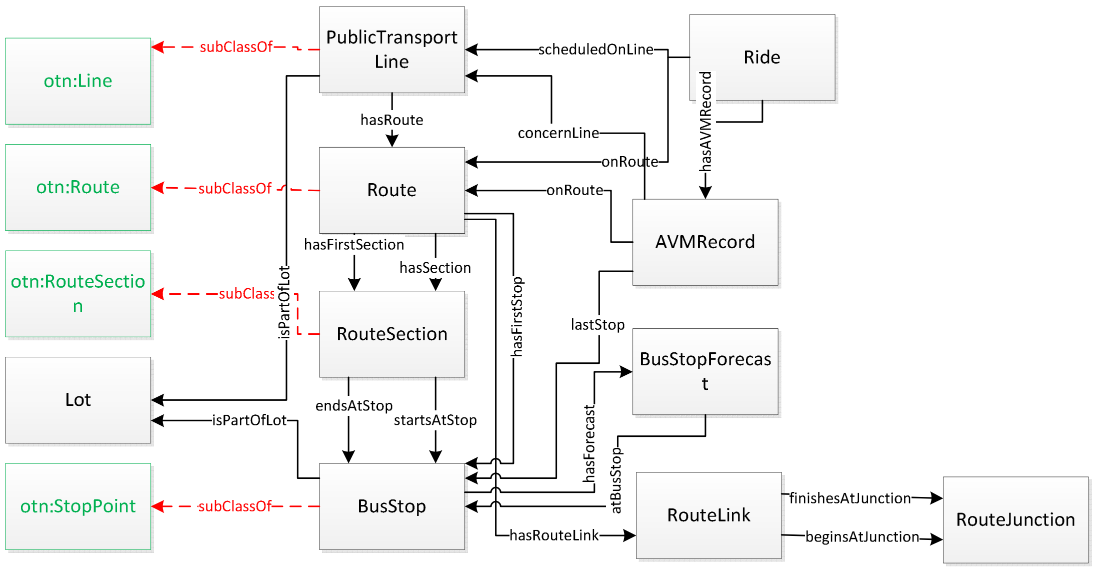


Figure 19 - Local Public Transport Macroclass of Km4City Ontology

The public transport by road is organized in public transport lots, represented by *Lot* class, each of which is in turn composed of a number of bus and tram lines, i.e. the class *PublicTransportLine*. The relationship between this two classes corresponds to the ObjectProperty "*isPartOfLot*", which connects each instance of *PublicTransportLine* class to the corresponding instance of *Lot* class. The *PublicTransportLine* class is defined as a subclass of *OTN:Line*.

Each line includes at least two ride per day (the first in ascendant direction, and the second one in descendant direction), identified through a code provided by the Local Public Transport company and each ride is scheduled to drive along a specific path, called route. Accordingly, the classes *Ride* and *Route* have been created, together with the ObjectProperty "*scheduledOnLine*", connecting the *Ride* class with the *PublicTransportLine* class, which is defined as a limitation of cardinality exactly equal to one, because each ride code may be associated to a single line. The case in which there are more than two rides for the same line occurs when, for example, some shortest paths are scheduled, in certain hours of the day.

A route can be seen as a series of road segments delimited by subsequent bus stops, but wishing then to represent to a cartographic point of view the path of a bus, 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 on road elements, has been reused: we can see each path as a set of small segments, each of which delimited by two junctions. According to what is stated two ObjectProperty linking the classes *Route* and *RouteSection*, has been defined, named "*hasFirstSection*" and "*hasSection*" and also two other ObjectProperty "*endsAtStop*" and "*startsAtStop*", connecting each instance of the *RouteSection* class to two instances of the *BusStop* class. This last class is defined as a subclass of *OTN:StopPoint* class.

Thanks to the defined model, knowing the starting bus stop and the first segment, the entire route taken by each means of transport can be reconstructed. For this purpose the ObjectProperty "*hasFirstStop*" has been also defined, which connects the *Route* and *BusStop* classes. Each stop is also connected to the class *Lot*, through the ObjectProperty "*isPartOfLot*", with a relation 1:N because there are stops shared by urban and suburban lines so they belong to two different lots.

Although the class *BusStop* is a subclass of *geo:SpatialThing*, since the data provided by the Local Public Transport companies contain a pair of coordinates that allows to locate them; inside the *Km4city* knowledge model the *BusStop* class is defined thanks to a cardinality restriction, set exactly equal to 1, on the DataProperty *geo:lat* and *geo:long*.

Wishing then to represent to a cartographic point of view the path of a bus, i.e. an instance of the *Route* class, a broken line must be represented, which composes each stretch of road crossed by the means of transport itself and to do so, the previously used modeling with the road elements, has been reused: each small segment composing the path, is delimited by two junctions: therefore the *RouteLink* and *RouteJunction* classes were then defined, together with the ObjectProperty "*beginsAtJunction*" and "*finischesAtJunction*". The *RouteLink* class was defined with a cardinality restriction on both just mentioned ObjectProperty, imposing that it is always equal to 1. The *Route* class is instead connected to the *RouteLink* class through the "*hasRouteLink*" ObjectProperty.

4.4.5 Sensors Macroclass

The *Sensors* Macroclass has not yet been completed, and actually it consists of the four parts shown in Figure 20, respectively relating to the car parks sensors, to the weather sensors, to the sensors installed along roads and rails, and to the AVM (Automatic Vehicle Monitoring) systems installed on buses, cars and/or bikes.

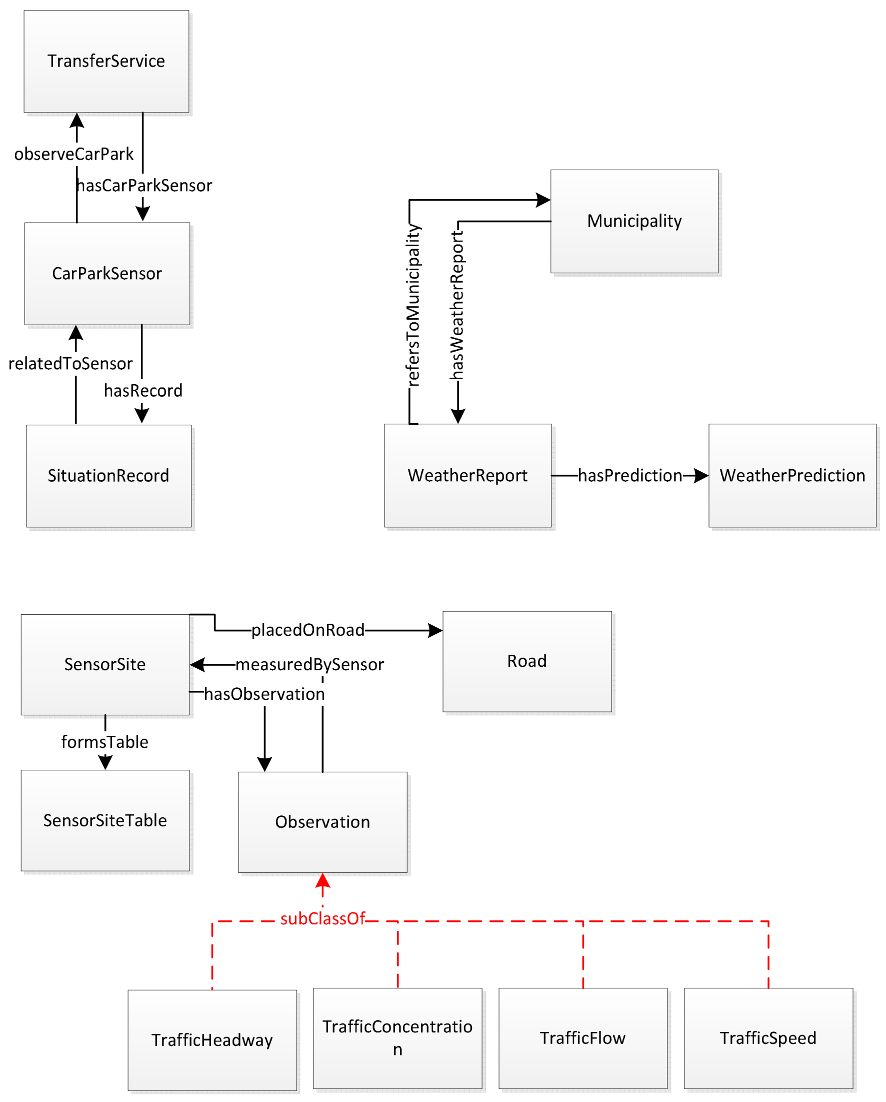


Figure 20 - Sensors Macroclass of Km4City Ontology (from the top left corner, parking sensors, weather sensors, Road sensors)

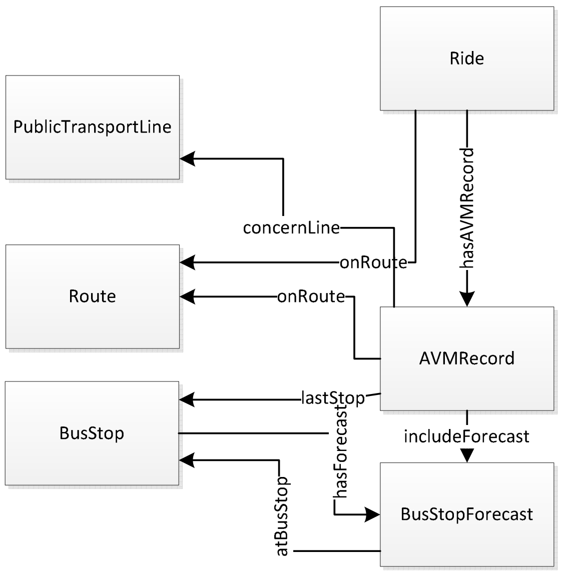


Figure 21 - Sensors Macroclass of Km4City Ontology (AVM sensors)

The first part is focused on the real-time data related to parking: for each sensors installed into different car parking areas, a status record is received every 5minutes. This scenario has been translated into ontological model, defining *CarParkSensor* class, which represents the sensor installed in a given parking and the *SituationRecord* class, which instead represents the state of a certain parking at a certain instant. The connection between the two classes just mentioned, is performed via the reverse ObjectProperties, "*relatedToSensor*" and "*hasRecord*". In each status report, there are information about the number of free and occupied parking spaces, for the main car parks in Tuscany Region. The *CarParkSensor* class is also connected to the *TransferService* class, belonging to the *Point Of Interest* macroclass, connection between different macroclasses is realized through two inverse ObjectProperties, i.e. "*observeCarPark*" and "*hasCarParkSensor*".

The weather sensors produce real-time data concerns the weather forecast, thanks to *LAMMA*. This consortium updates the municipality forecast report twice per day and every report contains forecast for five days divided into range, which have a greater precision (and a higher number) for the nearest days until you get only a single daily forecast for the 4th and 5th day. This situation is in fact represented by the *WeatherReport* class connected to the *WeatherPrediction* class via the ObjectProperty "*hasPrediction*". *Municipality* class is instead connected to a report by two inverse ObjectProperties: "*refersTounicipality*" and "*hasWeatherPrediction*".

The traffic sensors, represented by instance of *SensorSite* class, produce real-time data concerning the sensors placed along the roads of the region, which allow making different measures and assessment related to traffic situation. Unfortunately, the location of these sensors is not very precise, it is not possible to place them in a unique point thanks to coordinate, but only to place them within a toponym, which for long-distance roads such as FI-PI-LI road (the highway that connect Florence-Pisa-Livorno), it represents a range of many miles. Each sensor, is part of a group, represented by the *SensorSiteTable* class, and produces observations, represented by instance of *Observation* class, which can belong to four types, i.e. they can be related to the average velocity (*TrafficSpeed* subclass), car flow passing in front of the sensor (*TrafficFlow* subclass), traffic concentration (*TrafficConcentration* subclass), or to the traffic density (*TrafficHeadway* subclass). On this regards, Bluetooth sensors could be installed to trace the number of people passing by on car and bikes from a given point.

The connection between *SensorSite* and *SensorSiteTable* classes corresponds to the ObjectProperty "*formsTable*" and, as mentioned earlier, each instance of *SensorSite* class can be connected only to the *Road* class, thanks to the ObjectProperty "*placedOnRoad*". The classes *Observation* and *SensorSite* are instead connected via a pair of inverse ObjectProeprty, "*hasObservation*" and "*measuredBySensor*".

The AVM (Automatic Vehicle Monitoring) systems part concerns the sensors systems installed on most of buses, which, at intervals of few minutes, send a report to the management center. They provide information about: the last stop performed, current GPS coordinates of the vehicle, the identifiers of vehicle and of the line, a list of upcoming stops with the planned passage time. The described situation is mainly represented by two classes, *AVMRecord* and *BusStopForecast*: the first class mentioned represents the report sent by the AVM system, which contain a list of upcoming stops with the planned passage time, represented by instances of *BusStopForecast* class, and other information like the last stop done, GPS coordinates of the vehicle position, and the identifiers of vehicle and line. To keep track of the last stop carried out, the "*lastStop*" ObjectProperty has been defined, and it create a connection between the classes *AVMrecord* to *BusStop*; also the *BusStopForecast* class is connected to the *BusStop* class, through the "*atBusStop*" ObjectProperty instead, the *AVMRecord* class is linked to the *Line* class via the ObjectProperty "*concernLine*".

4.4.6 Temporal Macroclass

The Temporal Macroclass, is now only sketchy within the ontology, that is, its definition is complete inside the ontology but actually its potential has not been fully exploited. It is based on the Time ontology (http://www.w3.org/TR/owl-time/) but also on experience gained in other previous projects such as OSIM [Bellandi et al., 2012]. The integration of temporal concepts within *Km4city* Ontology is required because it allows to calculate differences between time instants and to be able to order them, and thanks to the *Time* Ontology, the definition of these concepts is greatly simplified.

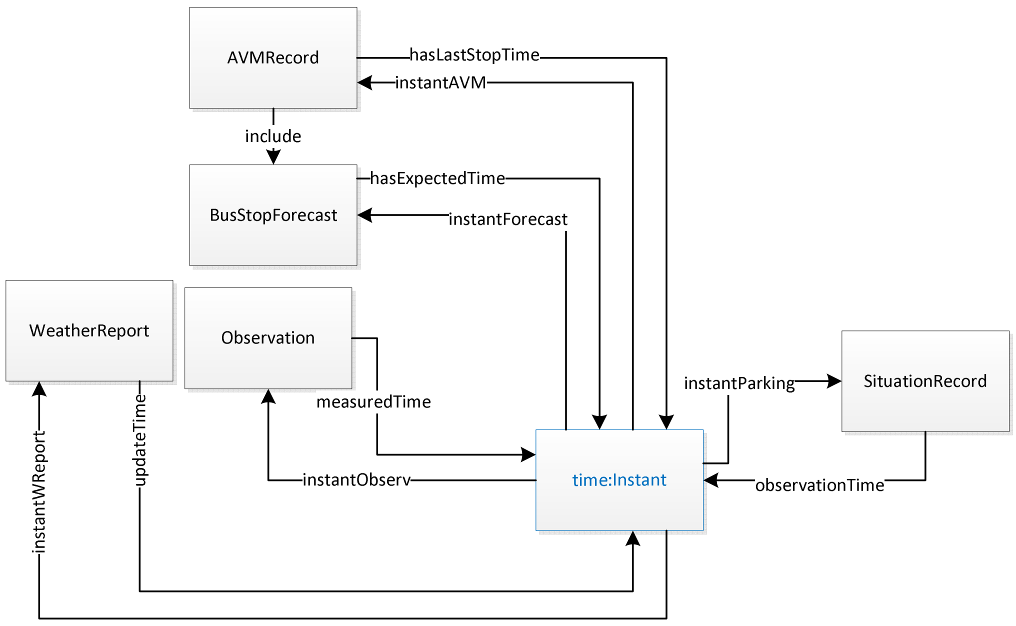


Figure 22 - Temporal Macroclass of Km4City Ontology

In fact the fictitious URI #instantForecast, #instantAVM, #instantParking, #instantWreport, #instantObserv are defined to following associate them to each URI which identifier a resource referred to the time parameter, i.e. respectively *BusStopForecast*, *AVMRecord*, *SituationRecord*, *WheatherReport* and finally *Observation*.

The fictitious URI #*instantXXXX* (in real cases, the *X*'s will be replaced with a time stamp), is formed as concatenation of two strings: for example, in the case of *BusStopForecast* instances the stop code string (which allows us to uniquely identify each stops) and the time instant to which the forecast is referred (in the most appropriate time format), are concatenate. To create a fictitious URI that links a time instant to each resource is a necessary action 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) .

The *Time* ontology helps define time instants as temporal information punctual and then allows to use them as extreme in the definition of intervals, a feature very useful to increase the expressiveness.

Pairs of ObjectProperties have also been defined for each class that needs to be connected to the class Instant: between classes *Instant* and *SituationRecord* the inverse ObjectProperties "*instantParking*" and "*observationTime*" have been defined, between classes *WeatherReport* and *Instant*, there are instead "*instantWReport*" and "*updateTime*" ObjectProperties, between classes *Observation* and *Time* the inverse

ObjectProperties "*measuredTime*" and "*instantObserv*" have been defined, between *BusStopForecast* and *Time* the ObjectProperties "*hasExpectedTime*" and "*instantForecast*" have been created, and finally, between classes *AVMRecord* and *Time*, there are the inverse ObjectProperties "*hasLastStopTime*" and "*instantAVM*".

The domain of all ObjectProperties with name like *instantXXXX* is defined by elements of class *Time:temporalEntity*, so that these properties can be extended not only to instants, but also at time intervals.

4.4.7 Context Macroclass

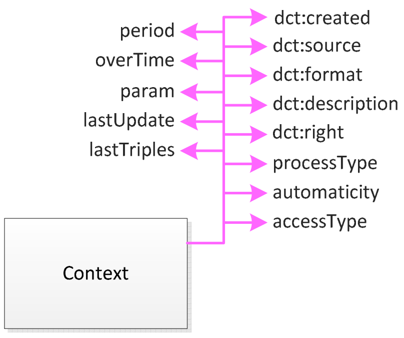


Figure 23 - Context Macroclass of Km4City Ontology

The seventh macroclass, as already mentioned above, relates to the metadata associated with each dataset. *Sesame* [Sesame] 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 defined as follow: subject-predicate-object-*context*.

OWLIM (http://owlim.ontotext.com/) 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, type of license bound to the dataset, kind of ingestion process, and how much automated is the entire ingestion process, type of access to the dataset, overtime, period, associated parameters, update date, triples creation date.

4.4.8 DataProperties of the main classes

Inside the ontology many DataProperties were defined, i.e. all those for which there was no possibility of using the values already defined within the reused vocabularies.

This paragraph analyze the main ontology classes and the corresponding DataProperties defined, and it also lists the possible values for all DataProperty that can take on only a specific set of values.

Within the class *PA* only three DataProperties were used, all previously defined in the vocabulary reused: *foaf:name* that represents the name of the public administration represented by the considered instance, its unique identifier present in the regional system i.e. *dct:Identified*, from DublinCore ontology (http://dublincore.org/specifications/) and finally *dct:alternative* where the municipal code present in the tax code, is stored. More information about PA can be found through the link to the corresponding instance of the *Service* class, where in fact there are more details that would otherwise be redundant. The *Resolution* class, whose instance as seen above are the municipality resolutions, has some DataProperties that allows to identified each instance, *dct:Identified*, the year of resolution approval, *km4c:year*, and some other property coming from the DC ontology like *dct:subject* (the resolution object), *dct:created* (the date on which the PA has resolved) and *foaf:page* that represents the URL to which the resolution is available online.

Each instance of the *Route* class is uniquely identified using the DataProperty *dct:Identified* where the toponym identifier for the entire regional network is stored, that consist of 15 characters and it is defined according to the following rule: *RT* letters 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 *km4c:roadType* instead, represents the type of toponym, for example:

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

Inside the Street Guide there are also two name fields for each toponym: the name and the extended name, which also includes the toponym's type. The name without type, is a string associated with the DataProperty *km4c:roadName*, while the long name, is store in another DataProperty i.e. *km4c:extendexName*; the name of each road can be written in different ways, that is there are alias, for example: for *Via S. Marta* a possible alternative could be *Via Santa Marta*, *Via di S. Marta*, *Via di Santa Marta*, etc. The Kmc4city ontology therefore provides the possibility to store them within a variable number of *dct*:*alternative* DataProperty, so that the later process of reconciliation can be facilitated.

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

- a highway
- a regional road
- a road
- a municipal road
- a military Road
- a driveway

Finally the field *dct:identifier* stores 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 always formed by 15 characters as follows: *RT* letters 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 *RoadElement*, and it can take the following values:

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

In the Street Guide of the Tuscany Region, the functional classification is also associated to the *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". The DataProperty *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 width of each road element, a reference to DataProperty *km4c:width* is made, which allows to detect the width band of belonging: "*less than 3.5m*", "*between 3.5 and 7.0m*," "*greater than 7.0 meters*" or "*not detected*"; for the length of each road element instead, the reference DataProperty is *km4c:length*, a freely insertable value that does not refer to any band.

Among other data available in Street Guide there is the direction of travel, essential to define the maneuvers permitted, corresponding to the *km4c:trafficDir* DataProperty, which may take one of the following four 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 DataProperty *km4c:operatingStatus* instead, is used to track the operating status of the different road elements and it can take only the values "*in use*", "*under construction*" or "*abandoned*". Finally the last DataProperty of the *RoadElement* class is a DataProperty that takes into account the speed limits on each road element, called *km4c:speedLim*.

Into the class *StatisticalData* the following DataProperties were defined: a DataProperty *km4c:value* to store the actual value of the statistic, and *dct:description*, *dct:created* and *dct:subject* where data necessary to maintain intact the statistic meaning, are stored.

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; so each instances of the *Node* class can be uniquely identified thanks to the DataProperty *dct:Identified*, made, like the previous identifier code, of 15 characters, according to the following rules: the first two letters are *RT*, followed by the 6-character ISTAT code of municipality where the node is located, followed by 5 characters of progressive starting from the value of ISTAT code, and finally letters *GZ*. 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

However the fundamentals DataProperty are *geo:lat* and *geo:long*, thanks to which a node can be localized with precision on a map.

The access rules, as mentioned in previous section, are described by instances of the *EntryRule* class, uniquely identifiable through a *dct:identified* of 15 characters thus formed: the *RT* letters 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 letters *PL*. The access rules are then characterized by a type, represented by DataProperty *km4c:restrictionType,* which can assume one of 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, 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

- o 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. Within the Streets Guide, there are other information associated to maneuvers such as the operation type, bifurcation type and maneuver type prohibited, but since the last two types are almost always "undefined", only a DataProperty has been defined, associated with the information of maneuver type, precisely named *km4c:maneuvreType*, which can take the following values:

- Fork
- Calculated forbidden maneuver
- Mandatory maneuver
- Forbidden maneuver
- Priority maneuver

The *StreetNumber* class, also presents a code *dct:Identified*, to uniquely identify each instance of the class, in the same format of those seen previously: the *RT* letters followed by 6 characters for the ISTAT code of the municipality, 5 other characters for the progressive from the ISTAT code and finally the letters *CV*. In Florence there are two different numberings, associated to a different color, i.e. red and black; so a street may have, for example, 4/Black and 4/Red, where red is the color for the numbering system for shops.

Therefore defined a DataProperty, called *km4c:classCode*, has been defined, in which the information just seen can be stored and that can take the following values: *red*, *black*, or *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 of the ontology. An additional value, the DataProeprty *km4c:extendNumber*, is also been defined, in which is stored the number together with its exponent in order to ensure greater compatibility with the different formats in which instances of this class could be written/researched.

The *Milestone* class, as seen in Section 4.4.2, 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 the DataProperty *dct:Identified* and formed as follow: letters *RT*, followed by 6 characters for the ISTAT code of the municipality, other 5 characters for the progressive from ISTAT code, and finally, the letters *CC*.

Inside the DataProperty *km4c:text*, the mileage, which corresponds to that point, is written; thanks to the information contained into the Street Guide, the name of the street, highway, etc. where the milestone is located, can be retrieved, by passing from the class *RoadElement*, to which the class *Milestone* is directly connected. Also in this case DataProperties *geo:lat* and *geo:long*, for localization, are defined.

The *Entry* class contains the point element that identifies, directly or indirectly external access to a specific place of residence/business on the territory; each entry in practice materialized the "plate" of the street number. As previously mentioned, each entry is logically connected to at least one number. Each instance of the *Entry* class is uniquely identified by DataProperty *dct:identified*, consisting of a code of 15 characters, like all other codes seen: letters *RT* followed by 6 characters of municipality ISTAT code, then another 5 character of the progressive from ISTAT code and finally the *AC* letters. There are only three types of accesses, and this value is stored into the DataProperty *km4c:entryType*: "*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 DataProperties to store coordinates *geo:lat* and *geo:long*, are present.

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

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

The class *RailwayElement*, has the same field *dct:identifier* of the previous two classes examined, consisting of 12 characters that follow this 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 property, the *km4c:elementType* has been defined, which can take only three values:

- ordinary railroad
- railroad AC/AV
- other

Another DataProperty of the class *RailwayElement* is *km4c:operatingStatus*, which can take only tone of the following values:

- railway construction
- railroad in operation
- disused railway

The DataProperty *km4c:elemLocation* indicates, instead, the rail element location and its possible values are:

- grade-level
- on bridge/viaduct
- in tunnel

Continuing to analyze the DataProperties defined for the *RailwayElement* class, there are the *km4c:supply* DataProperty that specifies whether there is an "*electrified line*" or a "*non-electrified line*”, the DataProperty *km4c:gauge*, i.e. a field that specified if the gauge is "*reduced*" or "*standard*", and the *km4c:underpass* which can take the following values:

- the item is not in underpass of any other object
- the element is in underpass of another object
- the element is simultaneously in overpass and underpass of other objects

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

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

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

Were then defined DataProeprties *dct:identifier*, i.e. the usual code 12 characters that begins with *RT* letters, followed by the regional code *T09*, 5 number for the sequential number and that ends with letters *TR*), the *foaf:name* containing the naming convention of line and *km4c:combinedTraffic*, which can assume the values:

- PC80
- PC60
- PC50
- PC45
- PC32
- PC30

- PC25
- PC22
- lines with the loading gauge FS
- undefined

For the *RailwayJunction* class, only three DataProperties has been defined: *dct:identifier*, that is the identification code of 12 characters format as in previous cases, but ending with the letters *GK*, *foaf:name*, containing the official name of the junction, stations or rail yard, and finally the DataProperty *km4c:juncType*, which can take one of the following values:

- rail crossing
- terminal (beginning or end)
- junction (junction or branch)
- station / stop / rail toll
- freight
- interporto
- change of state (COD_STA)
- change of venue (COD_SED)
- variation in the number of tracks (Num_bin)
- power variation (COD_ALI)
- administrative boundary

The class *TrainStation* presents the usual 12 characters DataProperty *dct:identifier*, consisting of *RT* letters followed by 3 characters for the regional identification - *T09* for Tuscany - 5 characters of progressive number and finally the letters *SF*; for this class also the DataProperty *foaf:name* has been defined, in which the official name of the train station is stored; the address retrieved from the list posted on the RFI's website is instead stored into the fields *schema:streetAddress*, *schema:postalCode*, *schema:addressLocality*, *schema:addressRegion*, and the managing body always found on RFI's website, is stored into the DataProperty *km4c:managingAuth*; the *km4c:state* DataProperty, contains the state of the station which can take only the values

- Active
- not Active
- optional stops on demand.

Finally, the DataProperty *km4c:category* contains the category to which the station belongs: the idea of the stations classification arises because they are open to the public and, through the analysis of some parameters, it is possible to distinguish the potential of a plant in terms of functionality, comfort and safety. The parameters used in the evaluation are the size of the plant, its attendance, the ability to interchange and the level of commercial offer, so 4 possible values for the DataProperty *km4c:category* are obtained, which in order of importance are:

- Platinum
- Gold
- Silver
- Bronze

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

The *Service* class has been equipped with the contact card provided by the *Schema.org* ontology (https://schema.org), with the aim to make the description of the various companies more standardized. The contact card is composed by the following DataProperties, and it is easy to understand the information each of which contains:

- schema:name;
- schema:telephone;
- schema:email;
- schema:faxNumber;
- schema:url (to store the Company's WebSite address);
- schema:streetAddress;
- schema:addressLocality;
- schema:postalCode;
- schema:addressRegion;
- skos:note (to store any additions such as the opening hours of an activity sometimes present in the data).

Besides DataProperties inherited from *Schema.org*, other DataProperties were defined for the *Service* class: *km4c:housenumber* to isolate the street number from street address, the *km4c:atecoCode*, for storing the corresponding Ateco code of each service, and, when possible, the DataProperties *geo:lat* and *geo:long* for localization.

The class *CarParkSensor* has a unique identifier stored into *dct:identified* DataProperty, always defined at the regional level through a 15 characters code beginning with letters *RT* and ending with the initials of the belonging province, for example, *FI* for the city of

Florence; this class also has *km4c:capacity* DataProperty, i.e. the total number of parking places of the examined car park.

The *SituationRecord* class, instead, contains properties closely related to car parks, that are *km4c:fillrate* and *km4c:exitrate*, respectively the number of vehicles entering/leaving the parking; *km4c:carParkStatus*, i.e. a string that describes the current state of the car park which possible values are:

- "enoughSpacesAvailable"
- "carParkFull"
- "noParkingInformationAvailable"

Other DataProperties belonging to the same class are *dct:identified*, i.e. a unique identifier for the report, the *km4c:validityStatus* DataProperty that represent a validity status of the record, which can only have the value "*active*" for parking, the *km4c:parkOccupancy*, i.e. the number of occupied space, or the corresponding percentage that is instead called *km4c:occupied*, and last the free places numbers stored into the DataProperty *km4c:free*.

The class *WeatherReport* is characterized by DataProperty *dct:identified* containing the unique id which identifies the different reports, *timestamp* that indicates the time when the report has been created in milliseconds. Other DataProperties have also been added, relative to the phase of the moon, and the hour when the sun and moon, rise and set which are *km4c:lunarphase*, *km4c:sunrise* and *km4c:sunset* and *km4c:moonrise* and *km4c:moonset*. The last two DataProperties defined for *WeatherReport* class, are *km4c:heightHour* and *km4c:sunHeight*, which represent the time when the sun reaches its maximum height and at which height.

Each instance of *WeatherPrediction* class is instead characterized by DataProperties like *km4c:day*, which is the day referred in prediction, the minimum and maximum temperature values stored into the DataProperties *km4c:minTemp* , *km4c:maxTemp*, the real and perceived temperature values that correspond to *km4c:recTemp* and *km4c:perTemp* DataProperties. The day, together with the *WeatherReport* identifier, form a new unique way to identify each forecast and this new value is stored into the DataProperty *dct:identifier*. Into the class *WeatherPrediction* there are also some DataProperties related to atmospheric parameters, such as *km4c:wind* that store the wind direction, *km4c:humidity* containing the percentage of humidity, *km4c:snow* describing the snow bulletin, *km4c:hour* representing the part of the day that is referenced by each individual forecast, and finally the UV index of the day, stored into the *km4c:UV* DataProperty.

The *SensorSite* and *Sensor* classes have only the DataProperty concerning their id, represented by *dct:identified*. The connected class *Observation* instead, is completed by DataProperties *dct:identified*, *dct:date*, both from DC ontology, respectively the identifier of each sensors observation and the day to which it refers; other DataProperties of this class are *km4c:averageDistance* and *km4c:averageTime*, representing average distance and average time between passage of two cars, *occupancy* and *concentration*, concerning the percentage of occupation of road the first one, and referred to the car concentration, Finally *vehicleFlow* is the DataProperty referring the flow of vehicles detected by the sensors and data related to the average velocity and the calculated speed percentile, are stored into three DataProperties *km4c:averageSpeed*, *km4c:thresholdPerc* and *km4c:speedPercentile*.

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

The *Route* class rather than *dct:identifier* and *dct:description* DataProperties, presents the field *km4c:routeLenght*, that is the route length in meters, and *km4c:direction*, i.e. the route direction.

The class *BusStop* has, in addition to DataProperty *dct:identifier*, the datProperty *foaf:name* containing the name of the bus stop, and a pair of coordinates *geo:lat* and *geo:long*, belonging to *Geo* Ontology (http://www.w3.org/2003/01/geo/). These last two DataProperties, together with *dct:identifier*, are the only ones in the *RouteJunction* class.

For the class *BusStopForecast* only DataProperties for the time of arrival and the identification, respectively named *km4c:expectedTime* and *dct:identifier*, have been defined.

The *AVMRecord* class requires instead DataProperties to identify the means to which the record refers, that is *km4c:vehicle*, the arrival time to last stop, i.e. *km4c:lastStopTime*, the ride state, that is, *km4c:rideStatus* which can only take one of the following values: "*early*", "*late*" or "*in time*". Inside DataProperties *km4c:managingBy* and *km4c:owner*, information about the managing company and the company that own the AVM system are stored; each instance of this class is uniquely identified by the *dct:identifier* DataProperty, and *geo:lat* and *geo:long*, indicated the exact vehicle position at the report time.
Continuing with the macroclass on the public transport, the class *Ride* has only the *dct:identifier* DataProperty, like the *RouteLink* class.
The *RouteSection* class, instead, has only the DataProperty *km4c:distance*, where the distance between two successive stops within a route, is saved.

The *TrafficGate* class, which as mentioned above represents the gates of the restricted traffic zone (RTZ) present in Florence, has as DataProperties *dct:identifier* and *dct:description*, inherited from DublinCore Ontology, the passage type dataProperty namely *km4c:typeOfGate* and the DataProperty for localization *geo:lat* and *geo:long*.

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

The *Context* class has a large number of information retrieved from tables that describe the individual ETL transformations, which process different public/private data; for each process, in fact, a source type has been defined, stored within the field *dct:source*; the date of ingestion into the ontology, is instead containing into *dct:created* DataProperty; the original data format (CSV, DBF, etc.) is instead stored into *dct:format* and a brief description of the dataset is saved into *dct:description* DataProperty.
Other DataProperties of the *Context* class are the DataProperty *dct:right*, where the dataset license bound is saved; the type of process to which the context refers, is instead stored into the *km4c:processType* DataProperty; the DataProperty *km4c:automaticity* says if the process is completely automated or not: for example, the Street Guide datasets cannot be fully automated because the process to obtaining data, needs a physical person to send and receive emails.
The DataProperty *km4c:accessType* has been also defined for the *Context* which refers to how the data are recovered (HTTP calls, Rest, etc..); the DataProperty *km4c:period* contains the time (in seconds) between two calls of the same process, it is one of the main parameters of the Process Scheduler, together with *km4c:Overtime* DataProperty, that indicates the time after which a process must be killed.
Finally, the DataProperty *km4c:param* contains the resource link, if the resource in question is an OpenData set retrievable via HTTP, *km4c:lastUpdate* represents the date of the last update of the data set, while the DataProperty *km4c:lastTriples* contains the data of the last triple generation.

Chapter 5

5. The Architecture

In this chapter, the description of the data engineering architecture is proposed; the built architecture is parallel and distributed, and it is formed by a Master machine and, currently four Nodes machine but, thanks to Hadoop, which is installed on all the computer, the number of nodes is easily and quickly incrementable. All three machines mentioned above, are equipped with Ubuntu 14.04 and, thanks to Hadoop and its HDFS, and thanks to the Process Scheduler, Nodes are able to perform operations on the unique storage system, located on the Master machine.

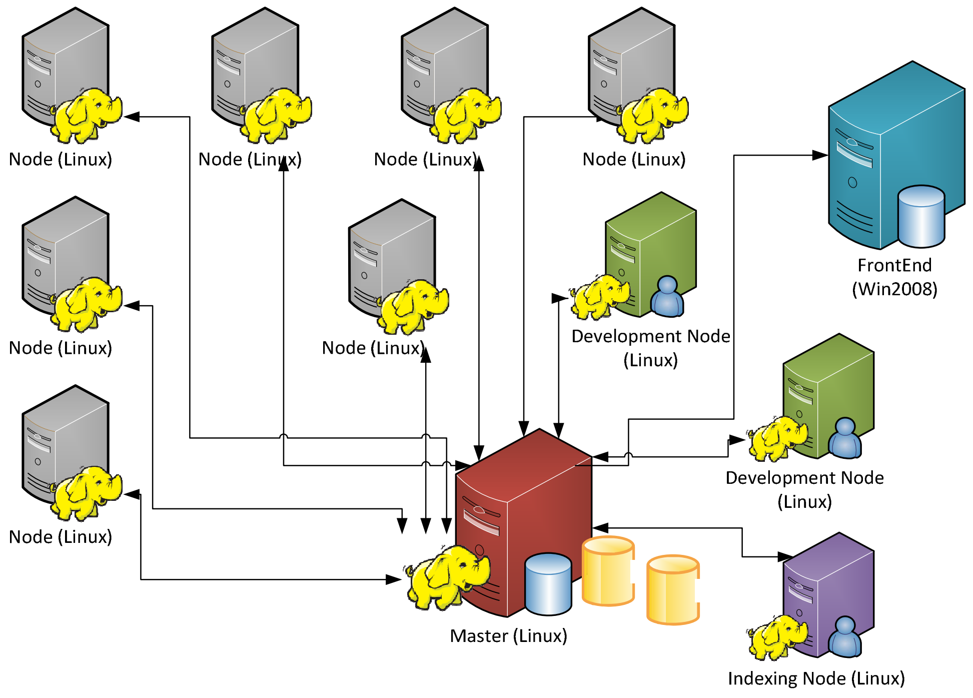


Figure 24 - Schematic representation of implemented architecture

Another machine, shown in Figure 24 as Indexing machine, deals with the indices regeneration and with the creation of new updated versions of the triplestore, which can then be made available to third parties via the Frontend machine, that is instead equipped with Windows Server 2008. Each node has 12GB of RAM, 200GB of storage space and they can harness the power of a Intel Xeon X5690@3.47Gz CPU; the Master machine and the Indexing machine are instead equipped with 12 GB of RAM, 500 GB of storage space and the same CPU.

The management of the distributed occurs thanks to Hadoop, which is a key component in the architecture because it allows the execution of multiple ingestion operations, with no effect on the single storage system located on the Master machine; in fact, the main limitation of a single machine solution, is the RDF tripestore based on OWLIM, which allows to run simultaneously maximum one read operation and one write operation, otherwise the system performance degrades very quickly, until make the machine unusable. Thanks to Hadoop and mutiple nodes, this problem can be solved and the triplestore can still be based on OWLIM.

All processes are managed by a Process Scheduler that allocates them on the parallel and distributed architecture, in which nodes can devote to the ingestion of new data, which will then be stored on the master machine, again thanks to the Process Scheduler that manages the order of all operations.

The Indexing machine takes care of generating a new triplestore containing the updated data ingested by nodes on the master machine, a task that can take several hours; when this operation is finished, the new triplestore, will replace that present on the Frontend machine, by avoiding to degrade the system performance, but especially avoiding to interfere with the availability of the system.

Regarding HBase [Aiyer et al., 2012], this type of NoSQL datastore has been chosen because it is a large-scale distributed database build on top of the HDFS, which deals with the management of the distributed part; furthermore both HBase and HDFS systems have been developed by considering elasticity as fundamental principle, and the use of low cost disks has been one of the main goals of HBase. Therefore, to scale the system results is easy and cheap, even if it has to maintain a certain fault tolerance capability in the individual nodes. In addition, another fundamental reason is that HBase is well integrated with Pentaho Kettle, i.e. the ETL tool used.

Within HBase ingested data is stored before being converted into triples; any updated versions of the static data are managed in such a way as to create a historical archive, in fact, each update data, even if only slightly changed, is saved again on HBase in a new row. This solution requires more storage space, but leaves open the possibility of performing statistical analysis on Static data and their changes over time.

The whole operation of the realized architecture can be regarded as divided into the following seven phases of:

- Data Ingestions
- Quality Improvement

- knowledge Mapping
- knowledge Reconciliation to make the model semantic interoperable
- Indexing regeneration
- Verification and Validation
- Access/exploitation from services.

In Figure 25 is possible to observe a schematic structure of the entire architecture, where each phases is highlighted.

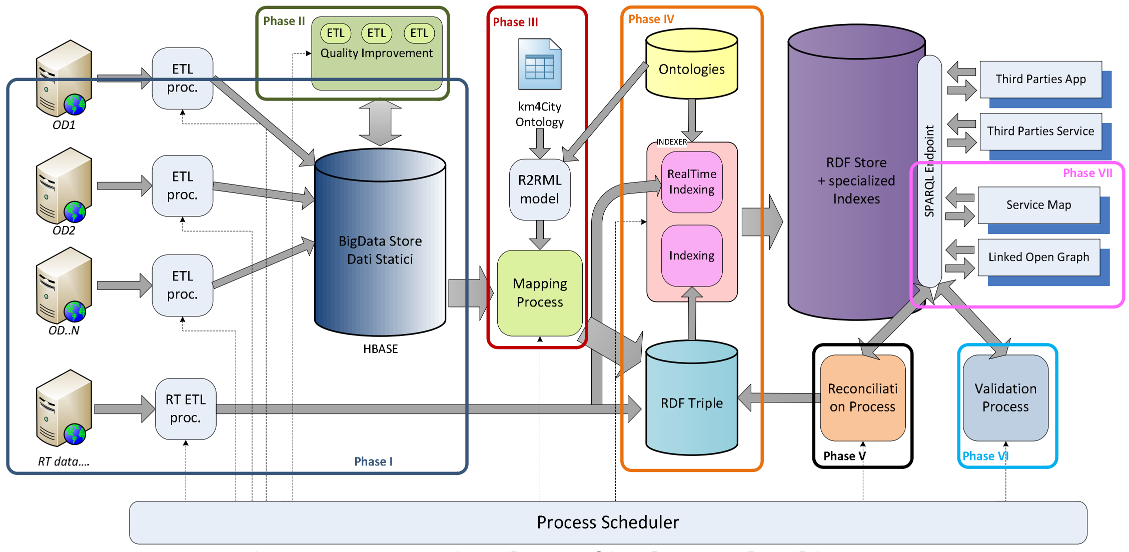


Figure 25 - Processing phases of implemented architecture

The whole phases of the ingestion processes are managed by a Process Scheduler that allocates processes on the parallel and distributed architecture. To allow the regular update of ingested data, the scheduler regularly retrieves data and check for updates. The ingested data are then subjected to a Quality Improvement process trying to correct the errors most frequently found, that is also managed by the Process Scheduler. Following data is transcoded and then mapped in the *Km4City* Ontology. After that, they are made available to applications on an RDF Store (OWLIM-SE) using a SPARQL Endpoint.
The next sections are dedicated to the operation of each phases identified in the architecture realized.

5.1 Phase I: Ingestion

The design of the ingestion process, i.e. the data acquisition phase, represented in the Figure 25 as Phase I, required much effort. The process takes as input, data from different data source, such as the *Osservatorio dei Trasporti* of the Tuscany region, the

Open Data portal of the Tuscany region and other Open Data portals of its main municipalities (Florence, first of all for its large production of Open Dataset) and the MIIC web services.

As seen in Chapter 3, there are both static and real time data to be integrated therefore, it is clear that due to the high heterogeneity of data and sources, the ingestion process will have to be made up of individual branches, that making possible to cover the totality of data type/source combinations, such as just represented in Figure 25.

In most cases, each branch is responsible for periodically download the dataset from the respective source, and, if the dataset had already been ingested, for check if it is updated, and then to ensure its management via the correspondent ETL (Extract, Transform, Load) transformation, which ends with the insertion of gathered data within a NoSQL database, more specifically an *HBase* (http://hbase.apache.org/) datastore.

The management of the periodic update check, is assigned to the Process Scheduler, which allows to set a different updates interval for each resource; for datasets currently managed, the update period varies from a few minutes for Real Time datasets, up to 1 time per month or even more, for static datasets.

For the development of ETL processes, the software Pentaho Kettle [Appendix A.1] is used; each branch, before being connected to the system architecture, is individually verified and validated.

In the next few pages the most significant branches, that lead to the ingestion of at least one different data/source type pair treated, will be analyzed.

5.1.1 *Street Guide Ingestion*

The Street Guide ingestion process is not fully automated, because the *Osservatorio dei Trasporti*, in order to download the entire dataset, requires that a request form is completed, after which the applicant will receive an email containing the link to download the requested resource. So, it was therefore necessary to modify, for this dataset, the standard "*checking for updates*" process used for the other dataset: in fact the Process Scheduler has been programmed to send a remainder to the administrator of this resource, who manually checks for data updates. Fortunately, Street Guide is a *Static* resource, which is really rarely updated (only two update in more than the past two years).

The downloaded dataset consists of 78 files for each province, each of which contains information about one of the main entities that make up the Street Guide (further details are provided in the Section 3.1).

The main job of Street Guide ingestion transformation, is *Wrapper.kjb*, shown in Figure 26.

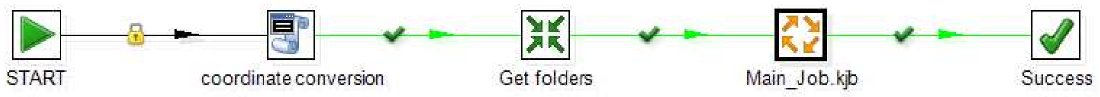
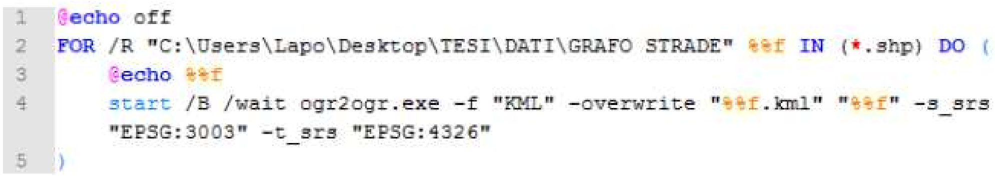


Figure 26 - Wrapper.kjb

This Job initially realizes a format conversion, from ShapeFile to KML, thanks to the script *coordinates conversion*; such conversion is necessary mainly because the *Osservatorio dei Trasporti* uses a particular map projection, different than the one used in all the other geo-referenced file. The geo-coordinates of the points are in fact supplied as Gauss-Boaga [reference], a standard adopted in Italy in past years, while now the standard for geo-referenced data, is the WGS84 projection [reference].
To solve this problem, the QGIS software [Appendix A.2] was used: its *ogr2ogr* command invokes a script that can convert geographic coordinates in different formats through different map projections. In Figure 27 the batch associated with the block *coordinates conversion*, is shows.



```
@echo off
FOR /R "C:\Users\Lapo\Desktop\TESI\DATI\GRAFO STRADE" %%f IN (*.shp) DO (
    @echo %%f
    start /B /wait ogr2ogr.exe -f "KML" -overwrite "%%f.kml" "%%f" -s_srs
    "EPSG:3003" -t_srs "EPSG:4326"
)
```

Figure 27 - Ogr2ogr script

The *for* loop extracts all files with extension *.shp*, contained into the folder, where the data downloaded from the web site of the *Osservatorio dei Trasporti*, are stored. The *ogr2ogr* command is executed on each shape file, using the parameter *-f* to select the file output format, i.e. KML, *-overwrite* to allow overwriting generated files when the script is run again, *"%% f. kml*" imposes that the generated file is saved with same name but extension *.kml*, and finally the parameters *-s_srs* and *-t_srs*, map the unique geodetic reference codes, respectively relating to Gauss-Boaga (also called Monte Mario) and WGS84.
When the conversion of the shape files in KML files is completed, the real processing of Street Guide files, can begin. The downloaded data are originally divided into sub-folders, one for each province of the Tuscany region; the *Get_Folders.ktr* transformation pulls all the folders names contained in the root directory, in which data related to the Street Guide, are stored, and provides them as input data for the next block, i.e. the Job *Main_Job.kjb*. This Job will be automatically repeated for each input folder, simply by selecting the "Execute for every input row?" check box, within the Advanced tab in the settings window of the Job.
In Figure 28 is shown the structure of the Job *Main_Job.kjb*.
The data concerning each entity in the graph are all processed in very similar ways: data are taken from the corresponding files, they are then processed by specific Kettle's transformation, and the rows obtained are finally stored in the NoSQL datastore built with HBase.

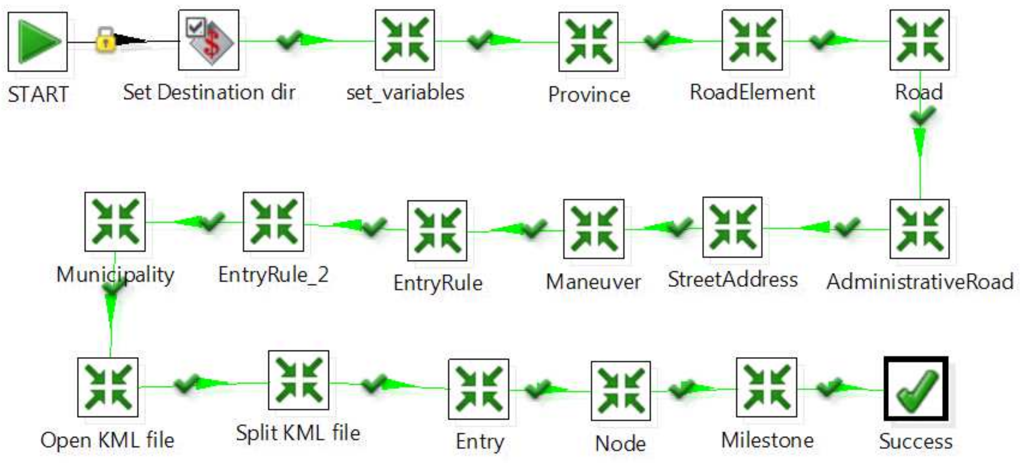


Figure 28 - Street Guide main_job.kjb

The first Job step, reads the name of the processing folder from the native resultset of Kettle and stores it in a variable named *FOLDER*; Pentaho Kettle allows to store information such as variables and later to reuse it, thanks to the special character *$*.
The next transformation of the main Job deals with general data relating to provinces, as it is shown in the following Figure 29.

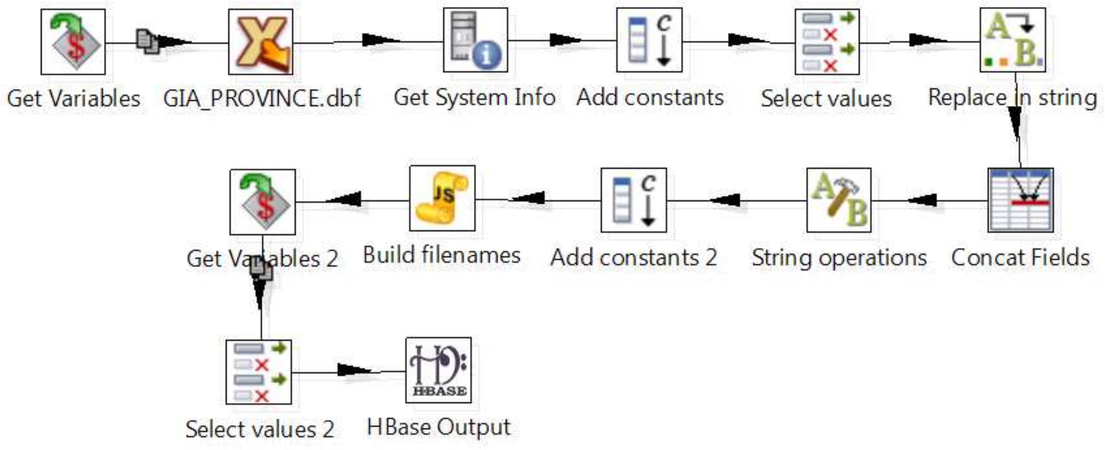


Figure 29 - Ingestion transformation for provinces

Initially data is read from the file *GIA_PROVINCE.dbf*, while the next steps have the task of picking the current date and convert it to the *xsd:dateTime* format, as suggested by the W3C.
In detail, current date is fetched from operating system, then a field containing the timezone used in Italy, i.e. "+1: 00" is created; the string initially taken is transformed to the desired format, thanks also to the addition of the *timezone* to the final string.

Finally, before saving data to HBase, the data source value, that is the string *Osservatorio dei Trasporti*, is properly stored in a separate column.
The next transformation, i.e. *Road_Element.ktr*, that processes the data relating to road elements and its structure, is shown in Figure 30.

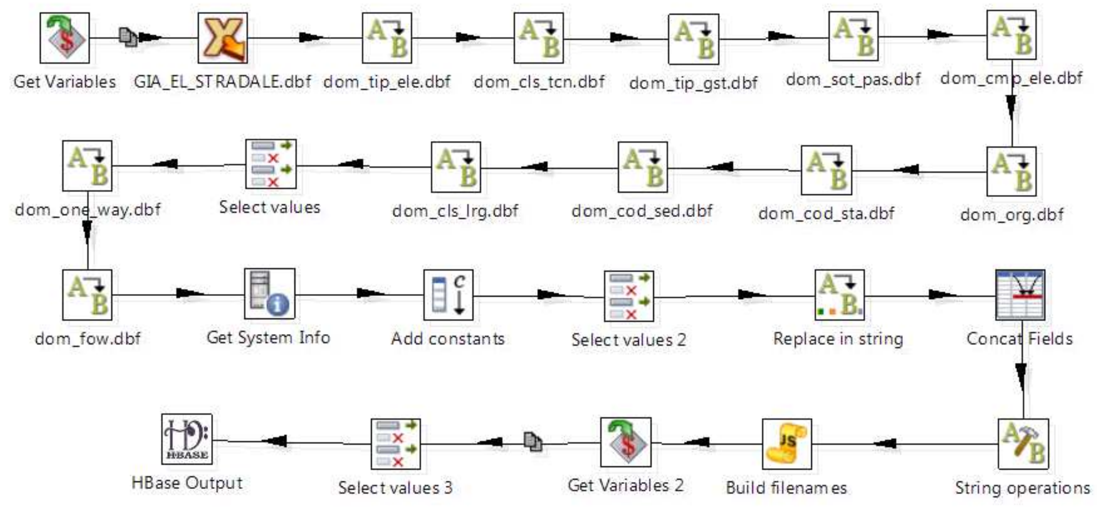
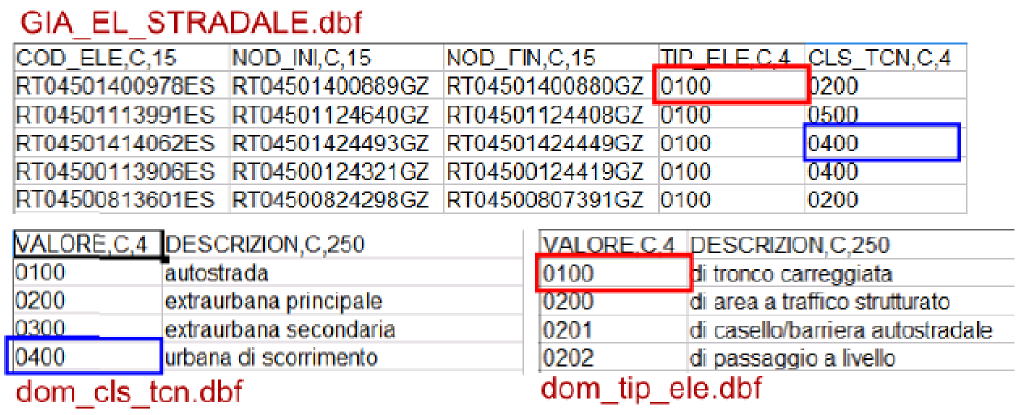


Figure 30 - Ingestion transformation for Road Elements

The first operation performed, is responsible for reading the data contained inside the *GIA_EL_STRADALE.dbf* file, earlier downloaded.
After that, a series of string replacement operations are performed to map with their extended meaning, each code used by the Tuscany region, to represent some features, which lie precisely into some domain tables. To better clarify the described procedure, see Figure 31.


GIA_EL_STRADALE.dbf

COD_ELE,C,15	NOD_INI,C,15	NOD_FIN,C,15	TIP_ELE,C,4	CLS_TCN,C,4
RT04501400978ES	RT04501400889GZ	RT04501400880GZ	0100	0200
RT04501113991ES	RT04501124640GZ	RT04501124408GZ	0100	0500
RT04501414062ES	RT04501424493GZ	RT04501424449GZ	0100	0400
RT04500113906ES	RT04500124321GZ	RT04500124419GZ	0100	0400
RT04500813601ES	RT04500824298GZ	RT04500807391GZ	0100	0200

VALORE,C,4	DESCRIZION,C,250
0100	autostrada
0200	extraurbana principale
0300	extraurbana secondaria
0400	urbana di scorrimento

dom_cls_tcn.dbf

VALORE,C,4	DESCRIZION,C,250
0100	di tronco carreggiata
0200	di area a traffico strutturato
0201	di casello/barriera autostradale
0202	di passaggio a livello

dom_tip_ele.dbf

Figure 31 - Mapping attributes

The subsequent operations, as in the previous transformation, leading to the creation of the current date in the suggested format by the W3C. So, the last step of this transformation, can insert obtained data into HBase.
The next transformation processes data related to roads, that is *Road.ktr*. Due to relationships that link roads, road elements and administrative road (see Section 4.4.2), this transformation is more complex.

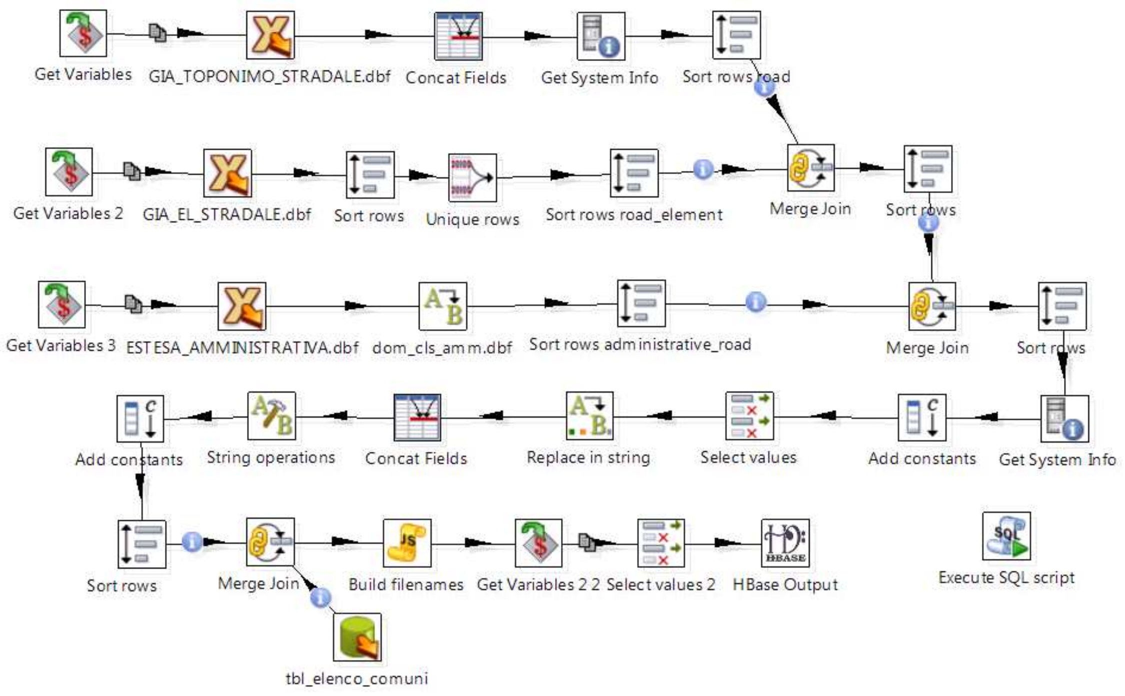


Figure 32 - Ingestion transformation for Toponyms

As shown in Figure 32, data contained in files *GIA_TOPONIMO_STRADALE.dbf*, *GIA_EL_STRADALE.dbf* and *ESTESA_AMMINISTRATIVA.dbf* are read, together with *tbl_elenco_comuni* contents, a support MySQL table containing information regarding all Tuscany municipality, their postal code and their ISTAT code.
Inside the toponyms table, derived from the first file read, a new field, named *EXT_NAME*, is created, concatenating *DUG* and *NAME* of each toponym; then the current date is retrieved and stored into *update_date* field and finally, the table rows are ordered by toponym code value.
To the table obtained from the second file, i.e. *GIA_EL_STRADALE.dbf*, a sort order based on to two keys, is initially applied: the first key is the toponym code, to which the element belongs, and the second key, is the administrative road code. The rows containing a unique pair of the two keys, are then selected, and ordered by the toponym code (which is also contained in the table of road elements); thanks to a *Merge Join* step, the two modified tables, are merged.

The third table, containing the same data of *ESTESA_AMMINISTRATIVA.dbf* file, is first ordered by *COD_REG* key, and then the administrative class codes are replaced by the corresponding descriptions, contained in the domain table *dom_cls_amm.dbf*. A new *Merge join* step is then applied to the table just processed, and to the table obtained with the previous merge operation: the result is a single table containing all data needed to define the relationships between toponyms, road elements and administrative road.
Finally, the *update_date* field is formatted, as in the previous transformations, in the W3C desired format. Before saving the processed data on HBase, a further step of *Merge join* is performed, to transform the municipality identifiers to which the road belongs to, with the corresponding ISTAT (reference) code.
The next analyzed transformation is *Administrative_Road.ktr*, shown in the following figure.

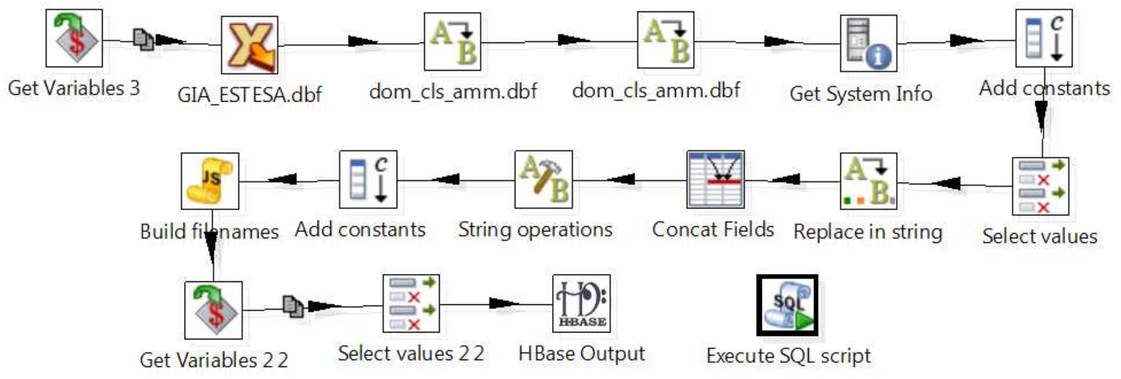


Figure 33 - Ingestion transformation for Administrative Roads

Similarly to the first two transformations described, this one first loads data from the source file *ESTESA_AMMINISTRATIVA.dbf*, then a domain tables mapping is applied to replace the feature codes with their descriptions, and, after that, the current date value and data source value are created creating, in the correct format, and finally the transformation saves all data into HBase.
Also the *Street_Number.ktr* transformation, which deals with data relating to street numbers, has the same structure of the previous transformation, therefore, it will only be reported in Figure 34.
The transformation *Maneuver.ktr*, however, differs from the previous ones, due to the semantic of the data relating to maneuvers: in fact, data released by *Osservatorio dei Trasporti*, are contained in two main files, *GIA_MANOVRE.dbf* and *GIA_MANOVRE_ELSTR.dbf* containing the individual maneuvers permitted or forbidden and their type, in the first file, and the list of road elements involved in each maneuver, in the second file.

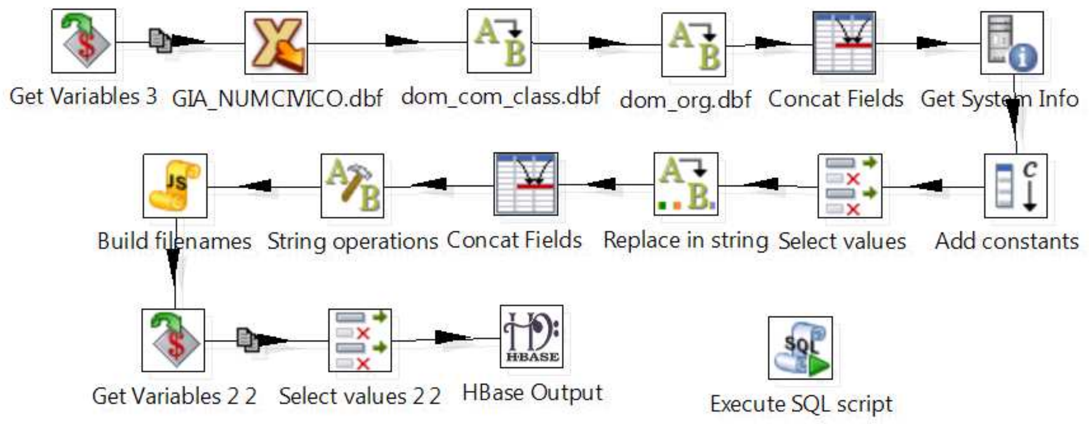


Figure 34 - Ingestion transformation for House numbers

In order to map data into the ontology described in Section 4.4, information contained in each files, must be joined together; for this reason initially these two files are reading, and then the transformation sorts the data contained in the file *GIA_MANOVRE.dbf* file, according to maneuvers identifier; as in previous transformations, codes contained into the files, are mapped respectively with the correspondent description, contained in the domain tables.

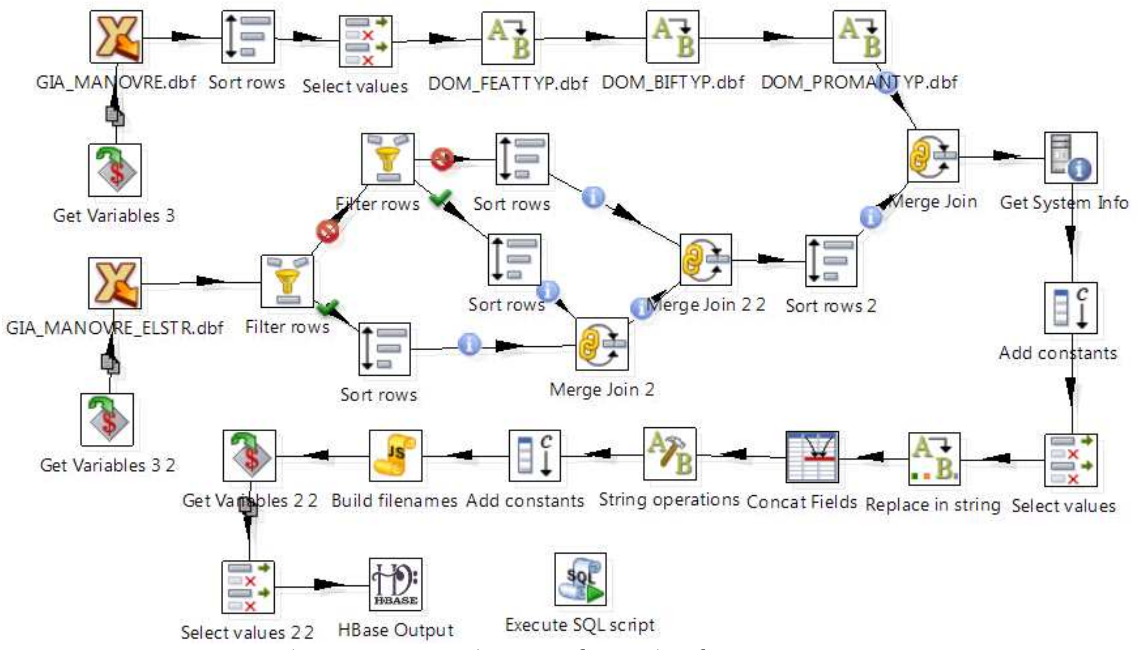


Figure 35 - Ingestion transformation for Maneuvers

The second file requires a deeper processing: in fact more rows in the table refer to a same maneuver and, according to what seen in Section 4.4.2, a maximum of three road

elements can be connected to the same maneuver; so data are divided according to the sequential number of elements involve. The three obtained flows, respectively relating to road elements with sequential numbering equal to 1, 2 and 3, are then merged into a single table, and then they are sorted according to the maneuver identifier.
Data thus obtained, is merged again with data coming from the first transformation branch. As in previous transformations, information about the current date and the data source, are also created, before the final HBase step, that save data.
Transformation in Figure 36 processes data on entry rules, i.e. restrictions or permissions access to defined road elements or maneuvers; the files, that collect this type of information, are *GIA_REGOLA_ACCESSO.dbf*, *GIA_ACCESSO_ELSTRADALE.dbf* and *GIA_ACCESSO_MANOVRE.dbf*.

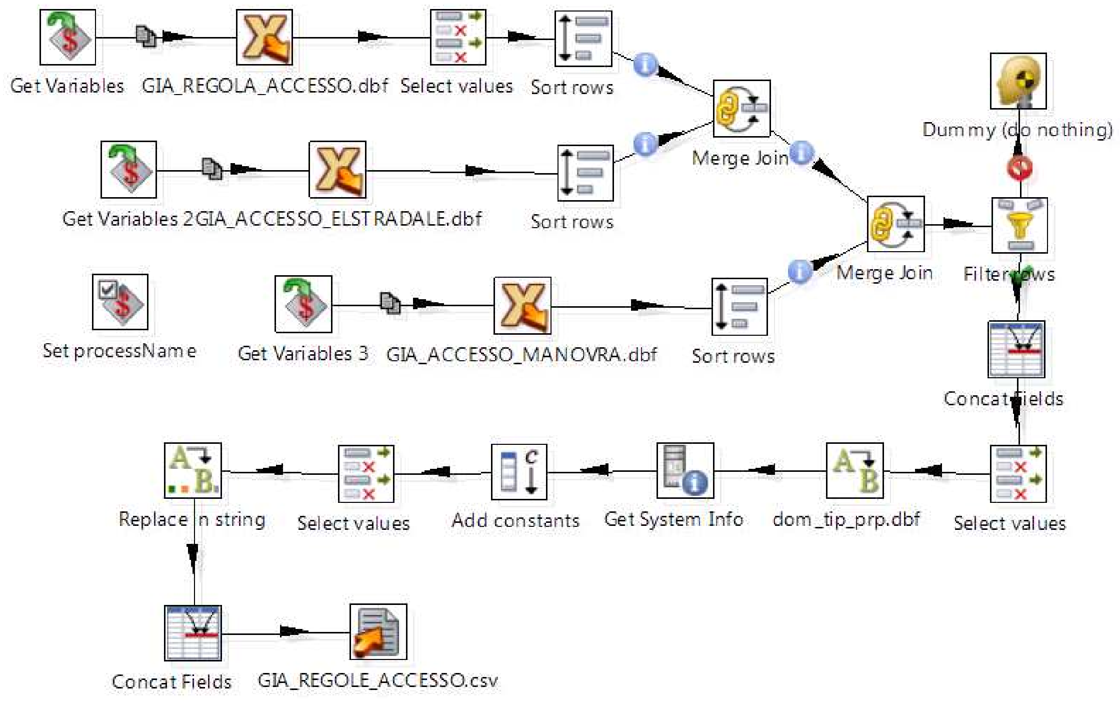


Figure 36 - Ingestion transformation for Entry rules

To facilitate the next phase of data mapping on the knowledge model, data contained in these three files, are together; after that, incomplete rows are removed together with rows for which no correspondence between the entry rules indices, has been identified.
Based on the value of two attributes, *RSTTYP* and *RESTRVAL*, the entry rule type is defined; the latter attribute is a numeric code that can take the same value for different values of the first attribute, that is *RSTTYP*. To simplify this situation, the two attribute values are unified in a single field, which is then converted to string. Finally, *RSTTYP* values are mapped with their descriptions, taken from the correspondent domain table.

Also in this transformation, the current date in the *xsd:dateTime* format is created and finally, all obtained information is stored in a temporary CSV file, which is the input of the next transformation, that is *Entry_Rule2.ktr*.
Such transformation is necessary to avoid a small Kettle bug, which does not allow to apply the mapping operation more than once for transformation, and consists of steps represented in Figure 37.

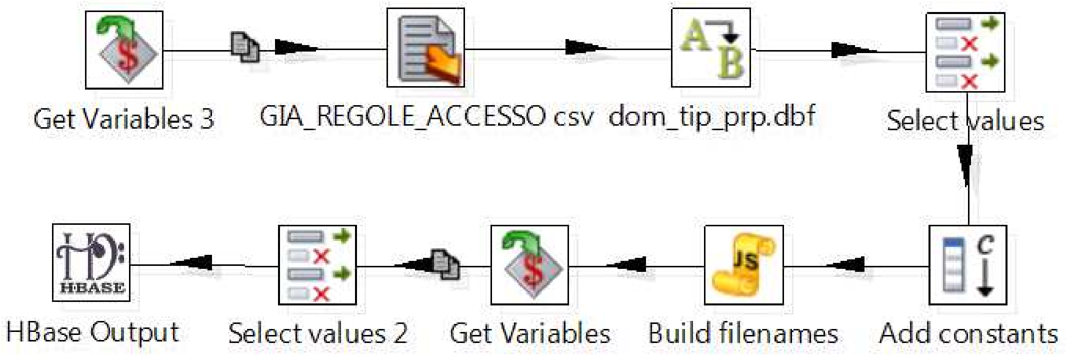


Figure 37 - How to correct a small Kettle bug

This transformation simply performs *RESTRVAL* values mapping (not allowed otherwise) and also defines field that contains data source information; finally, data is saved on HBase.
The next transformation is the *Municipality.ktr*, whose structure is shown in Figure 38.

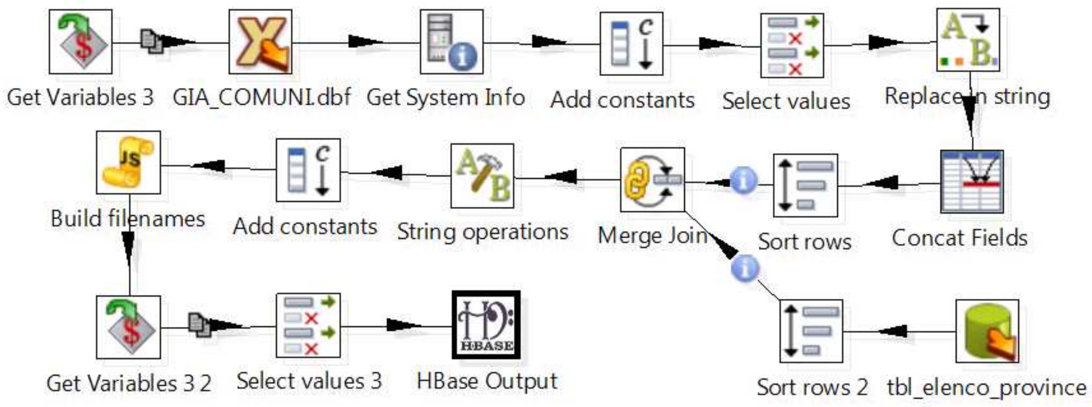
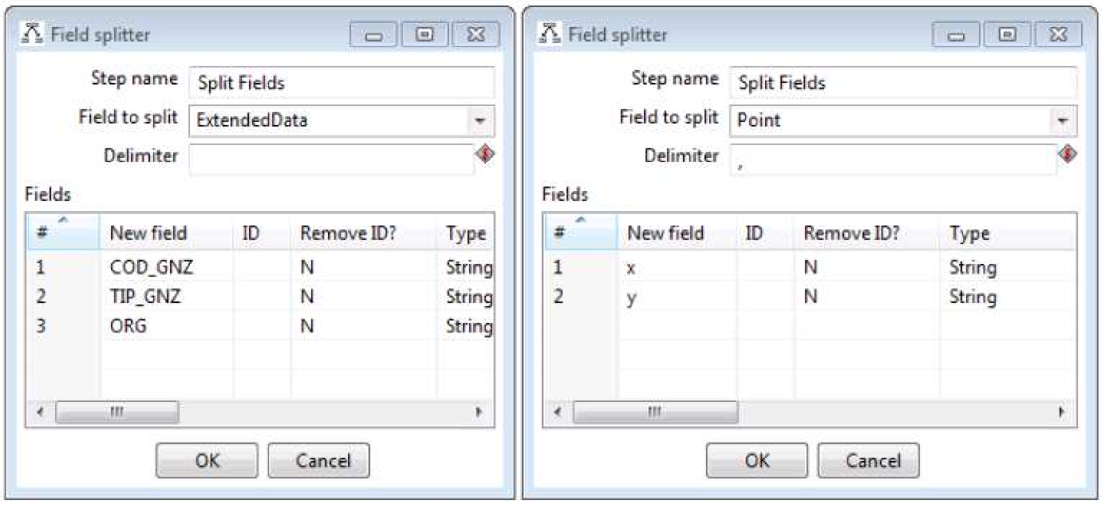


Figure 38 - Ingestion transformation for Municipalities

As is easy to observe, this transformation is very similar to transformation concerning provinces, previous described.
First of all, data is reading from *GIA_COMUNE.dbf* file, while the next steps create the current date field and convert it to the *xsd:dateTime* format, as suggested by the W3C.

Thanks to a MySQL support table, containing ISTAT codes of all Tuscany provinces, called *tbl_elenco_province*, a new column is created, containing just the province code to which each municipality belongs.
Finally, data is saved to HBase, after the data source information has been saved into a new column.
The next five transformations of the main job, pull data from a different file type, that is KML files, converted from the downloaded Shape files. Unfortunately, the KML files size is 5-6 times greater than the Shape files size, a factor that slows down the ingestion process; fortunately, the processed data are static and therefore, these transformations are performed infrequently, so, despite the increase of the execution time ,there is a small effect on the final architecture performance.
The *Keyhole Markup Language* is directly derived from XML, i.e., *eXtensible Markup Language*, a language more difficult to manage through *Pentaho Kettle*, whose functionalities are better suited to data in tabular form. For this reason, the next step, that is *Open_KML_File.ktr*, is responsible for the transcription of the data contained into the files *GIA_ACCESSO.kml*, *GIA_GIUNIONE.kml* and *GIA_CIPPO.kml* in *CSV* format.
The subsequent transformation of the main job, uses the created CSV file as input, within which there are two columns, *ExtendedData* containing all founded attributes, and *Point*, containing all founded geo-referenced information. Data contained in the two columns, are then spread across multiple columns, as shown in Figure 39.



Step name: Split Fields
Field to split: ExtendedData
Delimiter:

Fields

#	New field	ID	Remove ID?	Type
1	COD_GNZ		N	String
2	TIP_GNZ		N	String
3	ORG		N	String

Step name: Split Fields
Field to split: Point
Delimiter: ,

Fields

#	New field	ID	Remove ID?	Type
1	x		N	String
2	y		N	String

Figure 39 - Splitting fields

Again because the *Kettle* bug concerning data mapping, previously seen, some temporary CSV files are created, with names *GIA_GIUNZIONE2.csv*, *GIA_ACCESSO2.csv* and *GIA_CIPPO2.csv*.

The next three transformations of the main job, take care of each one of the CSV files and, after importing data, the attributes mapping with values contained in the respective domain tables, is performing, together with the generation of the current date in W3C recommended format and the creation of the data source field (that containing the value *Osservatorio dei Trasporti*). Finally, each transformations save data into HBase.
Given that the three transformations are very similar, in Figure 40 only that relating to the *Entry* class is reported.

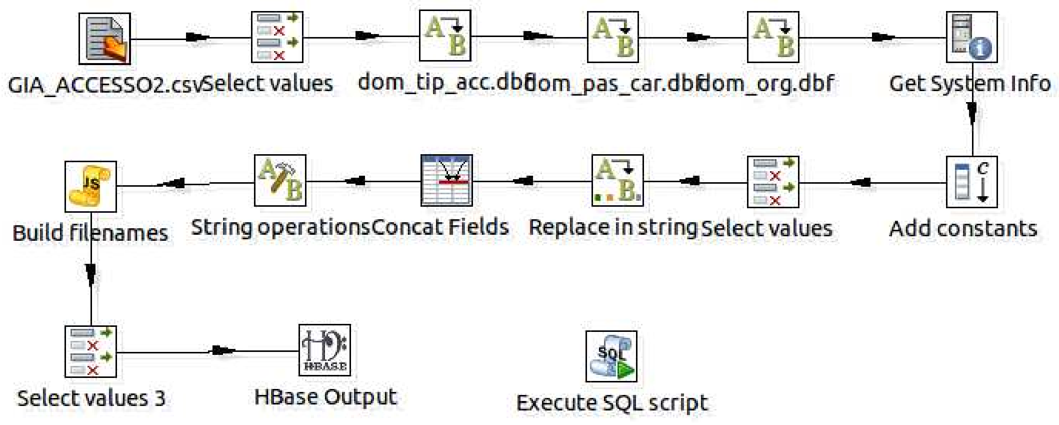


Figure 40 - Ingestion transformation for Entries

The next three transformations of the main job, are used to ingest data on road element composition (see Section 4.4.2). The first transformation, called *Open_Coordinates.ktr*, transforms the KML file in CSV file, thus creating the file *GIA_EL_STRADALE.csv*; the second transformation, being more complex than the previous one, is then shown in Figure 41.

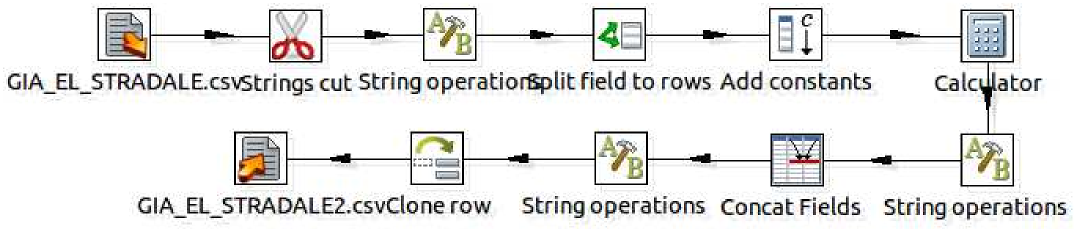
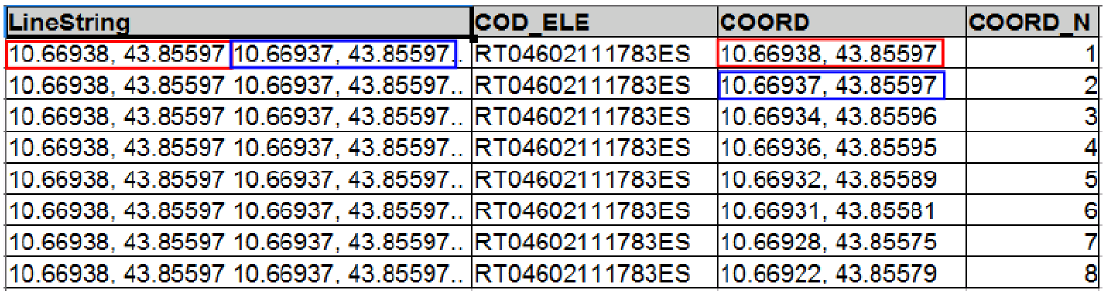


Figure 41 - RoadLInk transformation for Road Elements

Process_Coordinates.ktr takes as input the CSV file created by *Open_Coordinates.ktr* transformation and, extracts the fields *ExtendedData* and *LineStrings* (present in place of *Point* field seen previously, since in this case a series of coordinates are processed and not only a couple) together with the road elements identified. After this operation, *ExtendedData* is divided into multiple columns; the *LineStrings* field, instead, contains a series of pairs of coordinates that correspond to all junctions composing each individual road element.

So, for each pair of coordinates contained in *LineStrings* field, a new row in the table is created, which also contains a new column with the order number of each coordinate. In Table 3 is possible to observe an example of this processing.



LineString	COD_ELE	COORD	COORD_N
10.66938, 43.85597 10.66937, 43.85597.	RT04602111783ES	10.66938, 43.85597	1
10.66938, 43.85597 10.66937, 43.85597..	RT04602111783ES	10.66937, 43.85597	2
10.66938, 43.85597 10.66937, 43.85597..	RT04602111783ES	10.66934, 43.85596	3
10.66938, 43.85597 10.66937, 43.85597..	RT04602111783ES	10.66936, 43.85595	4
10.66938, 43.85597 10.66937, 43.85597..	RT04602111783ES	10.66932, 43.85589	5
10.66938, 43.85597 10.66937, 43.85597..	RT04602111783ES	10.66931, 43.85581	6
10.66938, 43.85597 10.66937, 43.85597..	RT04602111783ES	10.66928, 43.85575	7
10.66938, 43.85597 10.66937, 43.85597..	RT04602111783ES	10.66922, 43.85579	8

Table 3 - Coordinates transposition

An additional column in the data table, which contains the sequence number of the coordinate minus one; this procedure has the aim to simplify the creation of the correct identifier for *Junctions* of each *RoadLink*, for which, instead, the identifier is created concatenating the road element identifier and the identifiers of the coordinates pair that delimit it.
The created data are then duplicated and finally saved in a temporary CSV file called *GIA_EL_STRADALE2.csv*.

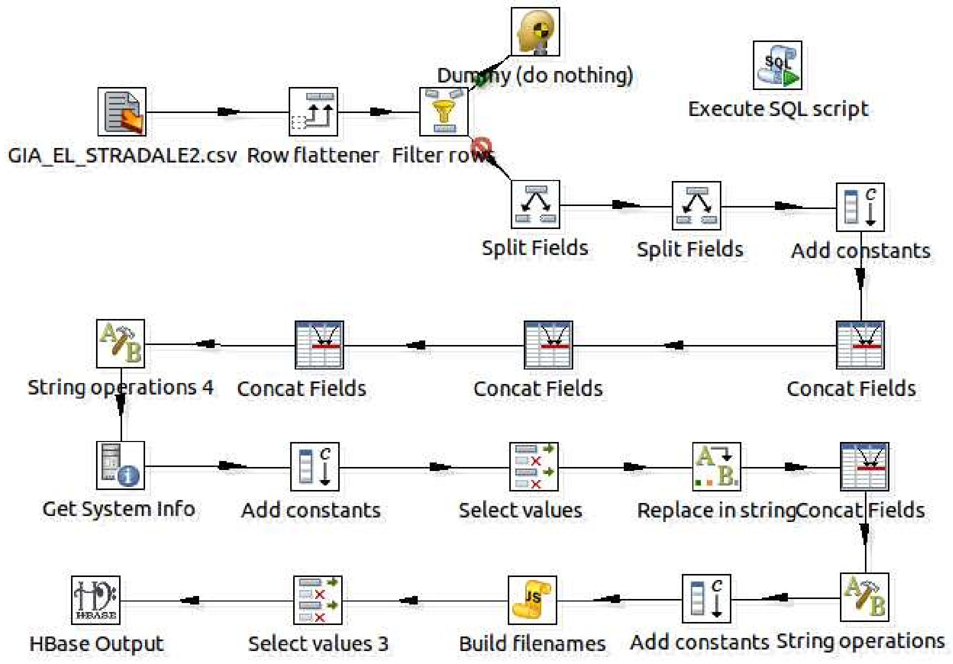
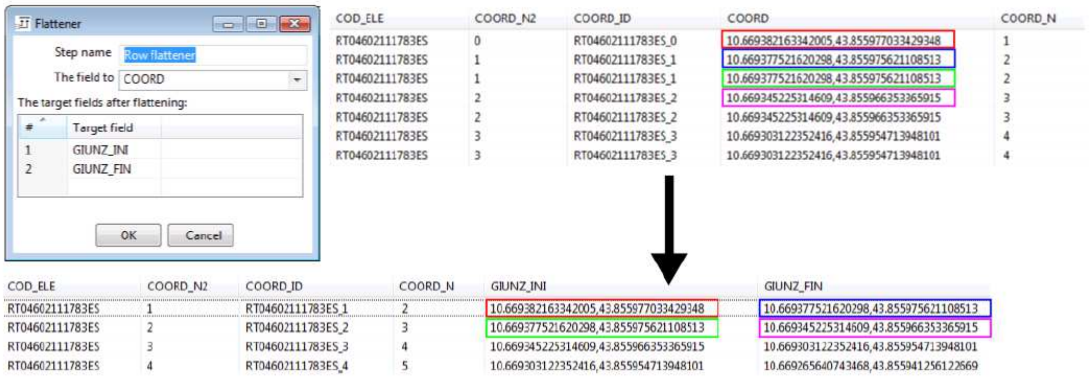


Figure 42 - Ingestion transformation for RouteLinks

The last of three transformations that deal with *RoadLink*, is *Save_Coordinates.ktr* (Figure 42) that mainly performs an operation of *row flattener*: this operation sequentially fetches data from a column and transposes them in other columns, specified by the user; Figure 43 tries to explain the applied procedure.



Step name	Row flattener
The field to	COORD

The target fields after flattening:

#	Target field
1	GIUNZ_INI
2	GIUNZ_FIN

COD_ELE	COORD_N2	COORD_ID	COORD	COORD_N
RT04602111783ES	0	RT04602111783ES_0	10.669382163342005,43.855977033429348	1
RT04602111783ES	1	RT04602111783ES_1	10.669377521620298,43.855975621108513	2
RT04602111783ES	1	RT04602111783ES_1	10.669377521620298,43.855975621108513	2
RT04602111783ES	2	RT04602111783ES_2	10.669345225314609,43.855966353365915	3
RT04602111783ES	2	RT04602111783ES_2	10.669345225314609,43.855966353365915	3
RT04602111783ES	3	RT04602111783ES_3	10.669303122352416,43.855954713948101	4
RT04602111783ES	3	RT04602111783ES_3	10.669303122352416,43.855954713948101	4

COD_ELE	COORD_N2	COORD_ID	COORD_N	GIUNZ_INI	GIUNZ_FIN
RT04602111783ES	1	RT04602111783ES_1	2	10.669382163342005,43.855977033429348	10.669377521620298,43.855975621108513
RT04602111783ES	2	RT04602111783ES_2	3	10.669377521620298,43.855975621108513	10.669345225314609,43.855966353365915
RT04602111783ES	3	RT04602111783ES_3	4	10.669345225314609,43.855966353365915	10.669303122352416,43.855954713948101
RT04602111783ES	4	RT04602111783ES_4	5	10.669303122352416,43.855954713948101	10.669265640743468,43.855941256122669

Figure 43 - RowFlattener explenation

Thanks to the *Clone rows* operation previously applied, this transposition can be done: in absence of doubled values, it would not be possible to use two times the same coordinate to represent the ending junction of a RoadLink and the starting junction of the next RoadLink.

Finally, the transformation generates the date field in *xsd:dateTime* format, and the field related to the data source.

The last step of the main Job, delete all temporary files that were generated and used within the Job and all transformations it recalled.

The ingestion process of the Railway Graph is very similar to that just seen, so for this reasons, it will not be deeply described in this thesis.

5.1.2 *Weather Forecasts Ingestion*

The transformation responsible for the ingestion of data provided by *Lamma*, through the *Open Data portal* of the Tuscany Region (http://dati.toscana.it) related to weather prediction, consists of a main job, a kind of skeleton, which allows to define the order in which each individual steps must be performed.

In Figure 44 the structure of the main Job, named *Previsioni_main.kjb*, can be observe.

The first transformation that makes up the Job, that is *Database.ktr*, extracts all the information needed to complete the operations, from the *ProcessManager* table.

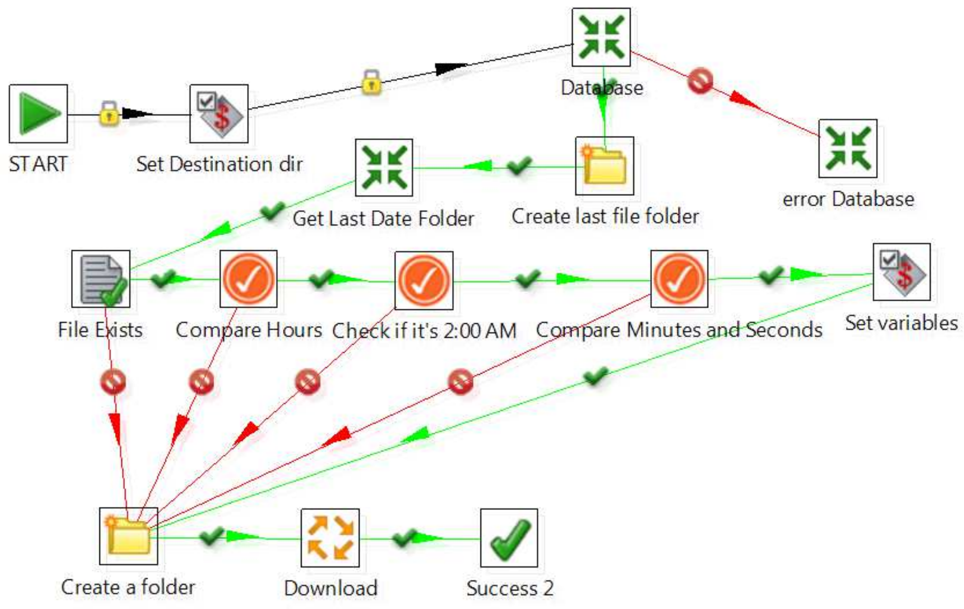


Figure 44 - Ingestion transformation for Weather Forecasts

The *processManager* table, is a MySQL table used mainly by the *Process Scheduler*, which can recover from this table all information concerning the processes associated with each single dataset, to be treated. As is possible to see from Figure 45, the execution of the process steps that must be performed for each dataset, can be activate directly from the *processManager* table, simply press the buttons indicated in the figure by letters *A*, *B* and *C*, and which respectively correspond to phases of Ingestion, Quality Improvment and Triple Mapping of the dataset referred by the row.
In this table the main information about each dataset is collected, and it is used by at least one of the seven phases that comprise the functioning of the architecture presented in this thesis. Such information includes, for example, the path of the processes that the *Process Scheduler* will use to start each single stage of Ingestion, Quality Improvment, Mapping or Indexing, but also information such as the number of RDF triples in which each dataset has been transformed and the number of RDF triples that were actually loaded into the repository, that are useful values for the dataset validation process.
This table also collected the final state and any errors resulting from the execution of each processing phase, in addition to the last execution date and the process scheduling time (i.e. how often the job should run again, a field especially used with Real Time datasets).
So each process interacts with the *processManager* table by writing or only by taking the required values to perform its tasks.
Getting back to the Ingestion process, the next steps, retrieve the creation date of the last downloaded file, perform a check to see if the file already exists, and they also check hour, minute and second to verify if they correspond to the last download file. Just in

case the file is actually the most recent available, it will be downloaded, as is easy to understand, thanks to Job *Download.kjb*, which also ensure the resource storage and processing. This job takes the *processName* variable as input, containing the name of the running process, a value recoverable inside the *ProcessManager* table.

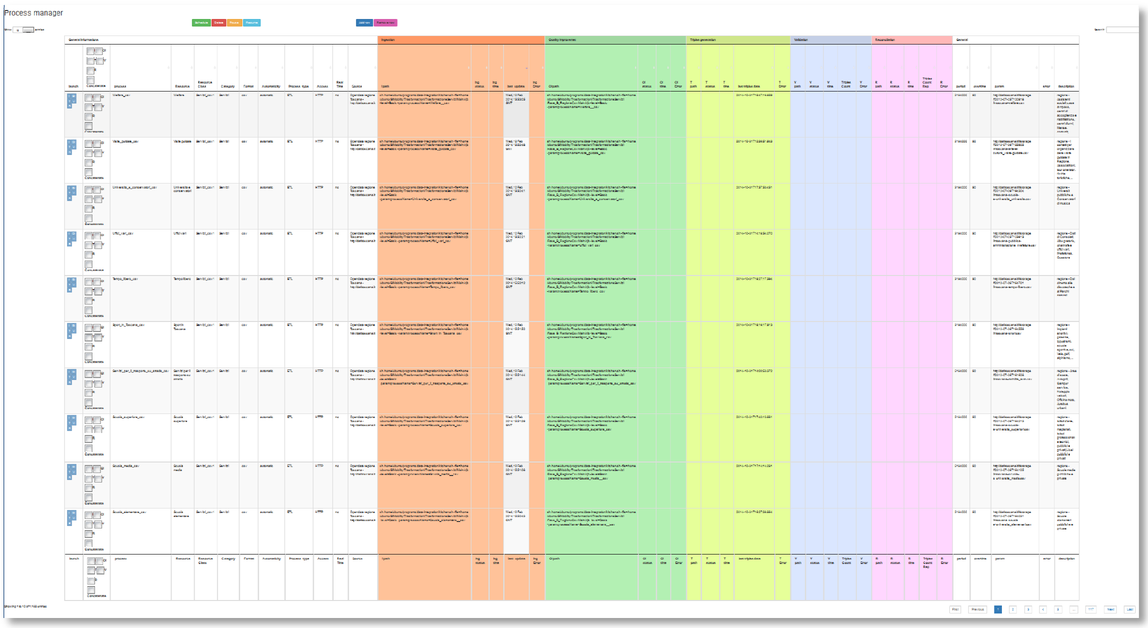


Figure 45 - The ProcessManager MySQL table

In the following pages, transformations and jobs, that make up the main job, will be more deeply analyzed.

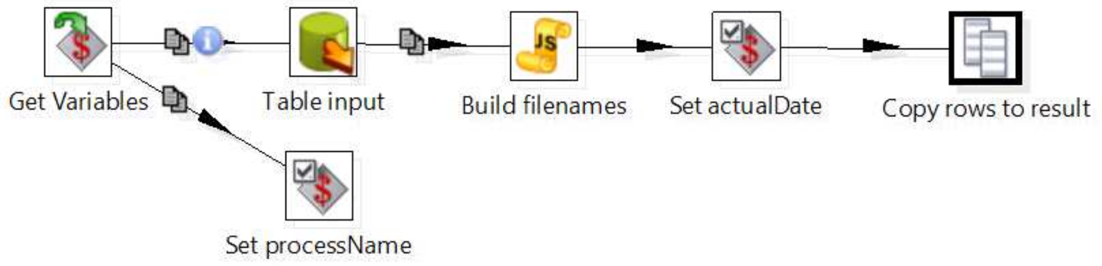


Figure 46 - Database.ktr transformation

Figure 46 shows the *Database.ktr* transformation, which initially retrieves the value of the required variables and then turns it into an usable format for next steps; this operation is necessary because not all steps are able to read parameters, but they might need such values, so they must be stored within well-defined fields. Afterward a query on the *processMananger* table, contained into the MySQL support database, is performed to retrieve rows in which the *process* field corresponds to *processName*.

Furthermore a number of fields need to be created: for example, the *target_file* field containing the path of the file that will be downloaded, *date*, a field in ISO format, that contains the execution date and time, the field *error*, that will contain any error message if some errors will occurs.
Finally, the last step, copies the previously generated lines in a Kettle internal table calling *result*, which is useful for passing information between transformations/jobs.
The *download.kjb* job, as seen above, deals with data retrieval, and it is shown in Figure 47.

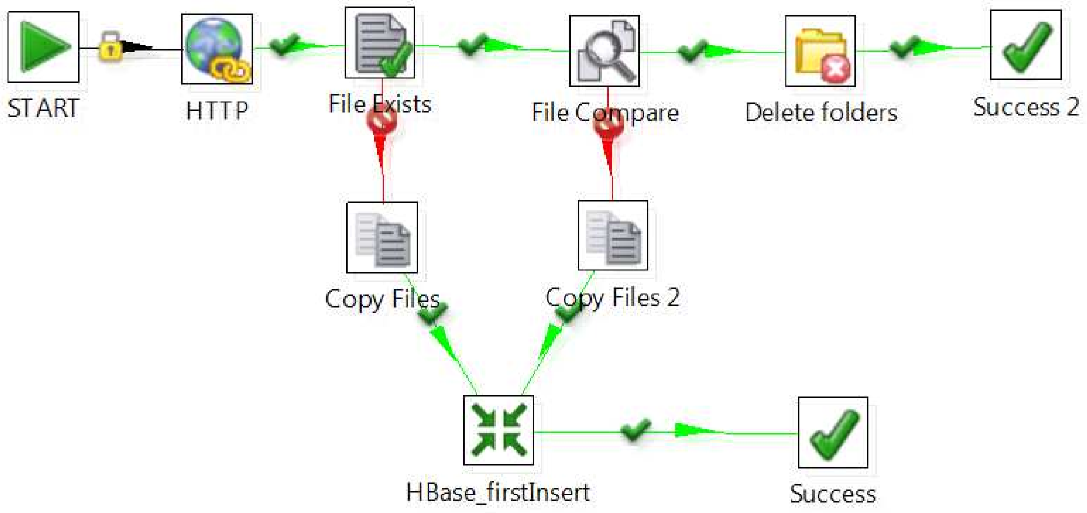


Figure 47 - The Download.kjb Job

Thanks to the *result* table, this job is able to use all information retrieved in the previous processing; initially an HTTP calls GET method [RFC2616] is implemented by specifying parameters as the download URL of the resource, the path where it must be saved and the date of the last download, useful for setting an header type like "IF-modified-since" inside the HTTP call. The specification of this header type allows to download the dataset only if it has been updated since the specified date. If the resource has been updated, it will be downloaded and the last update field will be updated with a new date.
The next step is the downloaded data preparation, for their HBase insertion, performed by the transformation shown in Figure 48.

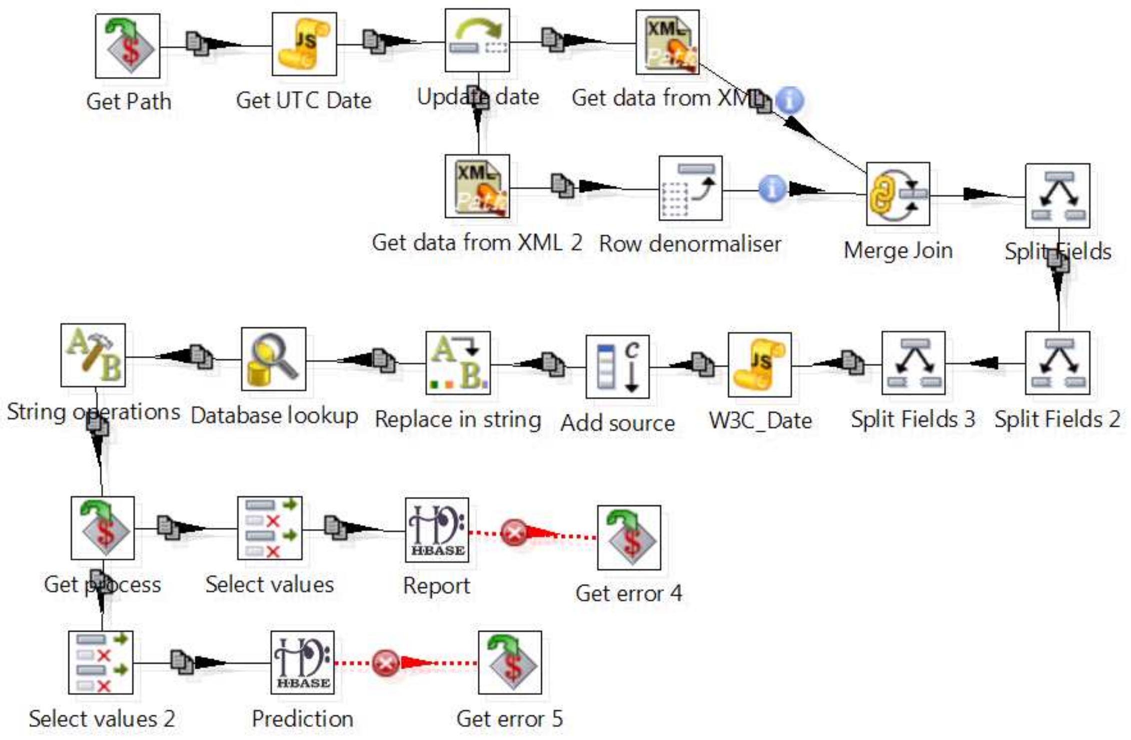


Figure 48 - Ingestion transformation for Weather Forecasts

This transformation initially retrieves the current date can then be saved into *ProcessManager* table, as *last_update* value.
Information contained into the downloaded file, is then extracted by using XPath [XPATH] expressions: this language, as previously seen, allows to see an XML file as a tree graph whose nodes, i.e. the values, are accessible through a path.
In order to clarify the concept, as it is possible to see from Figure 49, wanting to perform a loop on the content of *previsione* tag, in turn contained into the *dati* tag, it is sufficient to set the path as *"/dati/previsione "*.

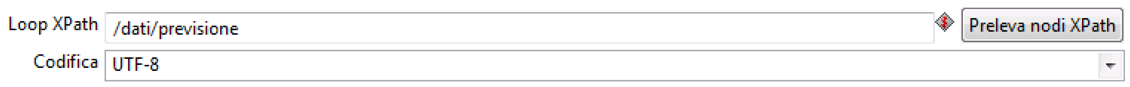
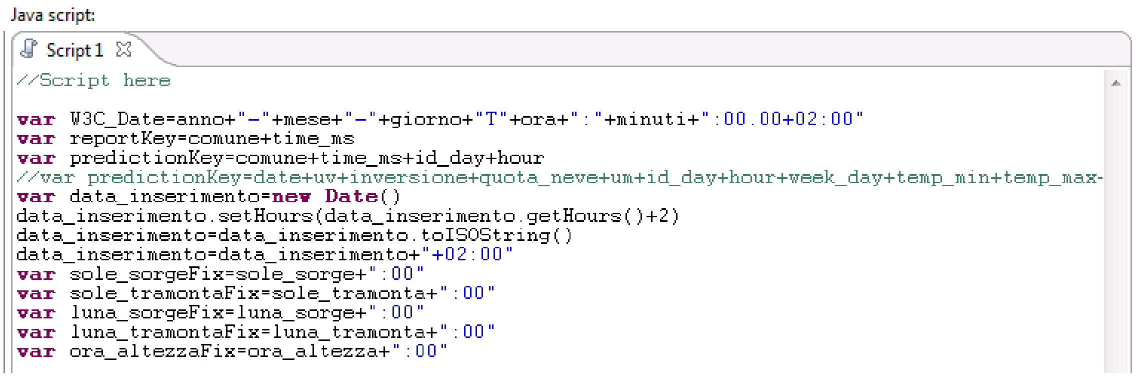

Loop XPath /dati/previsione
Codifica UTF-8

Figure 49 - XPath Loop

The just set loop, is necessary to retrieve information from the 16 different forecasts, contained in the file provided by the *Lamma* consortium, as seen previously in Section 3.4.2.
Moreover, to extract not repeated information, such as almanac information, the *"@"* clause is used, which allows to directly extract the contents of a specific XML tag. Any information relating to wind and cloudiness, are located at a more lower level than those seen so far: thanks to a loop on path *"/previsione/simbolo"*, these features are recovered.

With a dedicated step, the data retrieved from the XML file are transposed from rows to columns, and then all join on a unique row.
After this, the data contained in *aggiornamento*, is spitted into five different fields containing year, month, day, hour and minute values, useful for the construction of a new fields date, in *xsd:dateTime* format and the HBase keys is also created, thanks to the Java script, shown in Figure 50.



```
//Script here

var W3C_Date=anno+"-"+mese+"-"+giorno+"T"+ora+":"+minuti+":00.00+02:00"
var reportKey=comune+time_ms
var predictionKey=comune+time_ms+id_day+hour
//var predictionKey=date+uv+inversione+quota_neve+um+id_day+hour+week_day+temp_min+temp_max-
var data_inserimento=new Date()
data_inserimento.setHours(data_inserimento.getHours()+2)
data_inserimento=data_inserimento.toISOString()
data_inserimento=data_inserimento+"+02:00"
var sole_sorgeFix=sole_sorge+":00"
var sole_tramontaFix=sole_tramonta+":00"
var luna_sorgeFix=luna_sorge+":00"
var luna_tramontaFix=luna_tramonta+":00"
var ora_altezzaFix=ora_altezza+":00"
```

Figure 50 - JavaScript to split *aggiornamento*

The keys for the two forecasts classes, *WeatherPrediction* and *WeatherReport*, were formed through the concatenation of common fields like *Time_ms*, *id_day* and *hour*, for the first class key called *predictionKey*, and *comune* (containing the Municipality name to which the forecast is referred) and *timestamp*, for the second class key, called *reportKey*.
Thanks to *setHours()* function, the timezone is taken into account for calculating the update date; this feature also comes in handy to add the missing *second* parameter to some fields like *sunRise*, *sunSet*, *moonRise*, *moonSet*, *heightHour*, without which they could not be reported in *xsd:dateTime* format; it was decided to set all missing *second* parameters to zero.
The transformation continues creating the *source* field and setting its value to *"Lamma"*, and then some accented characters (such as inside the Italian days of the week) are replaced because they can cause display problems later, as well as the degree symbol "°" which is instead simply deleted.
Thanks to the special Kettle step that allows to create a connection to a MySQL database, an access to the ISTAT code support table, is guaranteed, that is a MySQL table containing, for each municipality in Tuscany, the correspondent ISTAT code, useful for defining the class *Pa* identifiers.
The splitting of the transformation execution flow, finally, allows to save on HBase, data relating to individual forecasts, and also the so called "*Almanacco*" data, avoiding the redundancy of the latter on multiple row.

5.1.3 *Tuscany Region Open Data Ingestion*

The initial part of the main job (Figure 51) that deals with the CSV file ingestion, downloaded from the *Open Data portal* of *Tuscany Region*, has a similar structure as the main job that instead ingests weather forecasts.

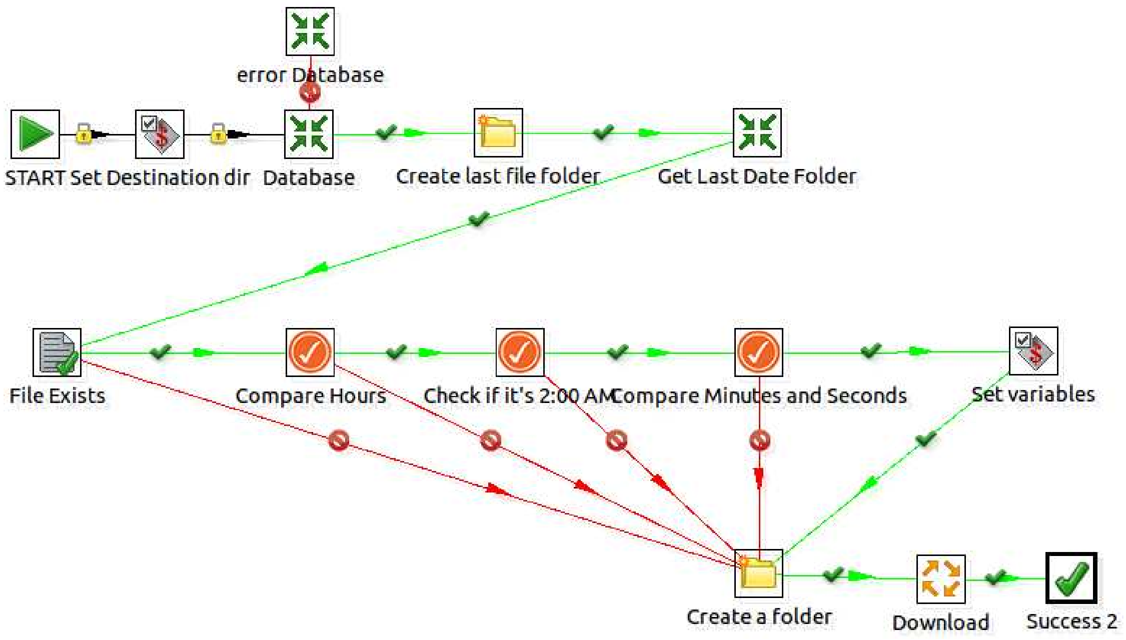


Figure 51- Ingestion transformation for Open Data of Tuscany region

In fact, the *Database.ktr* transformation is exactly the same, but the transformation continues with a series of controls, that allows to descend the folders' tree created within the directory, in which the downloaded datasets are saved, that verify also if the file has been updated. Only after this test, if the resource has been updated, it is downloaded.

Figure 52 shows the heart of the entire ingestion process, the *Files_Modification.ktr* transformation.
Initially a check to verify if the resource has been updated, since the last download, is performed, and if so, the *last_update* column of the *ProcessManager* table, will be updated with today's date, in order to track the latest update/download of each dataset.
Afterwards the downloaded file has to be read and such operation requires to set some CSV file properties, as the delimiter (set to comma in most of the examined dataset), so that the reading takes place correctly.

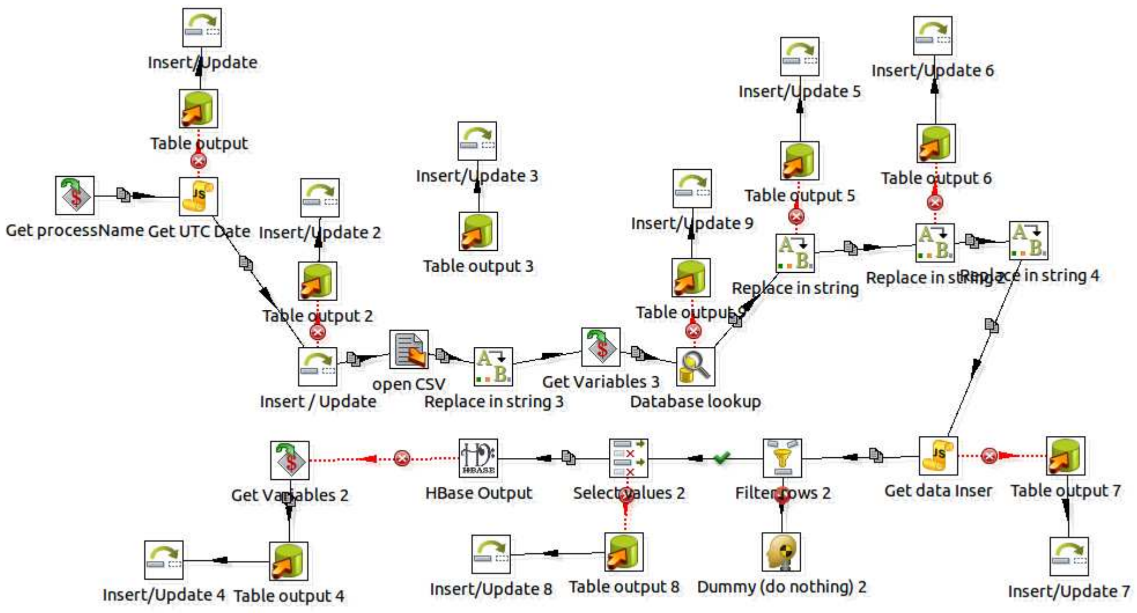


Figure 52 - Files_Modification.ktr transformation

To adapt a unique transformation for different types of CSV files, all coming from the portal of the Tuscany Region, the mapping of the fields to be taken, consists of all possible fields that can be found on these different files; in this way fields that are not present in the opened file, remain empty, and they will not picked up by next steps; therefore they will also not be saved on HBase.
The transformation continues recovering the *processName* parameter, and eliminating blank spaces in fields such as *email* and *address* and the comma in *address* field, because these two fields are used to create the HBase keys, an operation that does not accept special characters, such as punctuation. In addition to the key used on HBase, also the insert date must be formed, and this process takes place thanks to some lines of Java code.
Data is then filtered according to the value of *isService*, and only the values that make true this control, continuing execution, while the other rows are discarded. FInally, the step writing to HBase must be preceded by a *selecting data* step, defining which fields are actually stored on the NoSQL data store within the table called *regione.csv*.

5.1.4 *Tram Line Ingestion*

Data relating to the only Tram line in Florence, belong to the category of static data, since a few updates to the dataset are provided, probably only if new lines are built; so, the automatic download process has been set to a very long period, and it is probably more convenient a manual control to start the download process.
The following figure shows the transformation that takes care of the ingestion of Tram dataset.

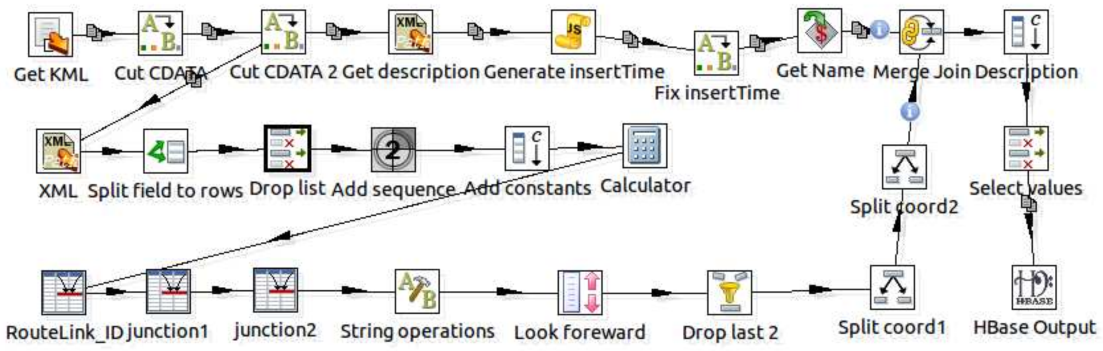
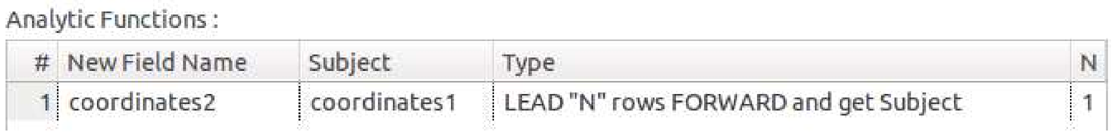


Figure 53 - Ingestion transformation for the unique Tram Line

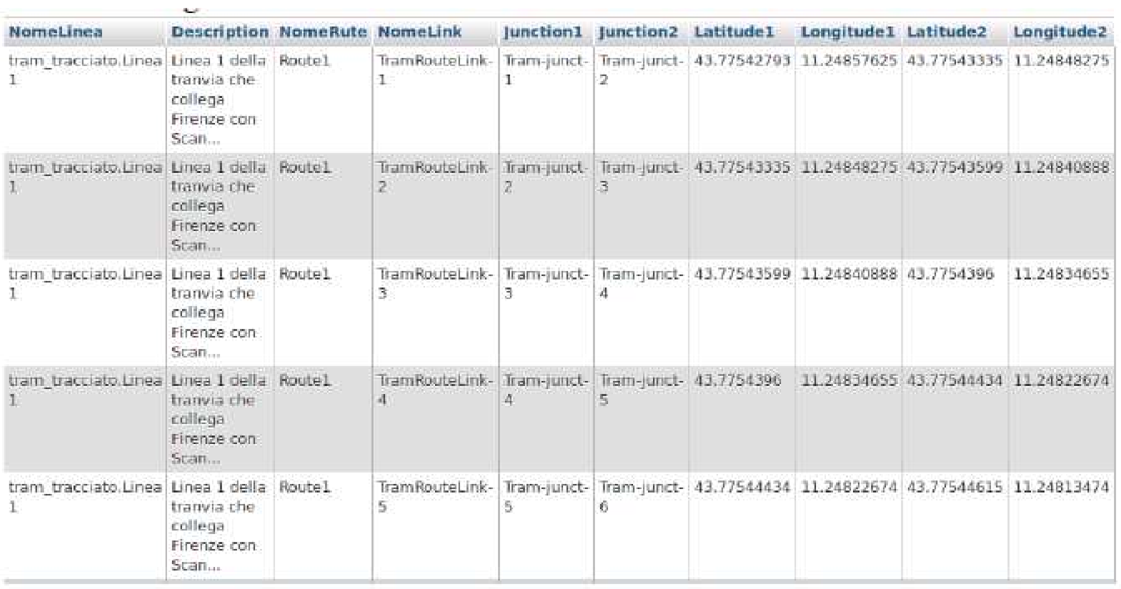
The transformation begins opening the *tram_tracciato.kml* file, without interpreting its contents, exactly as it was a single string of text.
The next steps delete the opening and closing tags of the CDATA field, contained in the KML file, i.e. "*<![CDATA[*" and "*]]>*", thus making possible the interpretation of the file by an XML parser.
The execution flow is then divided, in order to carry out the extraction and processing of data from the file, as parallel operations: each of the two processing branches, start with a step that extracts data from the XML file, respectively, the description and the general information at the beginning of the file, for the first branch, and the coordinates, for the second one.
Information extracted from the first branch of execution, does not require further processing (only the insert date is added) and it can be sent directly to the merge step. The second branch, instead, presents a far more complex structure: once the content of the field coordinates has been extracted, the string containing the set of coordinates is divided, and each pair of coordinates obtained, is inserted into a table.
The unique string that contains all of the coordinates is then eliminated, while an order number is associated to each pair of coordinates.
The *calculator* step, calculates two necessary integers to create identifiers for *RouteJunction* and *RouteLink* classes of the *Km4City* ontology. The *line feed* and *carriage return* characters, which would otherwise appear in triples as "*\n*" and "*\r*", are then eliminated from the various fields generated until now; the transformation continues with the creation of a field that contains the next coordinate pair to that found in each table row, to obtain new rows containing two successive coordinate pairs, which represent the starting and the ending point of each element, of the *RouteLink* class.
In Figure 54 is possible to observe settings for this crucial step.


Analytic Functions :

#	New Field Name	Subject	Type	N
1	coordinates2	coordinates1	LEAD "N" rows FORWARD and get Subject	1

Figure 54 - How to set Analytic Functions step

The last line in which there is only one coordinate pair, is deleted; the two fields containing coordinate pairs are splitted based on the comma position, thus creating four separate fields representing latitude and longitude of the starting point, and latitude and longitude of the ending point, respectively.



NomeLinea	Description	NomeRute	NomeLink	Junction1	Junction2	Latitude1	Longitude1	Latitude2	Longitude2
tram_tracciato.Linea 1	Linea 1 della tranvia che collega Firenze con Scan...	Route1	TramRouteLink-1	Tram-junct-1	Tram-junct-2	43.77542793	11.24857625	43.77543335	11.24848275
tram_tracciato.Linea 1	Linea 1 della tranvia che collega Firenze con Scan...	Route1	TramRouteLink-2	Tram-junct-2	Tram-junct-3	43.77543335	11.24848275	43.77543599	11.24840888
tram_tracciato.Linea 1	Linea 1 della tranvia che collega Firenze con Scan...	Route1	TramRouteLink-3	Tram-junct-3	Tram-junct-4	43.77543599	11.24840888	43.7754396	11.24834655
tram_tracciato.Linea 1	Linea 1 della tranvia che collega Firenze con Scan...	Route1	TramRouteLink-4	Tram-junct-4	Tram-junct-5	43.7754396	11.24834655	43.77544434	11.24822674
tram_tracciato.Linea 1	Linea 1 della tranvia che collega Firenze con Scan...	Route1	TramRouteLink-5	Tram-junct-5	Tram-junct-6	43.77544434	11.24822674	43.77544615	11.24813474

Table 4 - Results of Tram Line Ingestion Process

When also the execution of this second branch is finished, the *Merge join* step is performed: the result is the creation of a brief description associated with the tram line, and of keys, created as shown in Table 4.
Finally, the transformation ends with the selection of data that will be saved to HBase.

5.1.5 *Sensors ingestion*

The Kettle job shown in Figure 55, performs the ingestion of road sensors data.

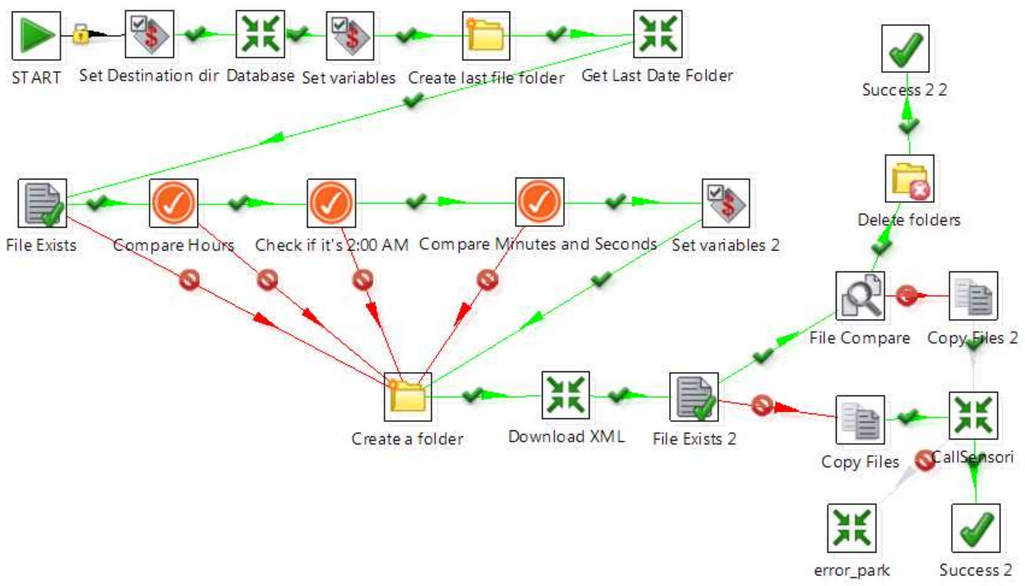


Figure 55 - Ingestion transformation for Road Sensors

This transformation has a structure similar to some already seen that allows to check whether the available data have been updated, and eventually create the correct folder path where they will be saved. The transformation shown in the following figure, is in charge of data download.

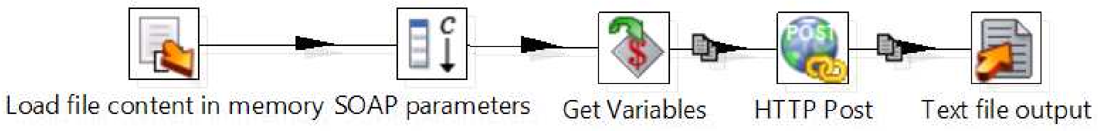


Figure 56 - Transformation to download data

This transformation uses a file *request.xml*, saved in the same location of the transformation, to make a SOAP request to the web service, which will provide the data detected by road sensors, in real time.
The transformation is also involved in the current date creation process, that must match the *xsd:dateTime* format (recommended by the W3C), and then it will enhance the variable *dataInserimento*.
Then the variables that contain the parameters of the SOAP request, are defined. A key parameter for querying the web service, as also seen in Section 3.5.1, is the catalog number, represented, in the transformation execution flow, by the *catalog* variable.
Thanks to an HTTP post, the web service of sensors can be invoked, using the URL provided by the *MIIC* and the *catalog* variable properly valued: the result is an XML file that contains information about the sensors of the indicated catalog.
The fundamental part of the main job, is represented by the *callSensori* transformation, that is shown in Figure 57.

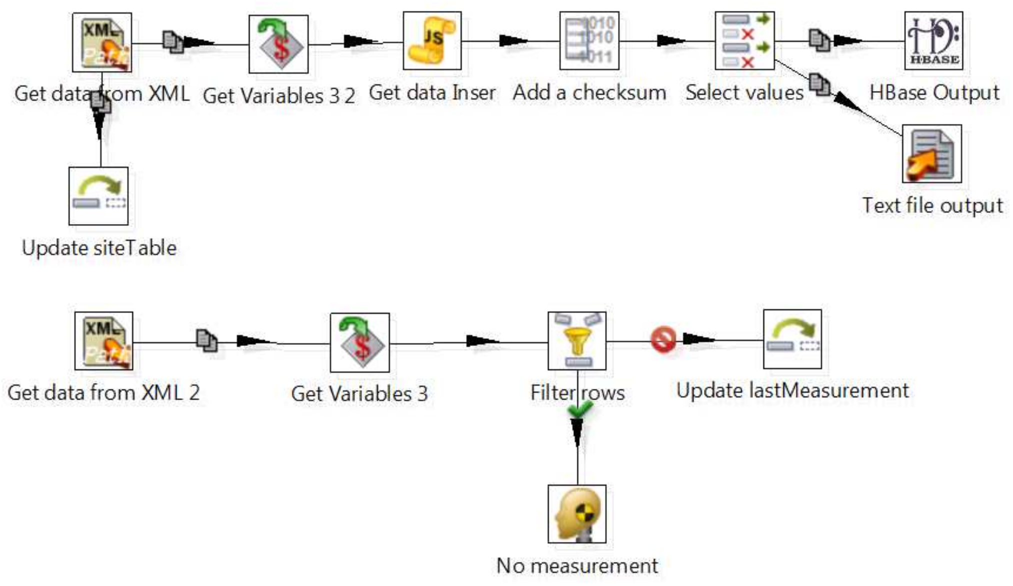
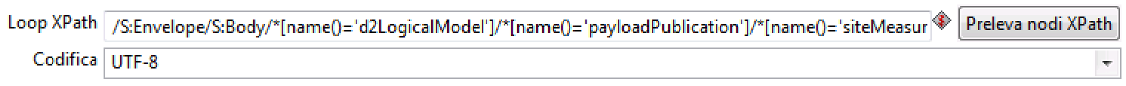


Figure 57 - The CallSensori transformation

The transformation sarts with data extraction from XML file, thanks to *XPath* [XPATH].
The last measurement date, contained in the received XML file, is properly stored in the *ProcessManage*r MySQL table, within the column *last_update*, so as to keep track of when the latest information from the Sensors Web Service, have been received.
The *payloadPublication* tag has been set as a loop node, because the most important information is contained within more tags *measurementTimeDefault*, located inside the *payloadPublication* tag; if some communication problems, with the server, occur, the latter tag will be empty and then, in the transformation, a control step to verify if *payloadPublication* is null, is also set; therefore, only if the tag is not null, the execution flow continues upgrading the correspondent field, in the *ProcessManager* table.



Loop XPath /S:Envelope/S:Body/*[name()='d2LogicalModel']/*[name()='payloadPublication']/*[name()='siteMeasur
Codifica UTF-8

Figure 58 - XPath Loop for sensors

A further *Xpath* loop is then set on another payload tag, that is *siteMeasurement*, thanks to which the following data internal to the tag, can be recovered: *measurementSiteReference*, *measurementTimeDefault*, *concentration*, *occupancy*, *vehicleFlow*, *averageDistanceHeadway*, *averageVehicleSpeed*, *threshold* and *value*.
In addition to these fields, even fields "external" to the loop node are selected: *measurementSiteTableReference*, *supplierIdentification* and *publicationTime*.
For each elements within the *siteMeasurement* tag, a new line, containing all information retrieved, is stored on *HBase*. The new obtained data is also used to update

the MySQL table *sensors_details*, where static information of each sensor, are written and where later, thanks to the additional information retrieved from the website of *Osservatorio dei Trasporti*, other information such as the toponym name and the toponym code, will be added.
For each *measurementSiteReference* element, the table field *siteTable* is updated with the *measurementSiteTableReference* value, where the condition *codiceSito=measurementSiteReference* is verified; in case the correspondence with the identifier is not found, the new identifier will be added into the *sensors_details* table, thanks to a SQL insert.
In Table 5 an extract of the table *sensors_details* is shown, when the toponym name and its code have already been added.

Site Code	Site Table	Toponym	Toponym Code
GR0100801	GR01008	Via sidney sonnino	RT05301104290TO
GR0100802	GR01008	Via sidney sonnino	RT05301104290TO
GR0101001	GR01010	Via nepal	RT05301101153TO
GR0101002	GR01010	Via nepal	RT05301101153TO
GR0101003	GR01010	Via nepal	RT05301101153TO

Table 5 - Sensor details Table

The sensor data are real time data, and key associated with HBase rows must be created carefully, so as not to create duplicates, given the large number of elements, that will be created daily. For this purpose, an MD5 checksum transformation is used to creates keys.
Even in this transformation, some concatenation steps are used, which produce, as seen above, the addition of special characters such as *line feed* and *carriage return*, that must be eliminated.
Regarding the Real Time data storage on *HBase*, Kettle requires some information to establish a connection with the master machine, i.e. where the repository is located. Moreover the exact mapping, that defines the correspondence between fields, of the transformation flow, and columns created on HBase, must be created. Also a key field must be defined, represented in this case by the unique identifier previously seen.

5.2 Phase II: Data Quality Improvment

As a result of a detailed analysis of ingested data, a long series of inconsistencies were found, especially related to the content of the services dataset, most likely derived from an export operation not carried out perfectly: clearly, it has a negative effect on the information amount that this data, suffering from problems, are able to provide. It was therefore decided to implement a data quality improvement process in order to recover the greatest amount of information as possible; this process, as it is possible to see from Figure 25 (Phase II), will be executed before the triples transformation phase (called

Phase III in the same figure) and it involves data extraction from HBase and storing them at the end of improvement, always on HBase, but on other table.
Such a process was carried out using the ETL tool *Kettle*, in order to ensure continuity to the ingestion processes already made.
From the analysis of ingested data, the following errors were detected and classified as correctable:

- Multiple telephone numbers;
- Typos in web site address;
- Some services have in place of the province, the final part of the address string, or an empty cell;
- Some services have problems with the municipality name: wrong name, typo, empty cell, number instead letters, location name instead of municipality name;
- Some service have charset encoding problems in various fields;
- Almost all phone and fax numbers are devoid of the initial 0;
- Instead of a phone number, some services show "Call municipality";
- Instead of email address, some services show the "email" string;
- Instead of web site address, some services show the "no" string;
- Syntax problems on the web site address;
- In some record, the service name is missing;
- There is no unique format for phone numbers and fax; in addition there are a number of errors on telephone numbers: in the case of multiple numbers, the prefix is often written only the first time, or in other records in the first number is written correctly and the others have only the last different digit;
- Many telephone number are written in exponential form (i.e. 3,90585E+11): this is clearly a conversion error from string to number;
- Some services only have the string "39" as the phone number, that is part of the international dialing code for Italy;
- The most frequently errors on web site address are double "http", several typos on the "www" string such as "ww", "wwww", "www:." etc.
- Domain missing in the web site address
- All CAP are missing;

In summary, the main problems are all related to the fields containing the service address, the service name, its email, its website, its telephone and fax numbers.
The following are the actions that will be carried out on each fields covered by the QI process; for each of them, the performed steps allow to eliminate most of the errors encountered.

- **Phone e Fax:** To delete blank space after prefix (all possible prefixes must be defined); To add 0 to landline telephone or green number (187), if missing; To check the number's length (for eliminating partial numbers); To divide numbers with multiple digits at the end (both in the presence of “/” that of “-“).

- **Address:** To separate house number (or "*SNC*", a string that means "*without house number*") after the comma.
- **CAP:** Need to be inserted.
- **Province:** To clear fields where more than two characters appear; Fill all empty cells based on the Municipality field
- **Website:** To check that field starting with http:// or www; To check that in field there are at least two full stops and, after the second one, there is at least one character.
- **Email:** To check that there is a @; To check that there is a full stop after the @, and a character after the full stop; To check that text before the @ has at least one character.

Of course, not all problems afflicting all datasets and then a job *Kettle* quite dynamic, which is appropriate for the different cases, need to be created. In addition, individual transformations are created, each of which manage a single field and its main problems, and they will be discussed in next sections.
Unfortunately, *Kettle* does not allow to create fully dynamic transformations, which are able to adapt themselves to the data content supplied as input. However, it is possible to create a mapping system, using a *MySQL* support table and the *ProcessManager* table, where all useful information about ingested datasets are stored: by performing the QI process, that is the same for all data sets, the execution will follow a precisely path in the entire execution flow defined, which is chosen according to the data that must be processed, to the contents of the *ProcessManager* table for that dataset, and to the corresponding value in the mapping table. Data related to services has some regularity, so it was enough to realize six different QI branches, which correspond to different sequences of issues to be deleted, and to map them in the mapping table. It is obvious that, for each new dataset ingested that presents a different structure of fields, a new branch in the QI transformation, must be realized and introduced in the mapping table.
To make more flexible the job acting on services, each processing branch has been realized as a separate sub-transformation, so that, for each type of file processed, thanks to an additional information which identifies the type-transformation correspondence, the right sub-transformation can be performed. This information is stored in a parameter initially provided as input to the job, and which is then transformed into a variable, so to make it visible during the whole execution.
In addition, the fields within the files on which the QI acts, may have different names (for example the *"telephone"* field can also be called *"phone"*), then the sub-transformations realized must take into account this additional factor of variation.
The structure of the realized *Quality Improvement* process, consists of a main job *Data_QI.kjb*, that calls the appropriate sub-transformations, depending on the class value, that identifies order and name of the resources, on which the QI have to act.
In particular, each class includes all the resource with the same structure; this class is stored inside *Resource_Class* column of the MySQL table, *ProcessManager*. When

calling for a *Quality Improvement* process on a particular file, a query to the *ProcessManager* table is performed, to retrieve the *Resource_Class* of the file. Based on the result of the query, the main job, runs the appropriate sub-transformations, which name is composed by the class name with a suffix referring to the process type, that is, *"class + _QI.ktr"*.

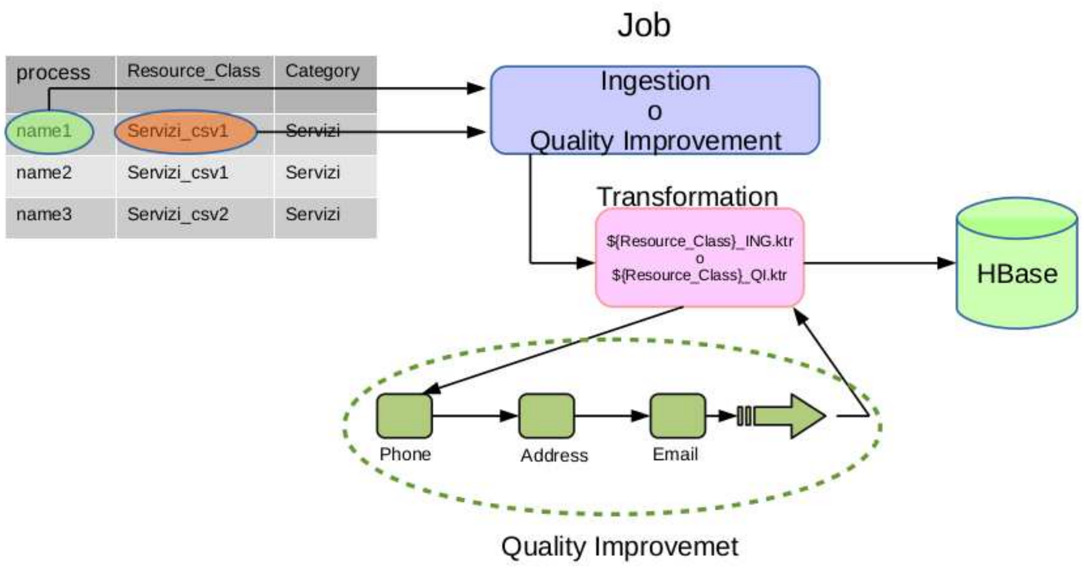


Figure 59 - Quality Improvement schematization

The Kettle's Job representing the backbone of Quality Improvement phase, initially recovered from the *MySQL* database, the last update date; this value is then used to recover, from HBase, only records relating to the last entered data, which will be used by the next steps.
Afterwards, a cascade series of transformations are executed, each of which deals with a specific field of the dataset, which will carry out a first QI phase (only relating to the data format).

Below, for each field subjected to QI, the relative transformation, that makes corrections, will be analyzed.

5.2.1 *Phone and Fax QI*

The *phone* field should be a numerical value, but in various files analyzed, the presence of non-numeric characters has been found in this field, and especially of different format and configurations.
In particular, some specific patterns were found, which are collected in Table 6:

Pattern	**Example**

Presence of alphabetic characters	**O**55 9162635
Presence of special characters such as / - .	0572/67352-3-4
Only a few fields have the international prefix 0039 or +39	**+39** 0577 300020 **3,9E+11**
Numbers written in accordance with the exponential notation	#######
Presence of blank spaces	0564 858111
Lack of 0 before the regional/provincial prefix	5,5E+07
Presence of multiple telephone numbers (2-3)	057161744 - 057161199
Presence of multiple numbers (suffixes extension number)	0572/67352-3-4
Partial/incomplete numbers	39

Table 6 - Telephone field most frequently errors

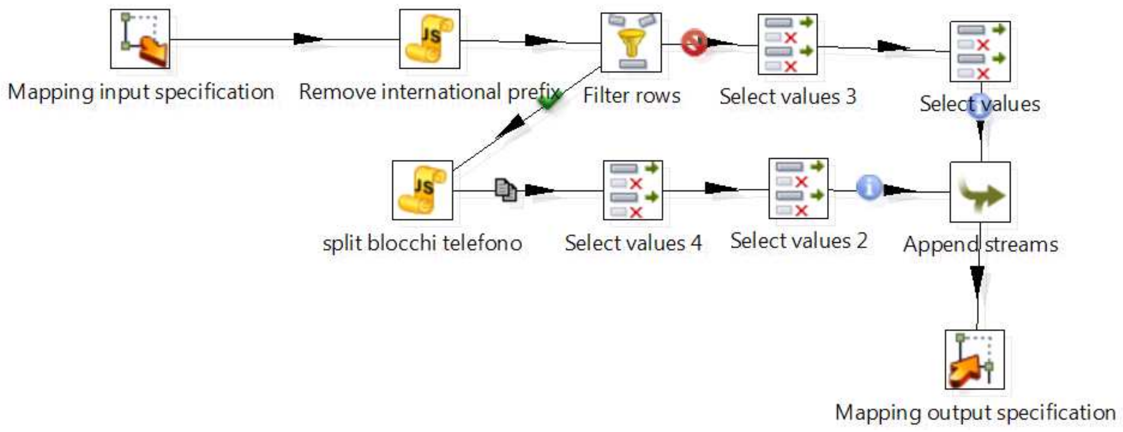


Figure 60 - Telephone Quality Improvement Process

The Quality Improvement process, shown in Figure 60, has the aim to recover part of the content information of the *phone* field, and it is based on the following assumptions:

- Prefix (for both landline and mobile numbers) is composed by 2-4 digits.
- The phone number is composed by 6-8 digits.
- Suffix for internal numbers, is composed of maximum 3 digits.

Furthermore, the *blank space* will be the only delimiter used, between prefix and number and between several numbers, then to proceed to the recovery of the main telephone number and any additional numbers, that will be reported in the *note* field.
Once the value of the phone field was recovered, some preliminary transformations are performed (using Javascript code), so as to bring the data into a suitable format for further processing. In particular, the following operations are performed, in the same order shown in the list:

- The national prefix, in format 0039, +39 e 39, is removed.
- The international prefix +44 is replaced with 0044.
- Any *blank space* before the number is removed.
- All *slash* (/) and *em dash* (-) characters present inside the field, are replaced with a blank space.

- All special and alphabetical characters are removed.
- All multiple *blank spaces* are replaces with a single *blank space*.
- If at the beginning of the number, the *zero* digit is missing, it will be added.

After that a filtering step is performed, to check whether any spaces appear into the number: fields which do not contain *blank space* are routed directly to the *Append Streams* step, whereas those in which some *blank spaces* are instead detected, they are sent in input to a further Javascript step.
This step aims to extract the main phone number from the input string, and it is based on the assumption that *blank space* is the separator between prefix and telephone number; after the extraction, the phone number will be saved in the appropriate *MySQL* field, while the remaining string, possibly containing any additional information or any other secondary phone numbers, will be saved inside the *Note* field, with the label "Additional Phone:". This processing is necessary because, during the triples creation phase, each service will be associated with a triple to only one phone number, in the required format; other numbers, however, can be moved to the *skos:note* triple, which contains all additional details on the service, avoiding the loss of any information.
Finally, the result of these operations is again merge with the other rows initially separated that did not contain spaces.

The *Fax* field transformation, is very similar to this just seen, its main action is relating to the mobile phone numbers elimination, since this information cannot be real.

5.2.2 *Address QI*

The *Address Quality Improvement* process is more complex than the previous ones, mainly because it was decided to split the house number from the street name.
In most cases, this field has the following form: "*street name, house number*", but sometimes, this standard is not observed, and in particular it can be written without the comma, that is "*address number*", or in the worst case, one of the two parts may be missing.
Furthermore, the address, with or without house number, can be followed by an additional text, containing variable information such as fraction, locality, mileage position, or also the text that preceded the house number can correspond just to the town, the locality or to a more generally indication.
The *Quality Improvement* process, is split in two execution branches: the first part deals with the house number extraction (possibly followed by additional text), from the entire field; the second one, instead, deals with the extraction of additional information that can follow the house number.
Given the large number of records to be analyzed (more than 30,000), to speed up data analysis, a "*data profiling*" tool has been used, which quickly allows to extract all observed patterns inside the address field, within the over 30,000 records; this open source software is *DataCleaner* (http://datacleaner.org/).

In Table 7 the main patterns found, thanks to the data profiler, are described.

Pattern	Esempio
Address, house number	Viale dei Mille, 11
Address house number	Via Del Belvedere 26
Address, house number additional text (1)	Via delle Torri, 20 - Cisanello
Address, house number additional text (2)	Via Oppilo, 00 - Oppilo, 00
Address, house number additional text (3)	Via Pino Gorgognano, 451 (Pino)
Address, house number additional text (4)	Strada Pianacce, 131 - - ALBERESE
Generic text (1)	- Selvaccia - Pietraviva
Generic text (2)	LOC. REALE
Generic text (3)	Loc. Contra - Fraz. Nicciano
Generic text (4)	Via Iano (Iano)
Address, double house number	Via Ferrucci, 11-13
Address, house number (terminating with a character)	Via Bartolini 1175/a
Address, house number, house number	Via delle Pinete, 3, 3
Address double house number (without a comma)	Via Nardini 11/13
Address house number additional text	Via Piave 9 (Pontassieve)

Table 7 - Address field most frequently errors

Regarding to the house number extraction, the first operation performed divides the address field, where the comma is located (if it exists); the two new fields, *addressOnly* and *civicNumber*, contain respectively the string preceding the comma and the string following the comma.
After this operation, some Javascript code is applied to *addressOnly* field: thanks to a regular expressions, the first occurrence of a number, inside this string, can be located and, if it exists, the *addressOnly* text is again divided into two sub-string. The first sub-string certainly contains only text, while the second one will contain the identified number, followed by any additional text. If the text contained in the first sub-string is shorter than six characters (which statically happens when the string contains the dug of the address, that is "street", "avenue", "square", etc.), then the two sub-string previously separate, are merged again: this is the case of street names contain a number, such as "*Via 1 Maggio*", "*Via 2 Giugno*".
Otherwise, a second check is then performed on the first sub-string derived by the *addressOnly* division, to verify if it ends with *"KM"*, *"CS"*, *"SS"* or *"SP"*: in fact, in these cases, the number at the beginning of the second sub-string, need to be recovered, and merge with the first part, because for example, *"KM"* is usually followed by the mileage value, while *"CS"*, *"SS"* or *"SP"* are usually followed by the road number.
A further text division of the second sub-string, is performed if it contains an *em dash* "-": for example, the second sub string can be "*Loc. Contra - Fraz. Nicciano*", so, in this case, the text following the special character is append to the field *civicNumber*.
The final part of the Javascript code, eliminates some special characters that can be found in the two fields created from *addressOnly*.

The output fields of this transformation are called *address1* and *civic1*, names that make clear which is the content of each field.

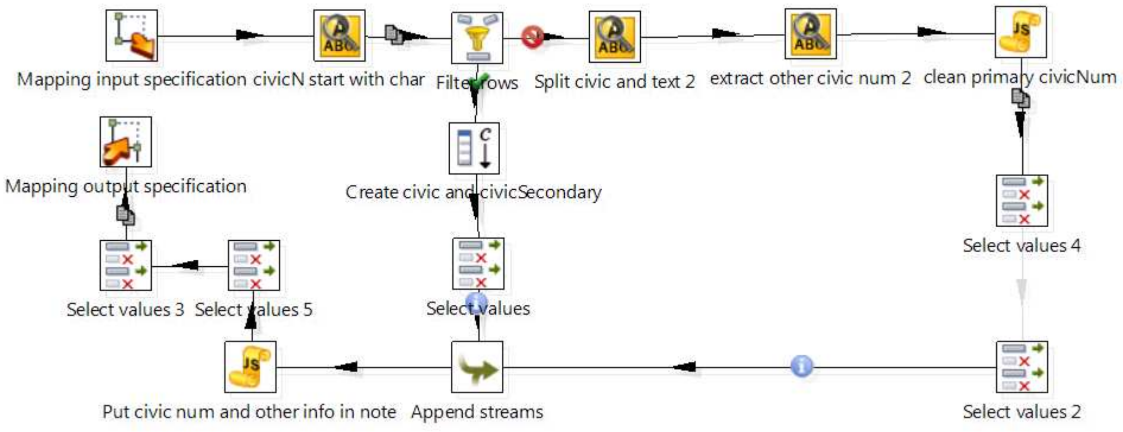


Figure 61 - House number cheking process

The second transformation, shown in Figure 61, checks if the *civicNumber* field, start with a character, instead of a number, because it is not able to recognize if any number is an house number, but it is expected to found the house number at the beginning of the *civicNumber* field. The execution flow is divided into two branches: on the vertical branch *civicNumber* fields that do not begin with a number are processed, while on the horizontal branch continuing those that begin with a number, which are again separated into two fields *civicNum* and *civicText*. This division takes place where an arbitrary number of *dashes* and *blank spaces* are founded (for example *"6 - - SCARLINO AIRPORT KM.222"*).

civicNum will contain the number above special characters, while *civicText* will contain the text following the number. With a regular expression, also civic numbers like "*91BIS*" or *"91A"* can be extracted. The field *civicText* previously created, is then split into two parts, where an arbitrary number of *blank spaces* and *dashes* were detected, creating the fields *civicNum2* and *civicText2*; the special character *slash* at the beginning of the string, is also removed, if it exists.

The step "*Clean primary civicNum*" takes as input the fields *civicNum*, *civicNum2* and *civicText2* and tries to enhance their readability: its output is composed of two fields, that is *firstCivic*, containing the primary house number (the first number appears) and a string with additional information.

As in previous transformations, the main value identified, that is the house number and the street name, are respectively saved on *civic number* and *street name* fields, while the other additional information are append to the *Note* field.

5.2.3 *Service Category QI*

Each service coming from the Tuscany region datasets, is associated with a *ServiceCategory* in Italian. Inside the *Km4city* ontology, actually, service categories are defined in English, in order to make the ontology more usable. It is therefore necessary to associate to each service, an English service categories, thanks to the dictionary specially created, that is a MySQL mapping table, containing all the Italian-English matches. The dictionary and the correspondent Kettle transformation, are easily adaptable to other languages, just put the new translations in the dictionary and associate them with the categories defined in the ontology.

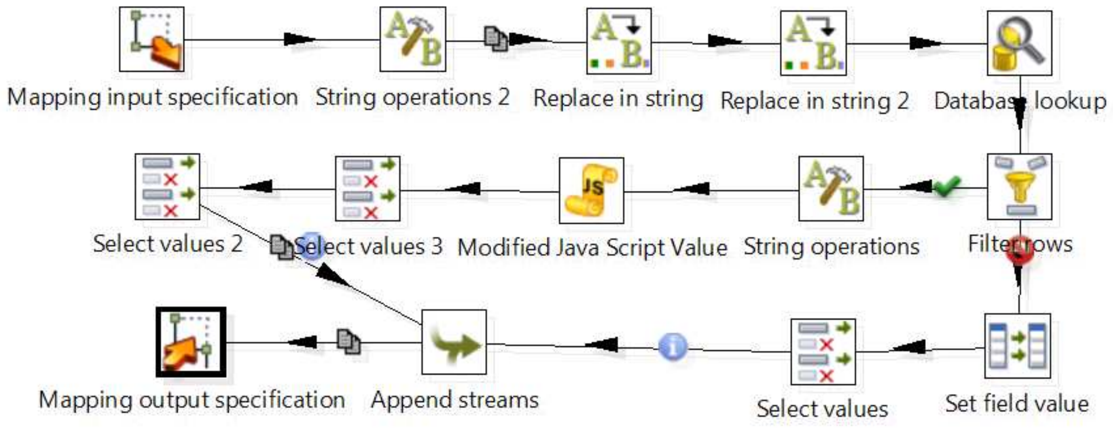


Figure 62 - Service Category Quality Improvement process

Initially, the *category* field of each record, is selected and cleaned from special characters (often there is, in fact, an *underscore* in place of *blank space*), because it is also used to make a query on the MySQL table *ServiceCategory*. This query has the purpose of extracting the English translation of the category, only if this is actually present in the dictionary; if the query is successful, the English recovered category is included in the new field *CategoryEng*, otherwise the field is left blank.
In case the *category* field contains the simple string "*services*", a special management is provided: these lines are subjected to a series of modifications, and then to a Javascript code, that search some keywords in the *Name* field modified to determine the English category, according to the default mapping (see Table 8).

Keyword	English Category
Associazioni/associazione	other_office
Comune	other_office
CGIL/CISL/UIL/CAF	other_office
Consultorio	family_counselling
Biblioteca	Library
ASL/azienda sanitaria locale	local_health_authority
Ambulatorio	doctor_office
Arci/circolo ricreativo	social_center
Ufficio	other_office

Sportello	other_office
Centro/centri	community_centers
Presidio socio sanitario	group_practice

Table 8 - Some association examples for Service Category field

The English category identified is finally included in field *CategoryEng* and each record is then processed by the following steps, that join together the other information with the new category.

5.2.4 *City QI*

The field containing the name of the city has, in most cases, a correct value, and the few errors detected are limited to the following:

- Empty field;
- Fields contain Municipality name/fraction name /locality name with typos.

To eliminate also these types of error, a Kettle transformation (shown in Figure 63) has been realized, which is able to perform a series of field evaluations on field province of the analyzed record.

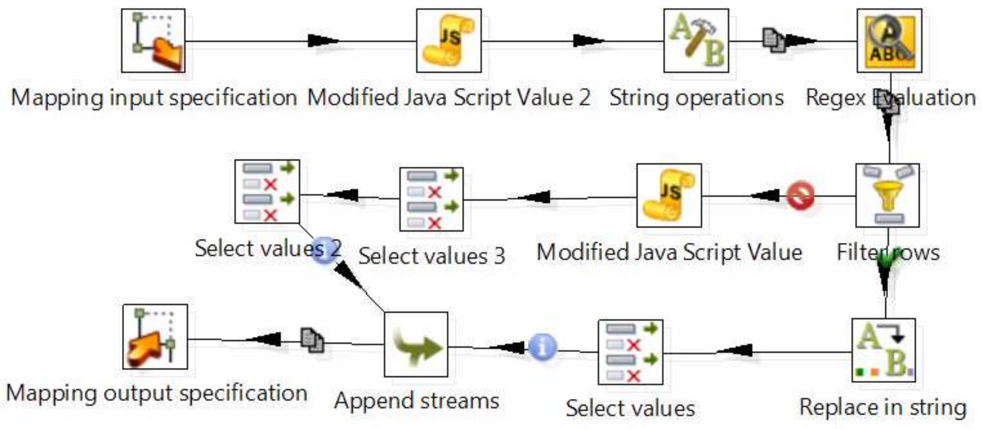


Figure 63 - City Quality Improvement process

Firstly, thanks to a Regex Evaluation the *City* field is analyzed to determine whether the value of the municipality name/locality name/and fraction name, is correct or not; lines containing the correct value are submitted to an operation that removes special characters such as the *em dash* "-", the other lines, however, are processed by a Javascript code, specially created, which eliminates spelling errors by the name of the city. The results of both processing branches, are then merge. If the city field is instead empty, unfortunately the available information does not allow to recover the correct value, and then it is leave blank.

5.2.5 *PostalCode QI*

Within the service datasets, most of record have an empty *PostalCode* field, so, using the other address information, such code can be recovered.
The transformation made to place the cap in its column, is very simple, and it first performs a series of assessments on the field *PostalCode* value, in order to separate empty fields from those that already contain a value, which is assumed to be correct.

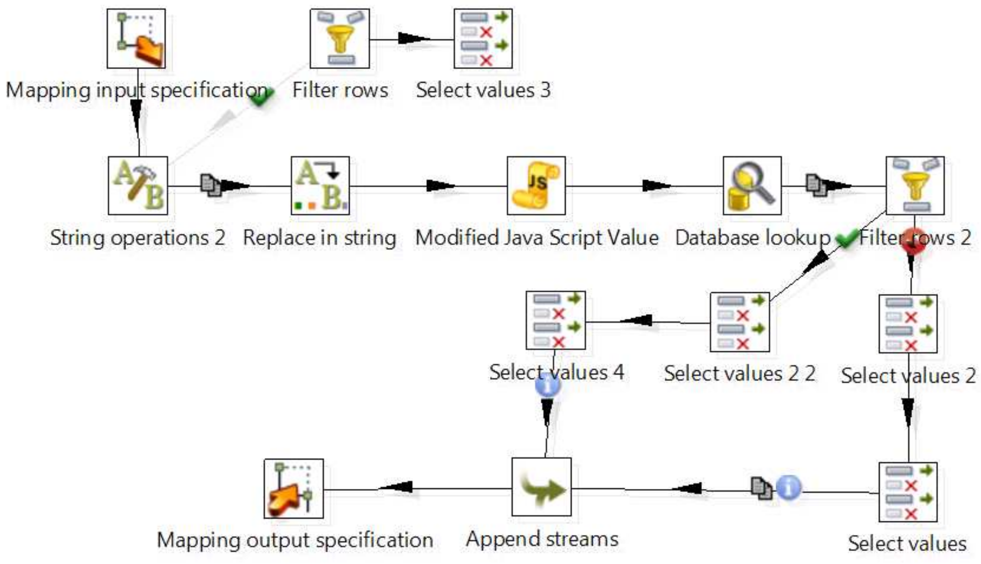


Figure 64 - Postal Code Quality Improvement process

Thanks to *Province* and *City* information, a query can be performed on the MySQL support table, containing all PostalCode of the Tuscany region, in which the conditions "*District like CityMod*" e "*Province like province*" need to be checked. If the query is successful, the recovered PostalCode is entered in the correspondent *cap* field, otherwise the field is left empty.
Finally, data from the two streams come together again, ready to be forward to further transformation.

5.2.6 *Province QI*

In most records, the *province* field contains the corresponding code to the belonging province of each service (e.g. "*FI*", "*AR*", "*PI*" etc.), however there are some cases where the field contains other types of misinformation. As can be seen from Table 9, the wrong information contained in the *cap* field, can be locality names, fraction names or partial addresses (with or without the street number).

Example Province field content	Pattern
e, 18 - Loc. Arsenale - Fraz. S.Piero in Campo	Locality + Fraction

22 - loc. Crespiano, 22 LOC. POGGIO	Locality
e di Mercatale, 25	Address
e di Pescaiola, 83/G - Viciomaggio -	Locality + Address

Table 9 - Province most frequently contents

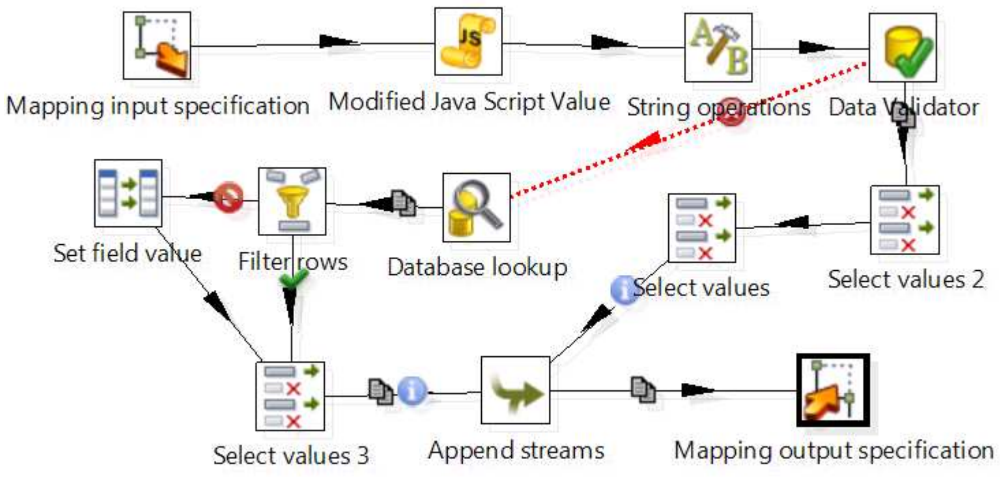


Figure 65 - Province Quality Improvement process

The transformation that takes care of the province QI phase, initially converts all content in "*upper case*" format, because in Italy the province, is usually expressed with capital letters. Then the values already made up of only two letters are divided from the others, which will instead be subjected to further evaluation.
Thanks to a query on MySQL table *MappingCity*, which trying to extract the correct province abbreviation, by testing the condition *District like CityUpper* (where *District* is a *MappingCity* field and *CityUpper* correspond to the *City* field modified to "*upper case*"), the province code can be retrieved, if the query is successful. The identified value is then copied inside the *province* field, replacing all the original content; if the query does not identify any correspondence, the contents of *province* field remains unchanged. The records from the two execution branches of the transformation, are finally gathered.

5.2.7 *Email QI*

The transformation that handle the email QI phase, i.e. *Modify_email.ktr*, is shown in Figure 66.

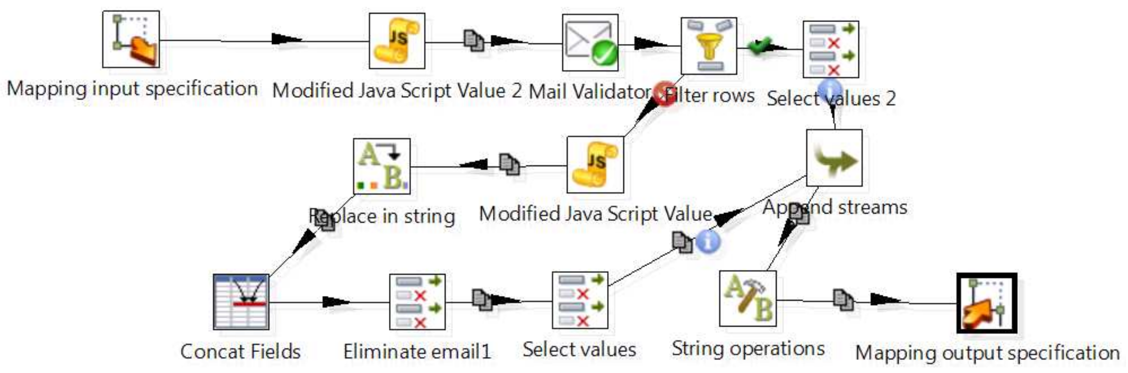


Figure 66 - Email Quality Improvement process

This transformation performs a series of assessments on email field, in fact, mainly thanks to a special regular expression, it is possible to identify those fields that meet the *name@domain.xxx* format, that represent the email correct syntax. On fields that meet this regular expression, no other transformation is carried out, while on the others, the evaluation proceed. In some cases the format previously indicated, is not met because of the lack of the "@" character, or the lack of the domain suffix at the end of the address, or for the presence of a double "@", or for the presence of special characters, not allowed in an email address. Table 10 shows some examples of error types just described.

Case	Example
Lack of suffix (it,com,...) after dot	agricolacheli@libero.
Lack of @	infochiantiandrelax.com
Lack of suffix (it,com,...) and the dot preceding it	info@lapievedipoggioallemura
Lack of dot before suffix (it,com,...)	rappuoli@valdorcia:it
Double @	info@bbacquacotta@yahoo.it
Illegal Characters (blank space included)	meloniluciano.))@tiscali.it ale.latini.alex.@tiscali.it albergolacasetta@abetone. com
Lack of dot before suffix (it,com,...)	albergo-italia@libero-it
Incomplate suffix (it,com,...)	osteriailpozzo@interfree.i

Table 10 - Email field most frequently errors

Initially, the fields containing two e-mail addresses in a valid format, but separated by one or more special character ("-", "/", ";", "-", *blank space*) are identified and separated from the other containing a single email address with incorrect syntax. This operation is performed thanks to a Javascript code that still uses regular expressions. The malformed email, are added to the *note* field, preceded by the string "*Invalid email :*" so as not to create a triple with wrong information, but leaving the information visible to a human, who could imagine the corrections to be made, to make it usable; double emails that were divided, are instead stored in the *email* field, the first one, by replacing the entire original contents, and in the *note* field, the second one, preceded by the string

"*Secondary email:* ". Even in this case, the results of the two processing flows, are gathered, before being forwarded to the next transformation.

5.2.8 *WebSite QI*

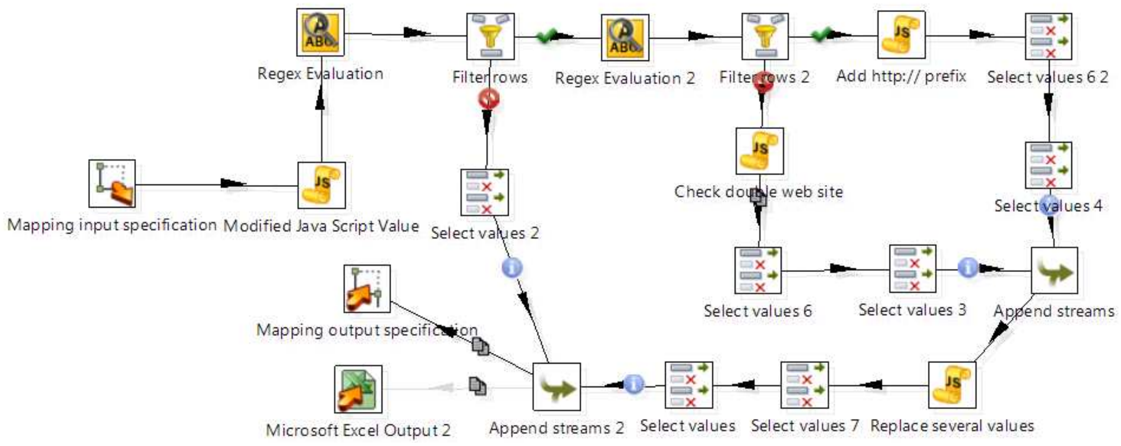


Figure 67 - Website field Quality Improvement process

The Kettle transformation that checking web addresses, i.e. *Modify_website.ktr*, initially controls, thanks to the support of regular expressions, if the website field is well-formed with respect to the standard pattern of a website URL, specifically:

- Website URL starts with a prefix like "*www.*", "*http://*", "*https://*", "*http://www.*", "*https://www.*";
- the prefix is followed by a series of alphanumeric characters;
- Website URL finishes with a dot followed by the web domain.

The fields that meet these characteristics are not subject to any change, while others are again compared with another regular expression, which checks whether the website field is devoid of the prefix, but still respects the original pattern, that is:

- A series of alphanumeric characters;
- Website URL finishes with a dot followed by the web domain.

If the website URL corresponds to this second case, the transformation adds the missing prefix, i.e. "*http: //*" or "*www.*" otherwise, a control to detect the presence of multiple web addresses, is realized.
Multiple addresses are then divided and the first is saved in the *website_output* variable, while the latter is stored in the *website_add* variable, which will be added to the *note* field of the *Km4city* ontology *Service* class. The data streams are finally gathered at the end of the transformation.
By analyzing website address values, which have not exceeded the first control, the different types of errors reported in the Table 11 were identified:

WEBSITE	Pattern
http:///www.agricap.it/ita/agriturismo.htm	Triple /
http://bebacquerello	Missing the final domain
http://http://albergogualtieri.interfree.it/	Double http://
http://lacasadeilimoni.interfree.it/	False positive
http://t	Incomplete
http://web.tiscali.it.valleantica	Invalid final domain
http://www il colle-siena.it	Blank space
http://www:agriturismo-tegline:com	: after www
http://www..tuscanyparkhotel.it	Double dot after www
http://www.agriturismo_stigliano.com	Underscore
http://www.agriturismocampagnellilit	Missing dot before the domain
http://www.borgolacapraia.com.	A final dot too many
http://www.campeggiodelforte.t	Incomplete domain
http://www.campingoasi.it/	False positive
http://www.casavacanzelibertà.it	Presence of accented letters
http://www.fantone@it	Presence of @
http://www.hotelcucciolo.com - www.paginegialle.it/hcucciolo	Multiple web address
http://www.lafornace..weebly.com	Double dot in the text middle
http://www.locandadell'agresto.it	Presence of an apostrophe
http://www.poggioaimonti/gabi.it	/ before domain
http://www.santalessandro:com	: before domain

Table 11 - Website field most frequently patterns

On the remaining data that does not belong to any typologies previously analyzed, a number of additional checks are carried out with the objective of correcting many of the errors that remain. Specifically, the following steps are performed:

- Expressions such as "*-com*", "*.co*", "*com*", "*:.com*", "*.com.* " are replaced with "*.com*";
- Expressions such as "*-it*", "*.i*", "*.t*", "*it*" and "*:.it*" are replaced with "*.it*";
- Double "*http://*", at the beginning, is removed;
- *Blank spaces* are removed;
- Expressions such as "*www*", "*www.:*", "*www..*", "*www.www*" are replaced with "*www.*";
- Expressions such as "*:org*" and "*.org.*" are replaced with "*.org*";
- Apostrophes are removed;
- Accented letters are replaced with non-accented letters.

After this additional correction phase, the different data streams are gathered at the initial flow containing websites that have originally corresponded to the correct pattern.

5.3 Phase III: RDF Mapping

The Mapping Phase deals with the transport of information, previously saved into *HBase* database, into an RDF datastore, in our case managed by *OWLIM-SE* [Bishop et al., 2011]. The first part of this procedure retrieves information from *HBase* to put them on a temporary *MySQL* database (required to use the Data Integration tool chosen), while in the second part data is translated into triples. A transformation is needed to map the traditional structured data into RDF triples, based on information contained in a well-defined ontology, that is the *Km4City* ontology, defined in Section 4.4 and all ontologies reused (*dcterms*, *foaf*, *schema.org*, *Wgs84*, *GoodRelations*, *Skos*). This process may be performed by ad-hoc programs, that have to take into account the mapping from linear model to RDF structures. This two steps process allowed to test and validate several different solutions for mapping traditional information into RDF triples and ontology. The ontological model has been several times updated and thus the full RDF store has been regenerated, from scratch reloading the definition (all the other vocabularies selecting the testing of several different solutions) and the instance triples according to the new model under test. Once the model has been generated, triples can be automatically inserted.

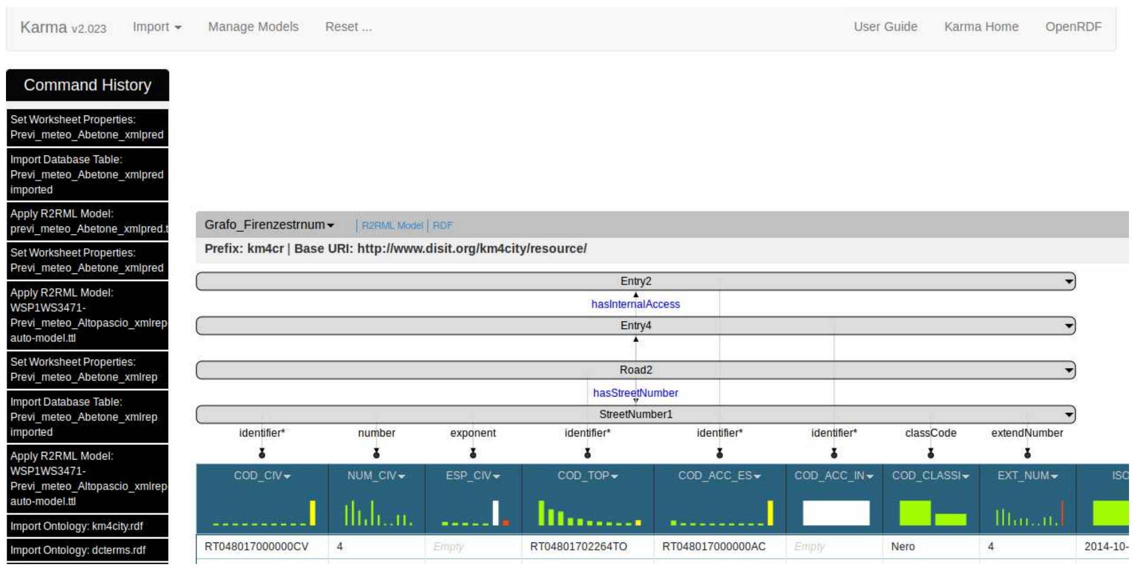
The first essential step is to specify semantic types of the data set, i.e., it is necessary to establish the relationship between the columns of the MySQL tables and properties of ontology classes. The second step consists in defining the Object Properties among the classes, or the relationships between the classes of the *Km4City* ontology. When dataset has two columns with the same semantic type but which correspond to different entities, thus multiple instances, of the same class, have to be defined, associate each column to the correct one.

The process responsible to perform the mapping transformation, passing from *Hbase* to MySQL database, has been produced as a corresponding ETL Kettle transformation associated with each specific ingestion procedure, for each data set. The second phase, that performing the mapping from SQL to RDF, has been realized by using a mapping model: *Karma Data Integration* tool [Gupta et al., 2012], which generates a R2RML model, representing the mapping for transport from *MySQL* to RDF, and then, the produced RDF data, is uploaded in a *OWLIM-SE* [Bishop et al., 2011] RDF Store instance. *Karma* is an information integration tool that enables users to quickly and easily integrate data from a variety of data sources including spreadsheets, delimited text files, *XML*, *JSON* and *MySQL* tables; this latter option has been selected for the project and then some *MySQL* support tables, have been also defined, in which the data to be mapped, will be temporarily stored, for the duration of the process itself.

Karma initialization phase involves loading the primary reference ontology and connecting dataset containing the data to be mapped.

The mapping process was performed for each dataset previously ingested within *HBase*. Due to the high number of datasets, in the following will be detailed examined only a few mapping processes, among the most significant within the ontology.

Regarding to the Street Guide mapping process, the following approach was used: for each available dataset, a portion of representative data has been extracted, and used for the manual definition of *R2RML* models, thanks to *Karma*.
To create the model, *Karma*, in addition to knowing data to which the model will be applied, must know all the vocabularies and the ontologies used; to this end, initially the individual upload of used ontologies must be completed, so that the tool can suggest, through the choice options, the correct classes and DataProperties.
Another thing to do before starting the *R2RML* model construction, is to set (i) the initial parameters, such as the default namespace of triples that will be generated, that is, within the realized project *http://www.disit.org/km4city/resource*; (ii) the connection parameters to the *MySQL* support database, that will be access during the mapping phase.
Karma is accessible via web browser, and its interface is shown in Figure 68.



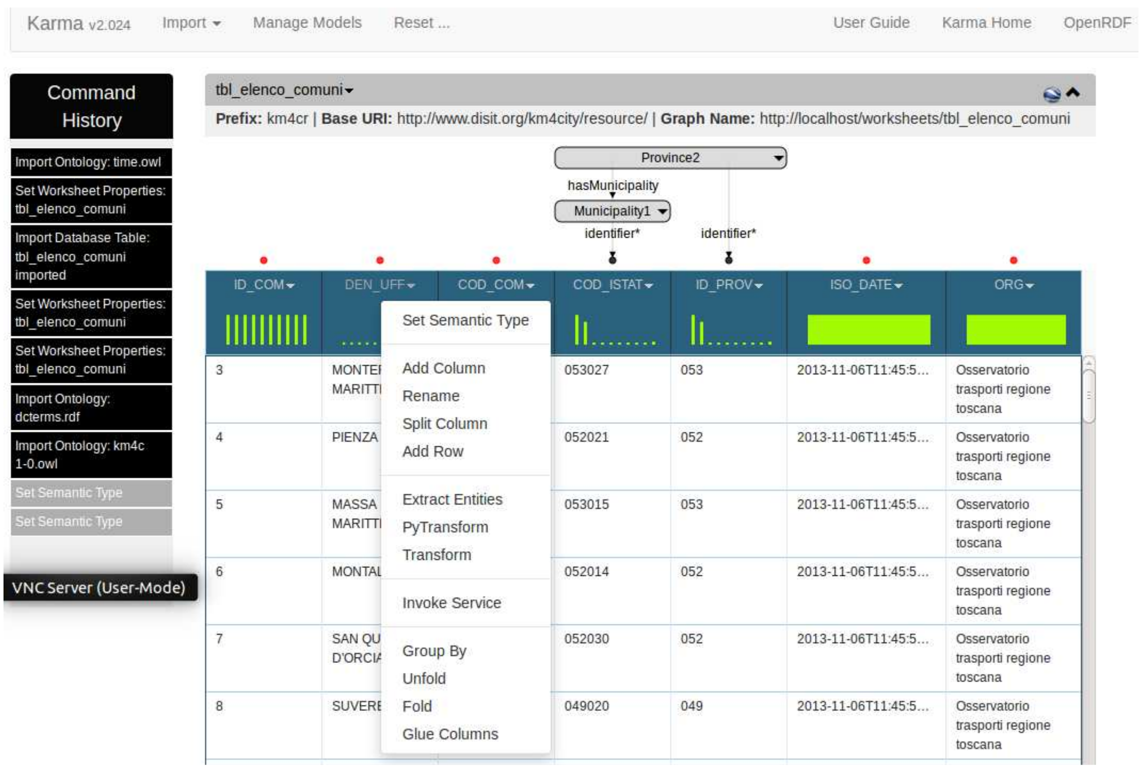
Set Worksheet Properties: Previ_meteo_Abetone_xmlpred
Import Database Table: Previ_meteo_Abetone_xmlpred imported
Apply R2RML Model: previ_meteo_Abetone_xmlpred.t
Set Worksheet Properties: Previ_meteo_Abetone_xmlpred
Apply R2RML Model: WSP1WS3471-Previ_meteo_Altopascio_xmlrep auto-model.ttl
Set Worksheet Properties: Previ_meteo_Abetone_xmlrep
Import Database Table: Previ_meteo_Abetone_xmlrep imported
Apply R2RML Model: WSP1WS3471-Previ_meteo_Altopascio_xmlrep auto-model.ttl
Import Ontology: km4city.rdf
Import Ontology: dcterms.rdf

Prefix: km4cr | Base URI: http://www.disit.org/km4city/resource/

COD_CIV	NUM_CIV	ESP_CIV	COD_TOP	COD_ACC_ES	COD_ACC_IN	COD_CLASSI	EXT_NUM	ISC
RT048017000000CV	4	Empty	RT04801702264TO	RT048017000000AC	Empty	Nero	4	2014-10-

Figure 68 - Karma Data Integration interface

To begin the data mapping, for example, of *Maneuver* elements of the *Km4City* ontology, data relating to the maneuvers, that are stored in a *MySQL* table, must be imported on *Karma*: to fulfill this action simply select the button "*Import Database Table*" located on the top left menu "*Import*", and, inside the dialog box appeared, the login information must be inserted, to access to *MySQL* database. After the connection has been created, *Karma* will display the list of tables that can be imported, including *tbl_manovre* that will be selected in this case.
The *Data Integration* tool will then re-create an HTML table containing the imported data, within its graphical user interface, that automates much of the mapping process, as shown in Figure 69.



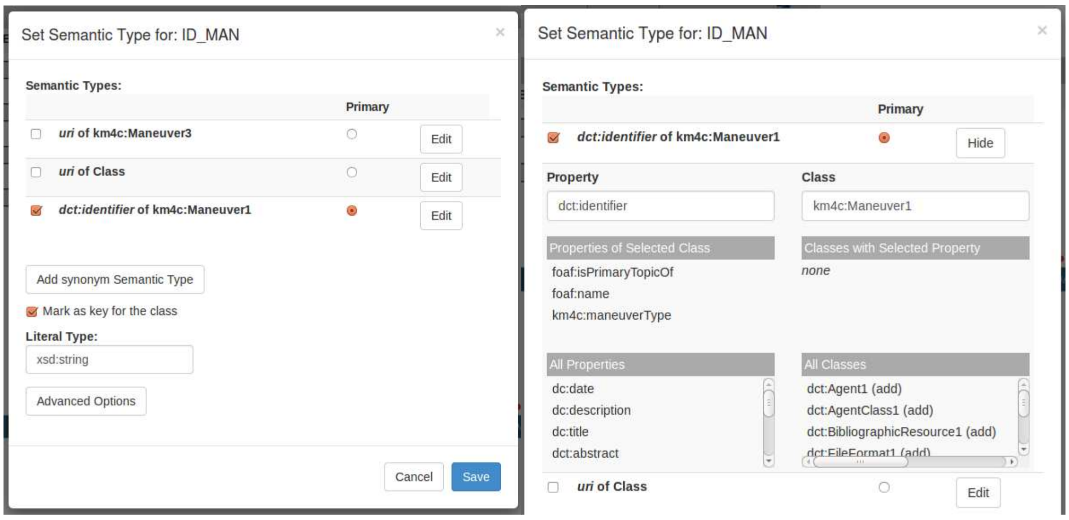
Import Ontology: time.owl
Set Worksheet Properties: tbl_elenco_comuni
Import Database Table: tbl_elenco_comuni imported
Set Worksheet Properties: tbl_elenco_comuni
Set Worksheet Properties: tbl_elenco_comuni
Import Ontology: dcterms.rdf
Import Ontology: km4c 1-0.owl
Set Semantic Type
Set Semantic Type

Prefix: km4cr | **Base URI:** http://www.disit.org/km4city/resource/ | **Graph Name:** http://localhost/worksheets/tbl_elenco_comuni

ID_COM	DEN_UFF	COD_COM	COD_ISTAT	ID_PROV	ISO_DATE	ORG
3	MONTEF MARITTI		053027	053	2013-11-06T11:45:5...	Osservatorio trasporti regione toscana
4	PIENZA		052021	052	2013-11-06T11:45:5...	Osservatorio trasporti regione toscana
5	MASSA MARITTI		053015	053	2013-11-06T11:45:5...	Osservatorio trasporti regione toscana
6	MONTAL		052014	052	2013-11-06T11:45:5...	Osservatorio trasporti regione toscana
7	SAN QU D'ORCIA		052030	052	2013-11-06T11:45:5...	Osservatorio trasporti regione toscana
8	SUVERE		049020	049	2013-11-06T11:45:5...	Osservatorio trasporti regione toscana

Figure 69 - How to set a semantic type with Karma

To begin to map columns, the small triangle, located on top of each column, must be selected; from the menu that will then be displayed, the "*Set Semantic Type*" option must be selected (Figure 69).
A semantic type defines the relationship between a column of data and a property or a class in an ontology. Semantic types can be specified in several different ways and can combine multiple pieces of information, and the most common way of constructing a semantic type, is to define it based on a property and a class in the project ontology.
Using the Data Integrations tool on multiple data sources related to the same domain (e.g., several data sources with data about museums), implies that *Karma* will learn the semantic types assigned to data, and it may offer them in future, as a suggestions; in fact Karma learns to recognize the mapping of data to ontology classes and then uses the ontology to propose a model that ties together these classes.
Within the displayed dialog box, a Class and the correspondent DataProperty, defined in one of the imported ontologies must be set, which represent the values inside the analyzed column (Figure 70).


Set Semantic Type for: ID_MAN

Semantic Types:

		Primary	
☐	*uri* of km4c:Maneuver3	○	Edit
☐	*uri* of Class	○	Edit
✓	*dct:identifier* of km4c:Maneuver1	●	Edit

Add synonym Semantic Type

✓ Mark as key for the class

Literal Type: xsd:string

Advanced Options

Cancel Save

Set Semantic Type for: ID_MAN

Semantic Types:

		Primary	
✓	*dct:identifier* of km4c:Maneuver1	●	Hide

Property: dct:identifier

Class: km4c:Maneuver1

Properties of Selected Class	Classes with Selected Property
foaf:isPrimaryTopicOf	*none*
foaf:name	
km4c:maneuverType	

All Properties	All Classes
dc:date	dct:Agent1 (add)
dc:description	dct:AgentClass1 (add)
dc:title	dct:BibliographicResource1 (add)
dct:abstract	dct:FileFormat1 (add)

☐	*uri* of Class	○	Edit

Figure 70 - How to specify a key for a class with Karma

After clicking on "*Save*" button, to assign the new semantic type to the selected column, *Karma* updates the model to show the new assignment.
To better clarify, for example, the *MySQL* column *ID_MAN* contains the unique regional identifier of each maneuver. Inside the *Km4City* ontology, this field corresponds to the DataProperty *dct:identifier* of the class *Maneuver*; this value is also used to create *URI* of the resource type *Maneuver*, i.e. the column is key for the class, so the "*Mark as key for the class"* option must be selected in the semantic type dialogue box, as in Figure 70. Inside its interface, *Karma* highlight with an asterisk, the semantic types with the "*Mark as key for the class"* option selected (see Figure 70).
Karma also allows to mark more than one column as key for the same bubble, and when do that, it will use the combination of all attributes marked as key, to construct the URI for the entities in the class.
For each columns it is possible to define the types of literals, using the *Literal Type* option in the panel for specifying semantic types. Karma offers the standard XSD types in a menu and also allows to enter an own URIs if that is appropriate for the application.
After defining correspondences between column and ontology, *Karma* show the created model, inside its interface as shown in Figure 71.
Often it may happen that there is need to assign multiple semantic types to a column: the semantic dialogue box allows to define multiple semantic types for a column, by selecting multiple checkboxes in the list of semantic type suggestions. One of the chosen semantic types must set as the primary one, by selecting the appropriate radio button in the *Primary* column. When more semantic types and relationships has been added to the model, *Karma* will connect the primary semantic types to the other parts of the model; the non-primary semantic types will be used in the publishing phase.

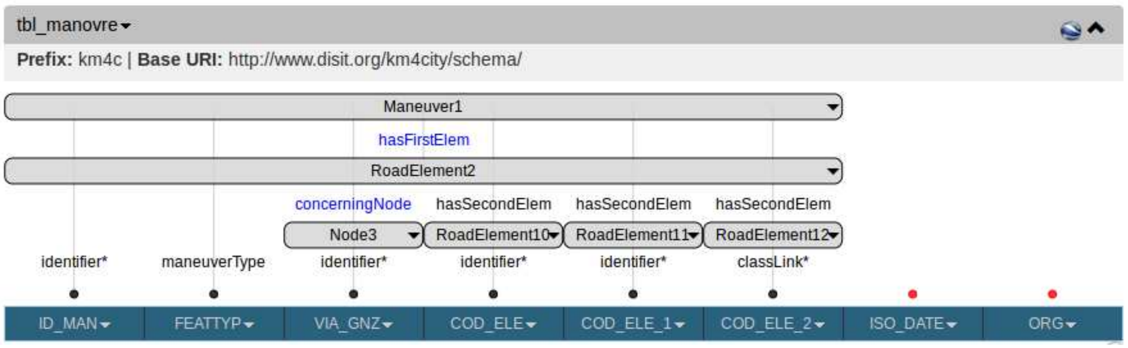
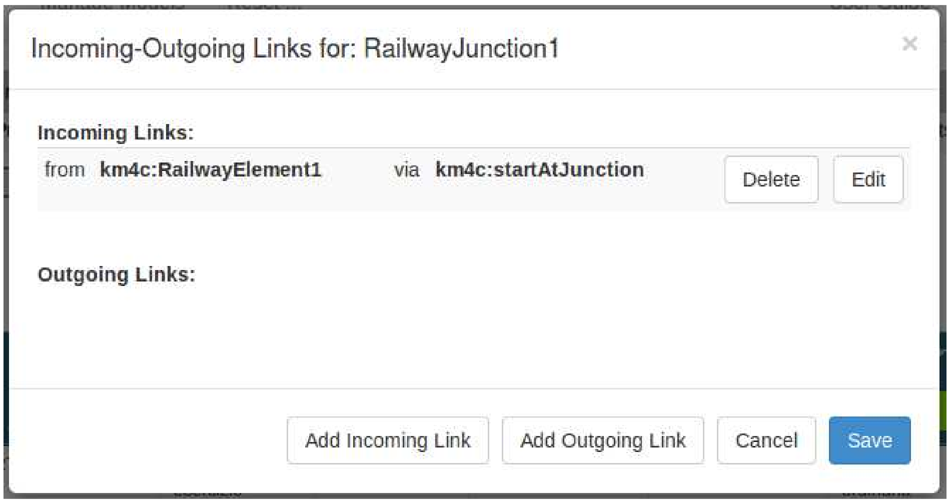


Figure 71 - Maneuver mapping example

The mapping must be done for each column that contains data relevant to the triples creation. Once the mapping has been carried out on Classes and DataProperties, the ObjectProperties, inserted in an automatic way by *Karma* according to ontologies preloaded, must be checked: unfortunately sometimes the automatic properties inserted are not correct and, in these cases, the input/output links from each class, must be changed, clicking on the bubble relative to the class from which the link arrive/start.
Regarding the example of the maneuvers mapping, the *MySQL* column *VIA_GNZ*, contains the identification code of the junction to which the maneuver is applied: therefore it corresponds to the *dct:identifier* of the *Node* class. Assuming that the ObjectProperty created between classes *Node* and *Maneuver* is not the correct one, to simply change it, open the dialog box relative to the incoming or outgoing connections, selecting one of the two classes, as shown in Figure 72, from which the ObjectProperty can be changed.


Incoming-Outgoing Links for: RailwayJunction1

Incoming Links:

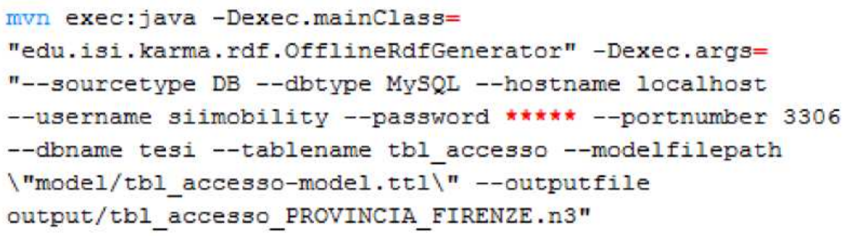
from **km4c:RailwayElement1** via **km4c:startAtJunction**

Outgoing Links:

Figure 72 - How to modify in/out links with Karma

After the manual mapping of all the columns in the imported table, *Karma* allows exporting data directly in RDF, or to publish the *R2RML* model, which can be used for

the triples generation also through the offline version of the *Karma* tool, that work in batch mode.
Once published the *R2RML* model, the triples generation process can be automated thanks to *Karma Offline* tool and an ETL job. *Karma* provides a command line scripts for the offline generation of RDF triples starting from a relational database, such as the one shown in Figure 73:



```
mvn exec:java -Dexec.mainClass=
"edu.isi.karma.rdf.OfflineRdfGenerator" -Dexec.args=
"--sourcetype DB --dbtype MySQL --hostname localhost
--username siimobility --password ***** --portnumber 3306
--dbname tesi --tablename tbl_accesso --modelfilepath
\"model/tbl_accesso-model.ttl\" --outputfile
output/tbl_accesso_PROVINCIA_FIRENZE.n3"
```

Figure 73 - Script to run Karma Offline

Thanks to this *Maven* (http://maven.apache.org/) command, and to the specified parameters after the special word *–Dexec.arg*, triples are created. Parameters that must been specified inside the *Maven* command, are the following:

- sourcetype: used to specified the data source that can be a database table (DB), a CSV file, JSON file o XML file.
- dbtype: if the sourcetype parameter is equal to DB, this value must be setting because it specified the database type to which Karma will be connected, that is MySQL, Oracle, SQLServer, etc.
- hostname: it contains the address to which the database can be accessed;
- dbpot: this field contains the port number used to establish the database connection.
- username and password: this fields contain login credentials to the database.
- tablename: it is the name of the table in which the data, that must be mapping in triples, are stored;
- modelfilepath: as the name suggest, this field contains the path to address the model, previously realized with the data integration tool.
- outputfilepath: this field allows to specify where the triples file, will be created, and its name.

After invoking the *Maven* command, *Karma* begins to generate triples, according to the rules contained in the *R2RML* model; the generated triples can be finally saved in various formats, such as N3, TTL, etc.
Regarding the project presented in this thesis, this procedure must to be repeated for each in gested dataset.
To use the huge amount of RDF triples produced from the conversion of Road Graph data, Open Data and Real Time data, a RDF framework that allows continuous saving data, while maintaining scalability and usability, has been necessary, which also allows to analyze and query the archived data, quickly and reliably.

As already mentioned earlier, the project relies to *OpenRDF Sesame* [SESAME] joined to *OWLIM-SE*, thanks to which a performing and scalable triplestore can be realized; *OWLIM-SE* provides all the tools needed to create, manage and query the triplestore, either through its web interface, or through the command line console. *Sesame*, instead, makes available a set of Java APIs for accessing and manipulating the RDF store, using Java applications also created by users.

In summary, this mapping process allowed the production of the knowledge base, that may present a large set of problems because of inconsistencies and incompleteness, that may be due to lack of relationships among different data sets, etc. These problems may lead to the impossibility of making deductions and reasoning on the knowledge base, and thus on reducing the effectiveness of the model constructed. These problems have to be solved by using a reconciliation phase via specific tools and the support of human supervisors, as explained in next paragraphs.

This mapping process must be repeated for each treaty dataset, and in some cases, such as the Street Guide and the Railway Graph, a model for each class defined within the dataset, must be created.

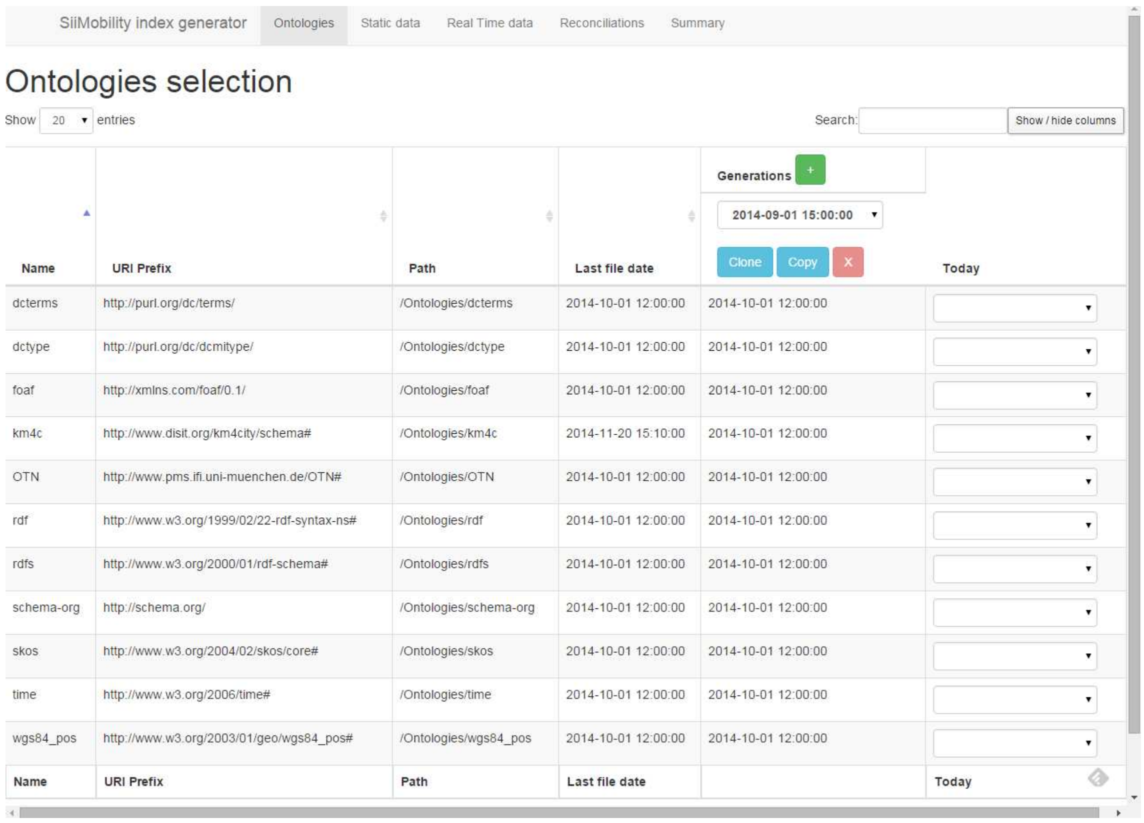
5.4 Phase IV: RDF Indexing

Storing and querying Resource Description Framework (RDF) data is one of the basic tasks within any Semantic Web application. A number of storage systems provide assistance for this task, in fact systems such as Jena2 [Wilkinson et al, 2003], Sesame [Broekstra et al., 2002], OWLIM [Bishop et al., 2011], Redland [Beckett, 2002] and others, provide a storage infrastructure for RDF data.
Additionally, each system has its own indexing schema, which can be constituted by a different number and a different type of indices. For example, *OWLIM-SE*, the system used in the project architecture to store and query data, maintains two main indices on statements for use in inference and query evaluation, i.e. the predicate-object-subject (POS) index and the predicate-subject-object (PSO) index.
OWLIM-SE also allows to define the so called "*predicate list*" index, which create a mapping from entities (subject or object) to their predicates; this type of index introduces considerable benefits, if used with certain data-sets and certain kinds of query activities, for example queries that use wild-card patterns for predicates.
Another type of index that can be created with *OWLIM-SE* is related to *Full Text Search* (FTS): this type of search concerns retrieving text documents out of a large collection by keywords or, more generally, by tokens (i.e. a sequences of characters). Formally, the query represents an unordered set of tokens and the result is a set of documents, relevant to the query. *OWLIM-SE* uses two approaches to this type of research, one proprietary called *Node-Search*, and one based on *Apache Lucene* (lucene.apache.org), called *RDF-Search*: these "*full-text indexing*" enable to perform complex queries against

character data, which significantly speeds up the query process. *OWLIM-SE* supports full text search capabilities using *Lucene* with a variety of indexing options and the ability to simultaneously use multiple, differently configured indices in the same query. *OWLIM-SE* also supports a geo-spatial index applicable to geo-spatial data that is structured according to the *WGS84* ontology (http://www.w3.org/2003/01/geo/); finally, custom indexes can be defined and used with *OWLIM-SE*.
However, most of the systems use an index structure which do not completely support typical query scenarios for data from the Web, which results in poor query answering performance, in some case.
In recent years, in fact, new types of triplestore have been developed, specifically created to improve query performance in RDF, i.e. *hexastore*, such as RDF-3X (https://code.google.com/p/rdf3x/) and H2RDF+ (https://code.google.com/p/h2rdf/) [[Papailiou et al., 2013]. This new type of triplestore enhances the vertical partitioning idea [Weiss et al., 2008], thanks to which the data graph, originally one single giant table, can be decompose into *n* two-column tables, where *n* is the number of properties. Furthermore, RDF data is indexed in six possible way, one for each possible ordering of the three RDF elements. Each instance of an RDF element is associated with two vectors; each vector gathers elements of one of the other types, along with lists of the third-type resources attached to each vector element. This is why they are called *hexastore*.
In addition to the problems of performance, triplestore also suffer from other problems related to the indices: in fact, they are often created on large amounts of data and then, notice any corrupted files, or files that contain errors, is a very difficult operation. A widely used rule to keep indices good and up to date, is to periodically re-create them.
For this purpose, within the project, it was decided to realize a tool that allows to further automate this re-indexing phase (in Figure 25 represented as Phase IV); the tool is addressed to system administrator, who, thanks to a wizard that consists of six steps, can define and run a script that allows to recreate a triplestore, identical to those available on the frontend machine, but with new indices.
The realized tool is accessible only from local network at the following address: http://192.168.0.100/indexgenerator.
The tool consists of a server-side, entirely made in *PHP* (http://php.net/) and a client side instead made using languages *HTML5*, *JavaScript* and features of the *jQuery* library (http://jquery.com/); the user interface has been created instead using the *Bootstrap* framework (http://getbootstrap.com/css/) which has significantly speeded up the whole process of development, and finally, tables have been created thanks to a *jQuery* plug-in called *DataTables* (http://www.datatables.net/).

Once logged in, the administrator can press the green button to start a new script generation process; every time a new generation is requested, a new line that uniquely identifies this new generation, is created into the *MySQL* table, located on Master

machine, and called *Generations*, which has been specially created to store all the triplestore index regeneration , which will be performed in time.
From the user-side instead, the first step of the realized interface will be displayed, dedicated to the ontologies selection: in this step, shown in Figure 74, the ontologies that will be pre-loaded on the new datastore are chosen, and for each, the version to be used must be specified; thanks to the tool, is also possible to retrieve an old indexing configuration previously used, choosing the date of the old generation and selecting the special *Copy* button.
The list of ontologies with the respective versions available, are stored in a *MySQL* table, called *Ontologies*, where also other information displayed by the tool are stored.



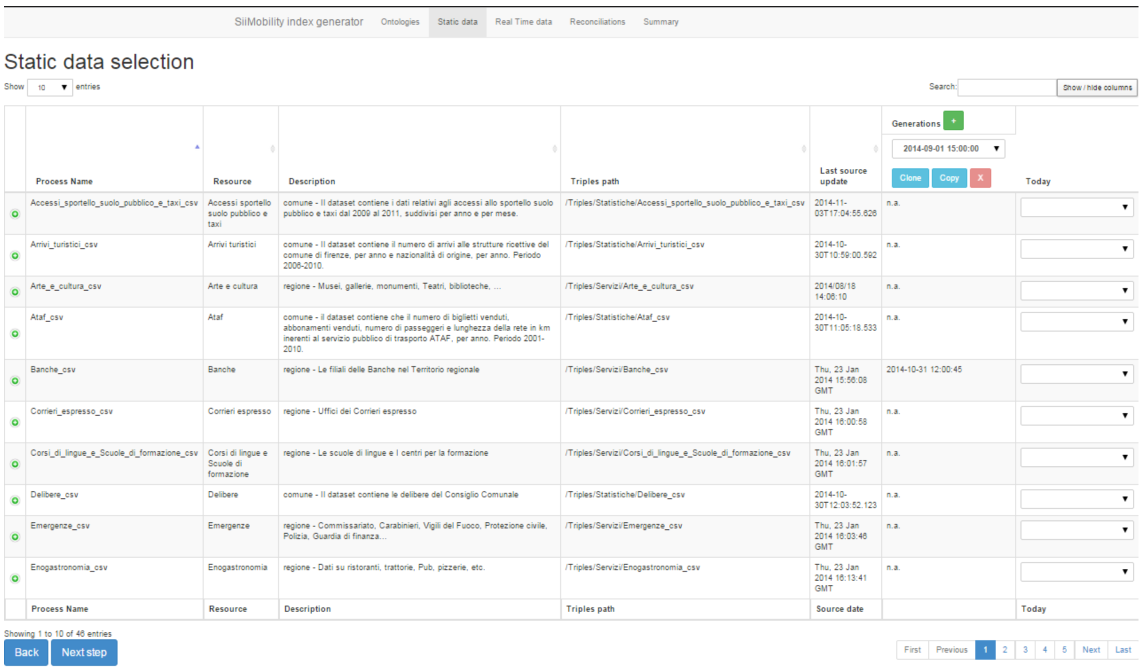
Ontologies selection

Name	URI Prefix	Path	Last file date	Generations 2014-09-01 15:00:00	Today
dcterms	http://purl.org/dc/terms/	/Ontologies/dcterms	2014-10-01 12:00:00	2014-10-01 12:00:00	
dctype	http://purl.org/dc/dcmitype/	/Ontologies/dctype	2014-10-01 12:00:00	2014-10-01 12:00:00	
foaf	http://xmlns.com/foaf/0.1/	/Ontologies/foaf	2014-10-01 12:00:00	2014-10-01 12:00:00	
km4c	http://www.disit.org/km4city/schema#	/Ontologies/km4c	2014-11-20 15:10:00	2014-10-01 12:00:00	
OTN	http://www.pms.ifi.uni-muenchen.de/OTN#	/Ontologies/OTN	2014-10-01 12:00:00	2014-10-01 12:00:00	
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	/Ontologies/rdf	2014-10-01 12:00:00	2014-10-01 12:00:00	
rdfs	http://www.w3.org/2000/01/rdf-schema#	/Ontologies/rdfs	2014-10-01 12:00:00	2014-10-01 12:00:00	
schema-org	http://schema.org/	/Ontologies/schema-org	2014-10-01 12:00:00	2014-10-01 12:00:00	
skos	http://www.w3.org/2004/02/skos/core#	/Ontologies/skos	2014-10-01 12:00:00	2014-10-01 12:00:00	
time	http://www.w3.org/2006/time#	/Ontologies/time	2014-10-01 12:00:00	2014-10-01 12:00:00	
wgs84_pos	http://www.w3.org/2003/01/geo/wgs84_pos#	/Ontologies/wgs84_pos	2014-10-01 12:00:00	2014-10-01 12:00:00	
Name	URI Prefix	Path	Last file date		Today

Figure 74 - Indexing Tool, Ontologies step

After the selection of ontologies there will be used for the new indexing, the choices made are stored within the *MySQL* table *Ontologies_Generations*, also used to be able to view the ontologies that have been selected, in each previous generations completed. In practice, the client side of the tool, with which the administrator interacts, sends an *Ajax* call to the server, which provides to storage all information relating to ontologies that have been used for the generation of the current script; a historian of the choices is thus created, containing all the ontologies and their versions used in each script indexing generation.

The second step concerns the choice of static dataset that will be part of the new triplestore, as shown in Figure 75.



Static data selection

Show 10 ▼ entries

Search: Show / hide columns

	Process Name	Resource	Description	Triples path	Last source update	Generations + 2014-09-01 15:00:00 ▼ Clone Copy X	Today
⊕	Accessi_sportello_suolo_pubblico_e_taxi_csv	Accessi sportello suolo pubblico e taxi	comune - Il dataset contiene i dati relativi agli accessi allo sportello suolo pubblico e taxi dal 2009 al 2011, suddivisi per anno e per mese.	/Triples/Statistiche/Accessi_sportello_suolo_pubblico_e_taxi_csv	2014-11-03T17:04:55.626	n.a.	▼
⊕	Arrivi_turistici_csv	Arrivi turistici	comune - Il dataset contiene il numero di arrivi alle strutture ricettive del comune di firenze, per anno e nazionalità di origine, per anno. Periodo 2006-2010.	/Triples/Statistiche/Arrivi_turistici_csv	2014-10-30T10:59:00.592	n.a.	▼
⊕	Arte_e_cultura_csv	Arte e cultura	regione - Musei, gallerie, monumenti, Teatri, biblioteche, ...	/Triples/Servizi/Arte_e_cultura_csv	2014/08/18 14:06:10	n.a.	▼
⊕	Ataf_csv	Ataf	comune - il dataset contiene che il numero di biglietti venduti, abbonamenti venduti, numero di passeggeri e lunghezza della rete in km inerenti al servizio pubblico di trasporto ATAF, per anno. Periodo 2001-2010.	/Triples/Statistiche/Ataf_csv	2014-10-30T11:05:18.533	n.a.	▼
⊕	Banche_csv	Banche	regione - Le filiali delle Banche nel Territorio regionale	/Triples/Servizi/Banche_csv	Thu, 23 Jan 2014 15:56:08 GMT	2014-10-31 12:00:45	▼
⊕	Corrieri_espresso_csv	Corrieri espresso	regione - Uffici dei Corrieri espresso	/Triples/Servizi/Corrieri_espresso_csv	Thu, 23 Jan 2014 16:00:58 GMT	n.a.	▼
⊕	Corsi_di_lingue_e_Scuole_di_formazione_csv	Corsi di lingue e Scuole di formazione	regione - Le scuole di lingue e I centri per la formazione	/Triples/Servizi/Corsi_di_lingue_e_Scuole_di_formazione_csv	Thu, 23 Jan 2014 16:01:57 GMT	n.a.	▼
⊕	Delibere_csv	Delibere	comune - Il dataset contiene le delibere del Consiglio Comunale	/Triples/Statistiche/Delibere_csv	2014-10-30T12:03:52.123	n.a.	▼
⊕	Emergenze_csv	Emergenze	regione - Commissariato, Carabinieri, Vigili del Fuoco, Protezione civile, Polizia, Guardia di finanza...	/Triples/Servizi/Emergenze_csv	Thu, 23 Jan 2014 16:03:46 GMT	n.a.	▼
⊕	Enogastronomia_csv	Enogastronomia	regione - Dati su ristoranti, trattorie, Pub, pizzerie, etc.	/Triples/Servizi/Enogastronomia_csv	Thu, 23 Jan 2014 16:13:41 GMT	n.a.	▼
	Process Name	Resource	Description	Triples path	Source date		Today

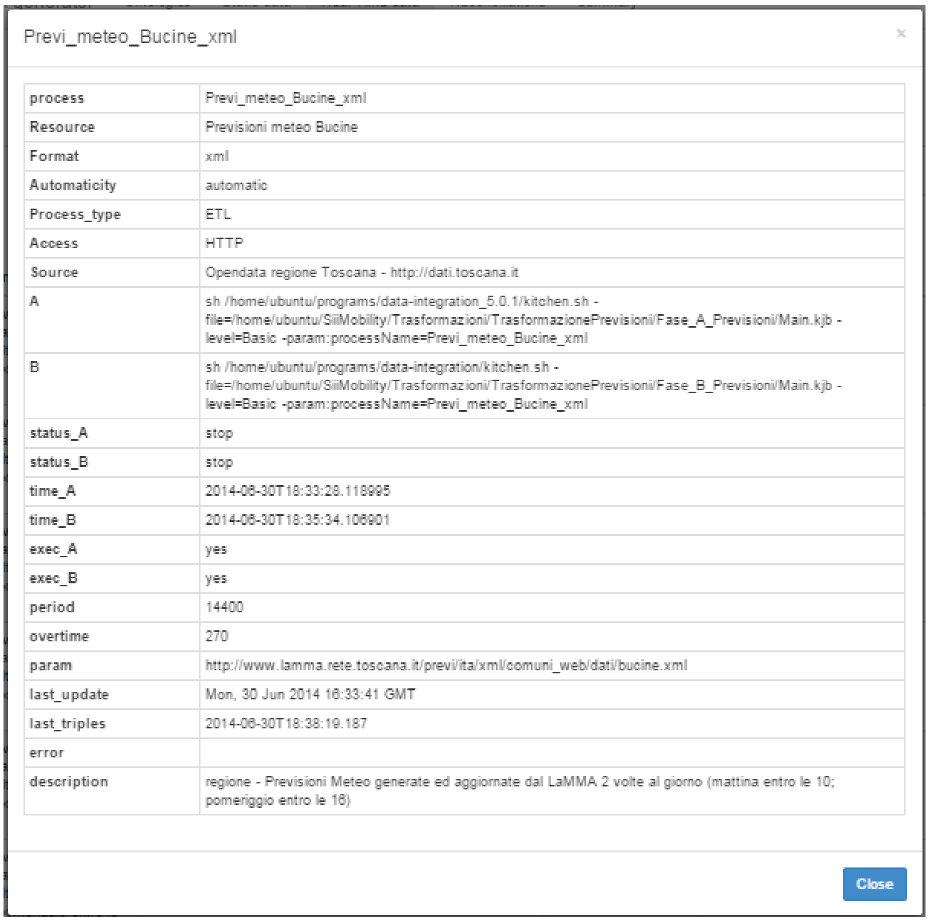
Showing 1 to 10 of 46 entries

Figure 75 - Indexing Tool, Static Data step

The information contained in the table shown by the tool, are retrieved from the *MySQL* table *processManager*; this table contains all information about each ingested dataset, thanks to the realized architecture.

Considering that information collected and stored for each dataset are several, to avoid that the table becomes unreadable, it was decided to collect part of information in a separate pop-up, which can be opened by clicking the green button, located at the beginning of each table row, shown by the indexing tools (see Figure 75); in Figure 76 the tab for a dataset can be observed.

Similar to the ontologies case, also this step uses a *MySQL* table called *OpenData_Generations* where, thanks to an *Ajax* call to the server, for each generation process, the open data sets chosen can be stored, which will be loaded into the new repository, thanks to the script that will be created at the end of the indexing process.



Previ_meteo_Bucine_xml

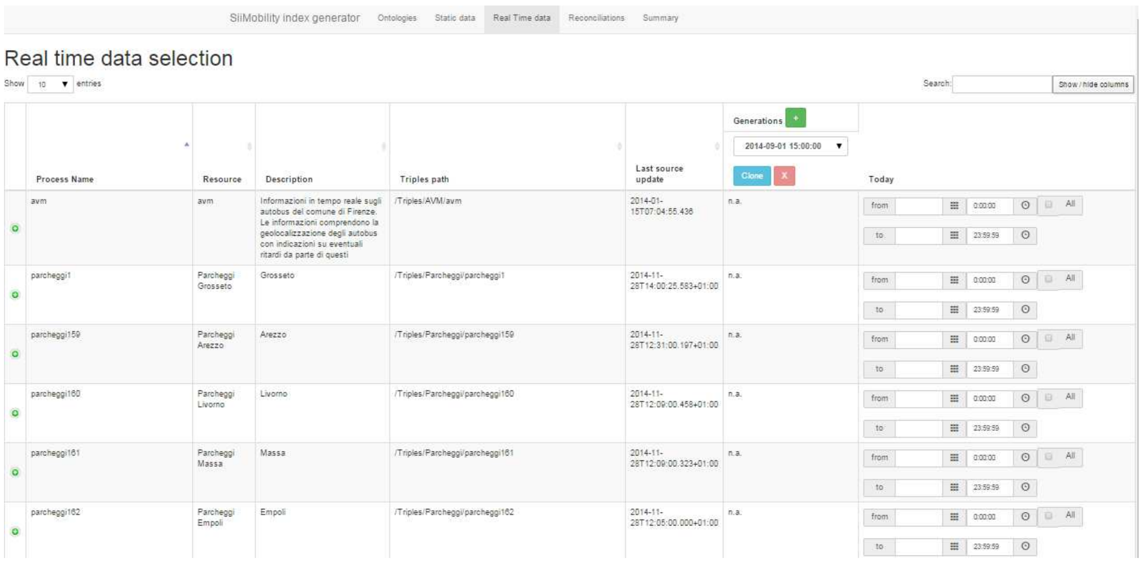
process	Previ_meteo_Bucine_xml
Resource	Previsioni meteo Bucine
Format	xml
Automaticity	automatic
Process_type	ETL
Access	HTTP
Source	Opendata regione Toscana - http://dati.toscana.it
A	sh /home/ubuntu/programs/data-integration_5.0.1/kitchen.sh -file=/home/ubuntu/SiiMobility/Trasformazioni/TrasformazionePrevisioni/Fase_A_Previsioni/Main.kjb -level=Basic -param:processName=Previ_meteo_Bucine_xml
B	sh /home/ubuntu/programs/data-integration/kitchen.sh -file=/home/ubuntu/SiiMobility/Trasformazioni/TrasformazionePrevisioni/Fase_B_Previsioni/Main.kjb -level=Basic -param:processName=Previ_meteo_Bucine_xml
status_A	stop
status_B	stop
time_A	2014-06-30T18:33:28.118995
time_B	2014-06-30T18:35:34.106901
exec_A	yes
exec_B	yes
period	14400
overtime	270
param	http://www.lamma.rete.toscana.it/previ/ita/xml/comuni_web/dati/bucine.xml
last_update	Mon, 30 Jun 2014 16:33:41 GMT
last_triples	2014-06-30T18:38:19.187
error	
description	regione - Previsioni Meteo generate ed aggiornate dal LaMMA 2 volte al giorno (mattina entro le 10; pomeriggio entro le 16)

Figure 76 - Indexing Tool, dataset inforation tab

Even in this case, the tool allows to copy a configuration previously used or to create a new one, first selecting the dataset that needs to be inserted and then the desired version of the triple, previously generated. In fact, it is possible that datasets are subjected to different mapping, for example due to the *Km4City* ontology, which is still being expanded, so may need to access to triples mapped to a specific version of this ontology.

The next step concerns Real Time data. This interface maintains a structure very similar to the previous dedicated to static data (Figure 77), except that in this case, for each type of data, should be shown the period of interest to which the triples must belong.

It is clear that every day, triples of these Real Time datasets are created, and then, during the re-indexing process, is useful to indicate if all triples of a specific type, collected over time, must be loaded, or if triples must be limited to a well defined period of time.



Real time data selection

Show 10 entries

Search: Show / hide columns

	Process Name	Resource	Description	Triples path	Last source update	Generations + 2014-09-01 15:00:00 Clone X	Today
⊕	avm	avm	Informazioni in tempo reale sugli autobus del comune di Firenze. Le informazioni comprendono la geolocalizzazione degli autobus con indicazioni su eventuali ritardi da parte di questi	/Triples/AVM/avm	2014-01-15T07:04:55.436	n.a.	from 0:00:00 All / to 23:59:59
⊕	parcheggi1	Parcheggi Grosseto	Grosseto	/Triples/Parcheggi/parcheggi1	2014-11-28T14:00:25.583+01:00	n.a.	from 0:00:00 All / to 23:59:59
⊕	parcheggi159	Parcheggi Arezzo	Arezzo	/Triples/Parcheggi/parcheggi159	2014-11-28T12:31:00.197+01:00	n.a.	from 0:00:00 All / to 23:59:59
⊕	parcheggi160	Parcheggi Livorno	Livorno	/Triples/Parcheggi/parcheggi160	2014-11-28T12:09:00.458+01:00	n.a.	from 0:00:00 All / to 23:59:59
⊕	parcheggi161	Parcheggi Massa	Massa	/Triples/Parcheggi/parcheggi161	2014-11-28T12:09:00.323+01:00	n.a.	from 0:00:00 All / to 23:59:59
⊕	parcheggi162	Parcheggi Empoli	Empoli	/Triples/Parcheggi/parcheggi162	2014-11-28T12:05:00.000+01:00	n.a.	from 0:00:00 All / to 23:59:59

Figure 77 - Indexing Tool, Real Time Data step

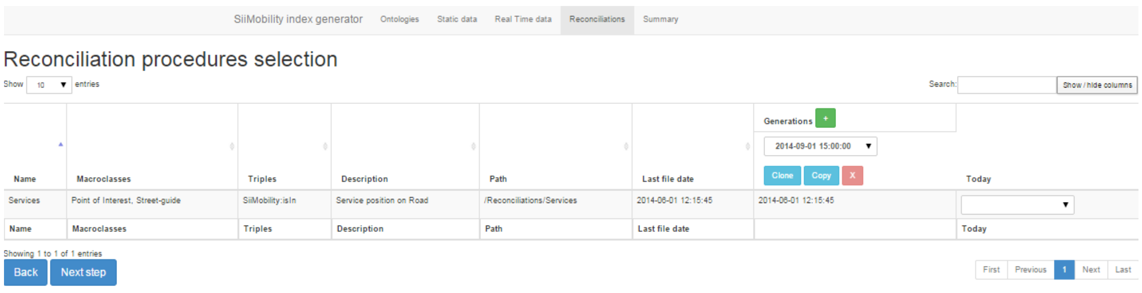
This possibility of limiting Real Time data assumes an important meaning considering the number of triple monthly created relative to only Real Time datasets, which currently manages the architecture: it was calculated, simulating an average operation situation of each sensors type (more details on how the number of triples has been calculated, are given in Chapter 6) the monthly amount of Real Time triples is about 20 million.

Such a large number of triples monthly added, would lead the repository, in a short time, to a situation of excessive size and its management would become too complex. In addition, Real Time triples are mainly used in the early days after their creation and then, the realized indexing tool, can be exploited to maintain lighter the triplestore, for example by running a monthly basis reconstruction of the repository, in which the Real Time triples history is reduced to only one week or less, that is the most recent, in preparation for the new triples that directly arrive by Real Time sensors, as soon as the repository will become the new Front End.

Another case in which the use of a tool like this, is of paramount importance, for example, is closely related to the possible detection of errors within the triples files, which are due either to a failure of the mapping process, or by an error in the mapping model, initially not identified, or for incorrect data sending from a sensor that has failed. Thanks to the Indexing tool, it is also possible to prevent that wrong triples, created starting from by the scenarios proposed above, are again put back into the datastore.

Coming back to the description of the tool, the fourth step concerns loading triples generated during the reconciliation phase: files containing all reconciliating triples are, in fact, stored in a datastore (called RDF store in Figure 78) and can be uploaded to a new triplestore.

It is evident, also in this case, the usefulness of choosing the version of triples to load, for each type of reconciled data, seen that there may be multiple file versions, for example related to an update version of the dataset, or to a new reconciliation technique applied to the same dataset.



Reconciliation procedures selection

Show 10 entries   Search: Show / hide columns

Name	Macroclasses	Triples	Description	Path	Last file date	Generations 2014-09-01 15:00:00 Clone Copy X	Today
Services	Point of Interest, Street-guide	SiiMobility:isIn	Service position on Road	/Reconciliations/Services	2014-06-01 12:15:45	2014-06-01 12:15:45	
Name	Macroclasses	Triples	Description	Path	Last file date		Today

Showing 1 to 1 of 1 entries

Figure 78 - Indexing Tool, Reconciliation step

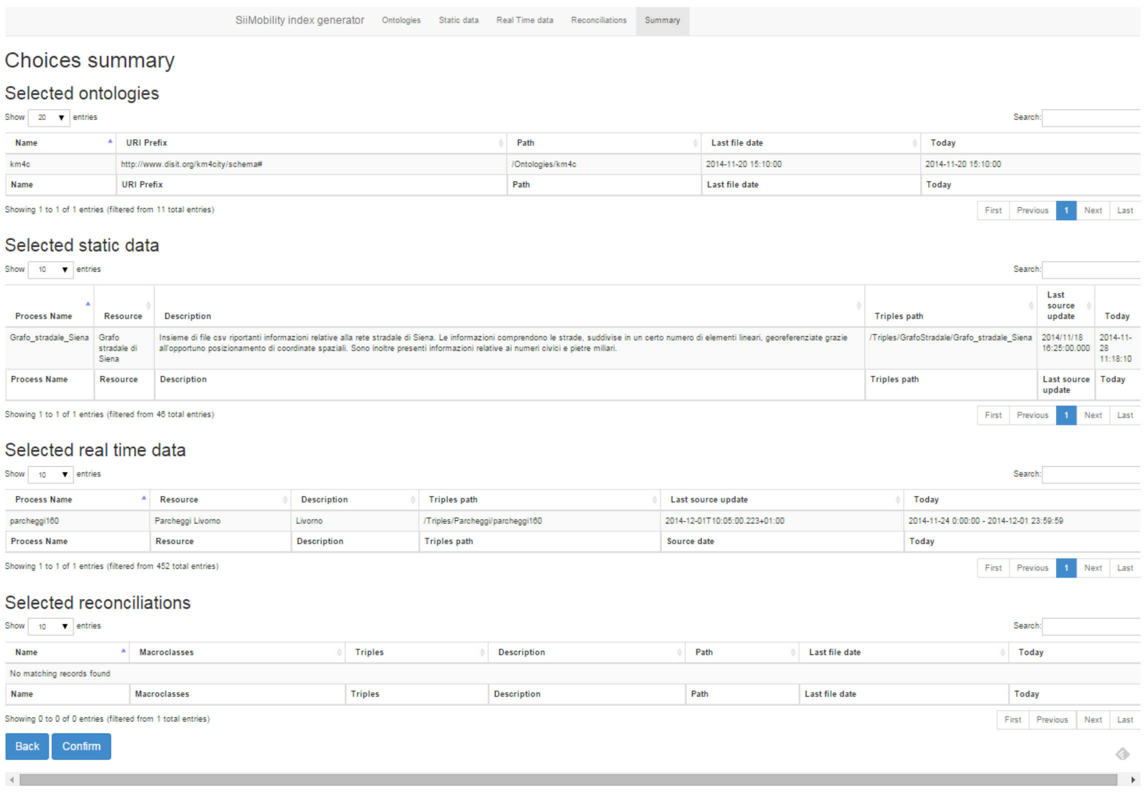
As in the previous phases, the copy of an old configuration used before, can be obtained, in order to facilitate the form compilation, by simply choosing the old configuration and clicking the button *Copy* (see Figure 78).

Similarly to the first step dedicated to Ontologies, this fourth step reads the information in the *MySQL* table *Reconciliations*, to recreate the list of reconciliation procedures for which triples to load, are available; to show the history of the choices made, the tool retrieves data within an other *MySQL* table, called *Reconciliatons_Generations*, in which reconciliation triples files loaded, for each generation completed,were stored, (thanks to an *Ajax* call to the Server side of the tool), relating to the data reconciled to load.

After all four selection steps have been completed, the tool displays a summary of the choices made during the creation process; thanks to this summary is possible to check and get back to the individual steps, to make some changes.
Once verified the correctness of the choices made, after pressing the confirmation button (Figure 79), the script generation begins, and it will be used to build the new datastore, according to the choices made. The script produced is locally stored, on the machine from which the tool is executed by the administrator.

At the end of selections, when the user presses the *Confirm* button, a read is performed on various *MySQL* tables that the tool uses to store the choices of each individual steps, and, based on the selections, the script is created as follows: the script consists of five parts, one to set parameters necessary for creating the repository via *OpenRDF*, and then there is a section devoted to each steps, that is one for ontologies, one for static data, one for Real Time, and the last dedicated to the reconciliations.

More in depth, initially some parameters valid throughout the script, are set, followed by a blank space for any manual changes that the administrator may have the need to add, before running the script.
The parts devoted to ontologies, to static data and to reconciliations are very similar, change the things that need to be changed: data is initially declared, specifying name, category and version choice; then a *for* loop, for each type of data, is defined, that invokes a script, *example.sh*, which provides to loading triples inside the indicated triplestore.
The only different part is linked to the Real Time data, for which in fact, data is declared, specifying name, category, start date and end date of the selected period; so, thanks to a *for* loop clearly more complex than the previous ones, the directory tree is navigated and, for each triples file identified, a check on its version is made, to verify if it is contained into the time interval chosen. If this occurs, the script *example.sh* is invoked.



Choices summary

Selected ontologies

Show 20 entries Search:

Name	URI Prefix	Path	Last file date	Today
km4c	http://www.disit.org/km4city/schema#	/Ontologies/km4c	2014-11-20 15:10:00	2014-11-20 15:10:00
Name	URI Prefix	Path	Last file date	Today

Showing 1 to 1 of 1 entries (filtered from 11 total entries)

First Previous 1 Next Last

Selected static data

Show 10 entries Search:

Process Name	Resource	Description	Triples path	Last source update	Today
Grafo_stradale_Siena	Grafo stradale di Siena	Insieme di file csv riportanti informazioni relative alla rete stradale di Siena. Le informazioni comprendono le strade, suddivise in un certo numero di elementi lineari, georeferenziate grazie all'opportuno posizionamento di coordinate spaziali. Sono inoltre presenti informazioni relative ai numeri civici e pietre miliari.	/Triples/GrafoStradale/Grafo_stradale_Siena	2014/11/18 16:25:00.000	2014-11-28 11:18:10
Process Name	Resource	Description	Triples path	Last source update	Today

Showing 1 to 1 of 1 entries (filtered from 46 total entries)

First Previous 1 Next Last

Selected real time data

Show 10 entries Search:

Process Name	Resource	Description	Triples path	Last source update	Today
parcheggi160	Parcheggi Livorno	Livorno	/Triples/Parcheggi/parcheggi160	2014-12-01T10:05:00.223+01:00	2014-11-24 0:00:00 - 2014-12-01 23:59:59
Process Name	Resource	Description	Triples path	Source date	Today

Showing 1 to 1 of 1 entries (filtered from 452 total entries)

First Previous 1 Next Last

Selected reconciliations

Show 10 entries Search:

Name	Macroclasses	Triples	Description	Path	Last file date	Today
No matching records found						
Name	Macroclasses	Triples	Description	Path	Last file date	Today

Showing 0 to 0 of 0 entries (filtered from 1 total entries)

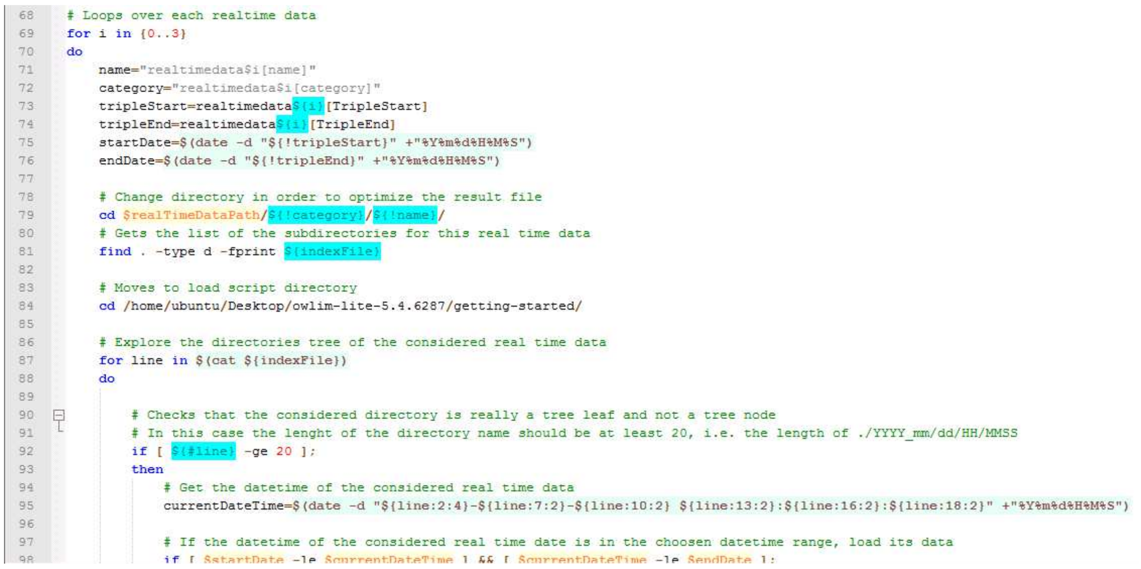
First Previous Next Last

Back Confirm

Figure 79 - Indexing Tool, Summary step

The user interface, after the script generation, shows a message that confirms the proper creation and allows the administrator to run the generated script, for really creating a new physical datastore.

In Figure 80, a portion of the script generated by the tool, in *Bash* language, can be analyzed.



```
# Loops over each realtime data
for i in {0..3}
do
    name="realtimedata$i[name]"
    category="realtimedata$i[category]"
    tripleStart=realtimedata${i}[TripleStart]
    tripleEnd=realtimedata${i}[TripleEnd]
    startDate=$(date -d "${!tripleStart}" +"%Y%m%d%H%M%S")
    endDate=$(date -d "${!tripleEnd}" +"%Y%m%d%H%M%S")

    # Change directory in order to optimize the result file
    cd $realTimeDataPath/${!category}/${!name}/
    # Gets the list of the subdirectories for this real time data
    find . -type d -fprint ${indexFile}

    # Moves to load script directory
    cd /home/ubuntu/Desktop/owlim-lite-5.4.6287/getting-started/

    # Explore the directories tree of the considered real time data
    for line in $(cat ${indexFile})
    do

        # Checks that the considered directory is really a tree leaf and not a tree node
        # In this case the lenght of the directory name should be at least 20, i.e. the length of ./YYYY_mm/dd/HH/MMSS
        if [ ${#line} -ge 20 ];
        then
            # Get the datetime of the considered real time data
            currentDateTime=$(date -d "${line:2:4}-${line:7:2}-${line:10:2} ${line:13:2}:${line:16:2}:${line:18:2}" +"%Y%m%d%H%M%S")

            # If the datetime of the considered real time date is in the choosen datetime range, load its data
            if [ $startDate -le $currentDateTime ] && [ $currentDateTime -le $endDate ];
```

Figure 80 - Script generated with the Indexing Tool

Some limitations to the indexing tool, have also been imposed, to make it more reliable, such as that related to the execution of the script: in fact, in case a script to re-generating the triplestore is already running (operation detectable by the presence of *lock* file), the indexing tool in its first page informs the administrator such a situation, preventing it from proceeding to a new generation; only when the script is finished, a new script will be generated.

5.5 Phase V: Data Reconciliation

After being loading and indexing into the RDF store, a dataset may be connected with the other, if their entities refer to the same triples; in fact missed connections strongly limit the usage of the knowledge base, for example, *Point of interest* macroclass is not connected with the *Street Guide and Rail Network* macroclass, inside the realized project.
The term reconciliation refers, in fact, to the process of verification and link RDF data that represent the same object on two different dataset, but which are not connected, due to some inconsistencies in their representation.
To connect services to the *Street Guide and Rail Network* macroclass in the project repository, a reconciliation phase in more steps, has been developed, because the notation used by the Tuscany region in some Open Data, within the *Street Guide*, does not always coincide with those used inside Open Data, relating to different points of

interest. In substance, different public administration are publishing Open Data that are not semantically interoperable.
A relevant process of data improvement for semantic interoperability is related to the application of reconciliations among the entities associated with locations as streets, civic numbers and localities. On this regard, there are different types of inconsistencies within the various integrated dataset, such as:

- typos;
- missing house number, or replacement with "0" or "SNC" (Italian acronym that means without civic number);
- Municipalities with no official name (e.g. Vicchio/Vicchio del Mugello);
- street names with uncommon characters (-, /, ° ? , Ang., ,);
- house numbers with strange characters (-, /, ° ?, Ang. ,(,);
- road name with words in a different order from the usual (e.g. Via Petrarca Francesco, exchange of name and surname);
- number wrongly written (e.g. 34/AB, 403D, 36INT.1);
- red street numbers (in some cities, street numbers may have a color. So that a street may have 4/Black and 4/Red, red is the numbering system for shops);Roman numerals in the street name (e.g., via Papa Giovanni XXIII).

Thanks to the created ontology, is possible to perform a services reconciliation at street number level, e.g. connecting an instance of class *Service* to an external access (*Entry* class of *Km4city* ontology) that uniquely identifies a house number on a road, or at street-level, with less precision (lack that can be compensated thanks to the services geolocation).
The reconciliation process can be performed with the aim of finding elements that identify the same entity, while presenting different URIs. Thus the identified reconciliations are solved creating an *owl:sameAs* triple to the selected location toponym. Reconciliation detection can be performed by using (i) a set of specific SPARQL queries, (ii) or program tools for RDF link discovering. To this end, declarative languages for link discovering such as SILK [Isele and Bizer, 2013] and LIMES [Ngomo and Auer, 2011] have been proposed. As the production of SPARQL queries, the programming of the link discovering algorithms also implies the knowledge of the ontological structure of the RDF stores to be compared/linked.
For the project a comparison between this two different reconciliation approaches, has been performed, applied initially to the services-roads reconciliation, to determine if the two methods are replaceable.

5.5.1 *SPARQL Reconciliation*

The methodology used for SPARQL reconciliation tries to connect each service first at house number-level, and then, at street-level, and it consists of more reconciliation step performed:

- The first step consists of an exact search of the street name associated with each service integrated.
- The second reconciliation step is based on the last word research inside the field *schema:streetAddress*, of each instance of the *km4c:Service* class, because, statistically, for a high percentage of street names, this word is the key to uniquely identify a match.
- The third reconciliation step involves *Google Geocoding* tool and *OpenStreetMap* tool.
- The above mentioned three steps have been also carried out without taking into account the house number, and so in order to obtain a reconciliation at street-level, of each individual service.

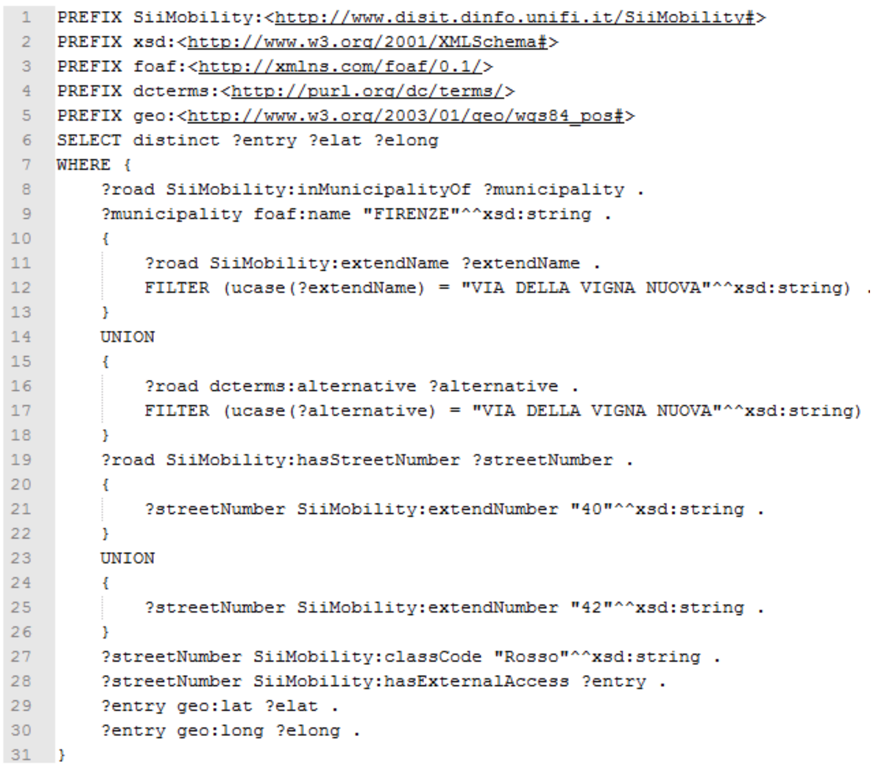
Below, the various steps will be described in more detail.
Because of the inhomogeneity in the addresses, writing inside files coming from different sources, and to help the next reconciliation steps, it was decided to apply an additional step of reconciliation to the names contained in the *Street Guide*.
There is a whole literature devoted at address matching that offers multiple solutions even very complex [Drummond, 1995], however, in this project, since data related to Street Guide is restricted to the Italian language, a solution that would lead to maximize the benefits in later reconciliation stages, has been applied. In fact a very frequent problem for exact search, is the existence of multiple ways to express toponym qualifiers, that is dug (e.g. *"Piazza"* and *"P.zza"*) or parts of the proper name of the street (such as *"Santa"*, or *"S."* or *"S"* or *"S.ta"*); to this end, a *MySQL* support table has been realized, in which the most frequent ambiguities are stored, as shown in Table 12.

ID	String to replace	String1	String2	String3	String4	Type
1	VIA	V.	V			strStarts
2	VIALE	V.LE	VLE			strStarts
3	PIAZZA	P.ZZA	P.ZA	PZZA	PZA	strStarts
4	PIAZZALE	P.ZZALE	PZZALE			strStarts
5	CORSO	C.SO	CSO			strStarts
6	FRATELLI	F.LLI	FLLI			contains
7	A'	À	A			contains
8	E'	È	È	E		contains
9	I'	Ì	I			contains
10	O'	Ò	O			contains
11	U'	Ù	U			contains
12	PRIMO	I	1	UNO		contains
13	II	2	DUE	SECONDO		contains
14	III	3	TRE	TERZO		contains
15	IV	4	QUATTRO	QUARTO		contains

Table 12 - dct:alternative starting table

After completing the table, all instances of the class *Road* have been researched, which contain at least one of the strings in the support table, inside DataProperty

km4c:extendedName. For all identified instances, a number of properties *dct:alternative* were created, e.g. starting from "*Via di Santa Marta*", the alternative name "*Via di S. Marta*", "*Via S. Marta*", "*V. Santa Marta* "and" *V. S. Marta*", were created.
So, whenever a query will look for the name of a *Road* class instance, the research will be extended to the alternative fields which allows greatly increase the number of matches found. In the following, the case of services reconciliation will be in fact analyzed, which uses the fields *dct:alternative* to search correspondences, that create reconciliation triple between macroclasses *Point of Interest* and *Street Guide and Rail Network*.
Coming from different sources, addresses of serviceS dataset are often written in a different way, if compared to the names associated with instances of the *Road* class.
Moreover, the services have always an address stored into some DataProperty (that is *schema:addressLocality*, *schema:streetAddress*, *schema:postalCode*, *km4c:hause Number*) and sometimes, even a pair of coordinates *geo:lat* and *geo:long*. From this known information, the reconciliation should allow the creation of a triple that connect each service to a toponym or to an house number.
As previously seen, two types of services reconciliation can be made, at street-level and at street number level, allowing respectively to create triples *km4c:isInRoad* and *km4c:hasAccess*. Given the huge amount of data it is impossible to implement a system based on a simple SPARQL query, but the *Sesame APIs* allow to build simple Java applications that perform a single query repeatedly, thanks to which all service addresses are searched one by one, among the instances of the *Road* class.
The first reconciliation step, that is the exact search, try to find an identical address to that contained in each *Service* instance, checking all the fields *km4c:extendedName* and *dct:alternative* for each *Road* class instance. Starting from information about the city name and the name of the street, a SPARQL query like the one shown in Figure 81, is performed for each service.
Lines 8-9 request all road belonging to the municipality extracted from the service address, while the *UNION* clause allows to specify, that the searched road may be contained, in both fields *km4c:extendedName* or *dct:alternative*. Line 19 extract all the *?streetNumber* belonging to that road.
The following instructions, separated again by a UNION clause, imposing that an instance of *StreetNumber* class exists and it corresponds to house number "40" or "42", with the *CodeClass* property *"Red"*. Finally, at lines 28 and 29, accesses related to filtered entities are extracted, that also correspond to the geographic coordinate *?elat* and *?elong*.



```
PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX dcterms:<http://purl.org/dc/terms/>
PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
SELECT distinct ?entry ?elat ?elong
WHERE {
    ?road SiiMobility:inMunicipalityOf ?municipality .
    ?municipality foaf:name "FIRENZE"^^xsd:string .
    {
        ?road SiiMobility:extendName ?extendName .
        FILTER (ucase(?extendName) = "VIA DELLA VIGNA NUOVA"^^xsd:string) .
    }
    UNION
    {
        ?road dcterms:alternative ?alternative .
        FILTER (ucase(?alternative) = "VIA DELLA VIGNA NUOVA"^^xsd:string)
    }
    ?road SiiMobility:hasStreetNumber ?streetNumber .
    {
        ?streetNumber SiiMobility:extendNumber "40"^^xsd:string .
    }
    UNION
    {
        ?streetNumber SiiMobility:extendNumber "42"^^xsd:string .
    }
    ?streetNumber SiiMobility:classCode "Rosso"^^xsd:string .
    ?streetNumber SiiMobility:hasExternalAccess ?entry .
    ?entry geo:lat ?elat .
    ?entry geo:long ?elong .
}
```

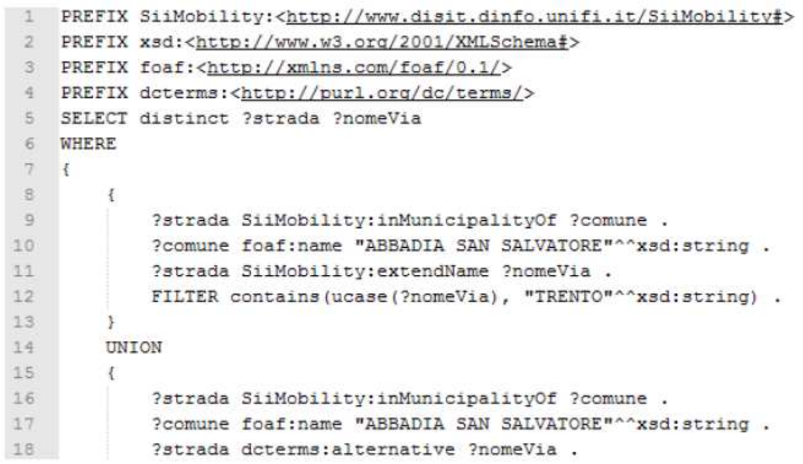
Figure 81 - SPARQL reconciliation query

Thanks to this query, if it exists, the resource of *Entry* type, corresponding to the indicated address, will be recovered; in particular, an example of query results for items of class *Entry* is shown in Table 13.

Service Address	Entry ID	Elat	Elong
VIA DELA VIGNA NUOVA 40/R - 42/R, FIRENZE	RT048017000746AC	43,7712868	11,2499669
VIA ARETINA 499, FIRENZE	RT048017006530AC	43,7663602	11,3147545
VI DI SANTA MARTA 3, FIRENZE	RT048017075274AC	43,798784	11,2552288
VIA ROMA 583, BAGNO A RIPOLI	RT048001007554AC	43,7305372	11,3613187
VIA TOSELLI 41, SIENA	RT052032017708AC	43,3165709	11,3554662
VIA FORLIVESE 64, SAN GODENZO	RT048039000281AC	43,9257775	11,6179469
VIA CUNIBERTI 12, MONTE ARGENTARIO	RT053016002929AC	42,4353327	11,1202326
VIA DEL CASTELLO 26, ISOLA DEL GIGLIO	RT053012000041AC	42,3604749	10,9185241
VIA TALENTI 36, MARRADI	RT048026002081AC	44,0753448	11,6124367
VIA STRADELLA 7, FIVIZZANO	RT045007024125AC	44,2369531	10,126546

Table 13 - Results of the previus SPARQL query

The next reconciliation step is based on the research of the last word inside the field *schema:streetAddress*, for each instance of the *Service* class, due to the reasons previously explained. Therefore, for each element contained into the not reconciled services list, all the road that contain, within its name, the last word searched, have been extracted; in Figure 82 a query used for this type of research, is shown.



```
PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX dcterms:<http://purl.org/dc/terms/>
SELECT distinct ?strada ?nomeVia
WHERE
{
    {
        ?strada SiiMobility:inMunicipalityOf ?comune .
        ?comune foaf:name "ABBADIA SAN SALVATORE"^^xsd:string .
        ?strada SiiMobility:extendName ?nomeVia .
        FILTER contains(ucase(?nomeVia), "TRENTO"^^xsd:string) .
    }
    UNION
    {
        ?strada SiiMobility:inMunicipalityOf ?comune .
        ?comune foaf:name "ABBADIA SAN SALVATORE"^^xsd:string .
        ?strada dcterms:alternative ?nomeVia .
```

Figure 82 - SPARQL query to find a specific Road in a specified Municipality

The fields extracted thanks to this query are *?Road*, which contains the unique URI of the recovered road, *?name* that instead contains the full name of the road. The query is composed by two sub-queries, both recovering the same fields, but in the first sub-query in the example, the word "*TRENTO"* is searched in the field *km4c:extendedName*, while in the second sub-query, the word "*TRENTO"* is searched within the various fields *dct:alternative*. In both cases, the results are filtered by municipality name, that in Figure 82, is *"ABBADIA SAN SALVATORE"*.
Looking at the results it was found that not all the obtained matches are accurate, in fact, often more than one match is found; for this reason a semi-automatic control of the results has been applied, in order to avoid the creation of incorrect reconciliation triples. The process carried out for services, for example, filter the results that get only one or two matches at most: in case of a single match, the triple is automatically generated, instead, if two matches are found, results were subjected to a manual control. In Table 14 a small portion of the results obtained, is shown.

Municipality	Service Address	Street Name	URI Toponym
FIRENZE	VIA BORGO DEI GRECI	VIA GRECIA	http://www.disit.org/km4city/resource/RT4801707463TO
FIRENZE	VIA BORGO DEI GRECI	BORGO DE GRECI	http://www.disit.org/km4city/resource/RT4801701830TO
FIRENZE	VIA NUOVA D CACCINI	VIA GIULIO CACCINI	http://www.disit.org/km4city/resource/RT4801703143TO
FIRENZE	VIA NUOVA D CACCINI	V NUOVA DEI CACCINI	http://www.disit.org/km4city/resource/RT4801703400TO
POPPI	VIA CASA DAMIANO	VIA CASE DAMIANO	http://www.disit.org/km4city/resource/RT5103119173TO
POPPPI	VIA CASA DAMIANO	PZA DAMIANO CHIESA	http://www.disit.org/km4city/resource/RT5103120951TO
PONTEDERA	VIA LOTTI	VIA FELICE CAVALLOTTI	http://www.disit.org/km4city/resource/RT5002901189TO
PONTEDERA	VIA LOTTI	VIA FELICE LOTTI	http://www.disit.org/km4city/resource/RT5002901191TO

Table 14 - Query reconciliation results

In the above table is possible to observe a searching example related to last word "*GRECI*", for a service located in "*VIA BORGO DEI GRECI*" in Florence: the query returns two toponyms with relative URI, i.e. "*VIA GRECIA*" and "*BORGO DE 'GRECI*": the second result has been manually selected, because it is obviously the correct one. This reconciliation step performed, has also allowed to increase the number of reconciled services at house number-level, performed immediately after. In fact, starting from the results filtered above, a search on street number was performed, limited to *km4c:StreetNumber* instances.

The next type of reconciliation performed, is based on tools like *Google* geocoding API (http://developers.google.com/maps/documentation/geocoding) and *OpenStreetMaps* Nominatim (http://wiki.openstreetmap.org/wiki/Nominatim).

The *Google* geocoding procedure allows to convert addresses (e.g. "*Via di Santa Marta 3, Florence*") in geographic coordinates (e.g. *43.7976054*, *11.253943* in WGS84). Through APIs provided by both services previously mentioned, many associations can be made between services that, in the previous reconciliation steps, have not been matched. In particular, after recovering the spatial coordinates of a service, all *Entry* class instances geographically closest, are extracted, limiting the search within a narrow radius, to reduce possible errors.

5.5.2 *Silk Reconciliation*

The second approach, defines a semi-automated method for services data reconciliation, which also allows to evaluate the results obtained from the first approach and to verify if the two approaches are both applicable, and interchangeable.

The software used to implement this second approach, is *Silk* presented in [Appendix A.8], which allows to define rules to identify possible links, between the two data sources to be compared, and to assign a score to each identified couple.

The choice of *Silk* for the evaluation of the project, come from an analysis conducted on some similar tools and on the resulting ability of the tool to carry out supervised learning. The fundamental idea of active learning in the context of entities correspondence, is to reduce the number of candidate links that need to be labeled by the user. Silk, through the use of genetic programming and active learning allows learning link rule, asking the user to confirm or reject a number of candidate links that the algorithm has automatically selected.

Link discovering based reconciliation, in fact, consists in writing specific SILK algorithms, grounded on distances and similarity metrics between patterns and relationships mainly based on string matching and distance measures (*Euclidean, weighted models, tree distances, patterns distance, string match, taxonomical, Jaro, Jaro-Winkler, Leveisthein, Dice, Jaccard, etc.*) [Isele and Bizer, 2013].

Silk specifies rules through a tree structure, composed of four elements in cascade:

- Path: element that, starting from a RDF path, retrieves entities values;

- Transformation: item that applies transformations for data normalization;
- Comparison: block to evaluate the similarity of two input, based on a user-specified distance measure;
- Aggregation: element that combines the confidence values.

As in previous section, to make possible the comparison, reconciliation is applied between services and *Street Guide*.
To perform reconciliation, both manual and semiautomatic checks, from the command line, were performed, using *Silk 2.6*.
To assess the reliability of *Silk*, in terms of *Precision* and *Recall*, two datasets services were chosen, *"Mobilità Auto"* and *"Salute e Sanità"*. Part of these two datasets and the list of toponyms with respective codes, extracted from the *Street Guide*, were placed on an *Excel* spreadsheet and, through a manual comparison, (that involves the street name where the service is located and all toponyms name from the *Street Guide*) for each service, the correct correspondence was written. In this way a *Groundtruth* was created, thanks to which the number of data reconcilable and not reconcilable, were found.
The following table shows the *Groundtruth* (GT) founded:

Dataset	Total	Reconciliable	Not Reconciliable
Mobilità auto	197	185	12
Salute e sanità	1127	1066	61

Table 15 - Groundtruth for datasets Mobilità auto and Salute e sanità

The main types of inconsistencies in the datasets used, corresponding to only a portion of those listed above, in Section 5.5, that is:

- Street names with words in a different order from the usual (e.g. *Via Petrarca Francesco*, exchange of name and surname);
- Street names with uncommon characters (-, /, ° ? , Ang., ,);
- Municipalities with no official name (e.g. *Vicchio/Vicchio del Mugello*);
- Typos;

Some tests with *Silk* were then carried out on selected datasets, and the obtained results were compared with the *GT* results. First *LSL* files were created, each of which contains a rule with different thresholds. Furthermore a script to automatically run *Silk* has been created, which processes files, and for each produces a results file, containing the N-triple associations.
Thanks to an *Excel VBA Macro* specially designed, the results were reorganized on worksheets, and the values of *True Positives, False Positives, True Negatives, False Negatives* were calculated, based on *GT*.
For each *Silk* elaboration, the calculated values are defined as follows:

- *True Positives*: services for which a correct association has been made, with the *Road* instances, that is *Silk* has reported a result and the result is correct.

- *False Positives*: services for which a wrong association has been made, with the *Road* instances, that is *Silk* has reported a wrong result.
- *False Negatives*: services for which no association has been found, with the *Road* instances, despite it was contained in the *GT*.
- *True Negatives*: services for which no association has been found, with the *Road* instances, and no association was contained into the *GT*.

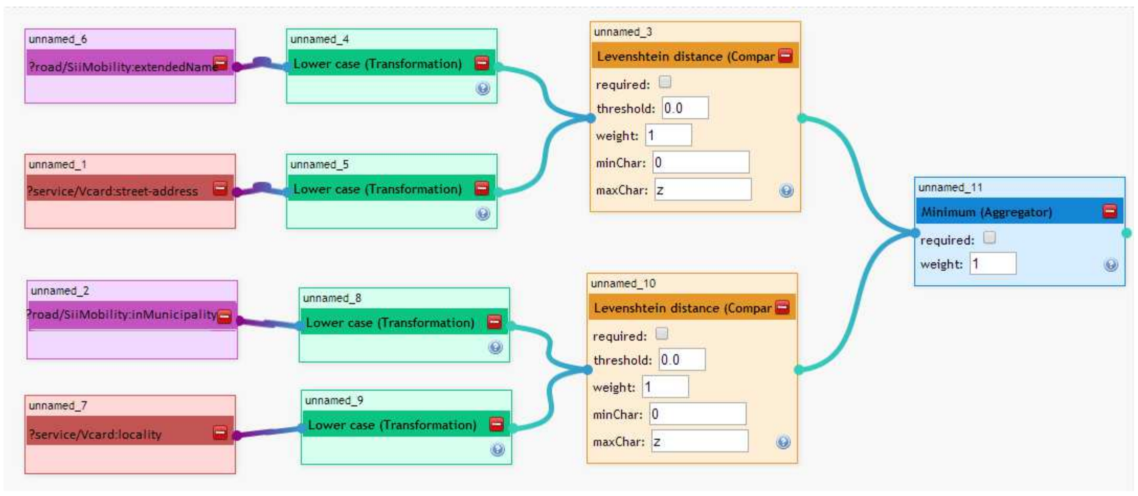

unnamed_6
?road/SiiMobility:extendedName
unnamed_4
Lower case (Transformation)
unnamed_1
?service/Vcard:street-address
unnamed_5
Lower case (Transformation)
unnamed_3
Levenshtein distance (Compar
required:
threshold: 0.0
weight: 1
minChar: 0
maxChar: z
unnamed_2
?road/SiiMobility:inMunicipality
unnamed_8
Lower case (Transformation)
unnamed_7
?service/Vcard:locality
unnamed_9
Lower case (Transformation)
unnamed_10
Levenshtein distance (Compar
required:
threshold: 0.0
weight: 1
minChar: 0
maxChar: z
unnamed_11
Minimum (Aggregator)
required:
weight: 1

Figure 83 - Connection rule for Address-City couples

The basic rule, that *Silk* uses during processing, is composed of the following comparisons:

- Address mapped on two datasets using the URI:
  <?road/km4city:extendName>
  <?service/schema:streetAddress>
- City mapped on two datasets using the URI:
  <?road/km4city:inMunicipalityOf>
  <?service/schema:addressLocality >

For the *address* attribute comparison, a lowercase transformation was performed, to make the data insensitive to the upper and lower case variation, and the following *Silk* string comparison functions have been modified: *Levenshtein*, *Jaccard*, *Dice*. In fact, for each of these, threshold for which the match is classified as correct, has been modified: *Levenshtein* threshold changes from 2 to 8 characters, *Jaccard* threshold changes from 2 to 7 characters and *Dice* threshold changes from 2 to 5 characters. These values were chosen based on the calculation method of each metric, and thanks to the detected inconsistencies knowledge, acquired during the preliminary analysis.
For the *city* comparison, the *Levenshtein* distance measure, with zero threshold, was chosen, this means that this attribute must be the same in the comparison, i.e. each pair of city names must match on each character. This very restrictive measure, was chosen

based on the *a priori* dataset knowledge: during the *GT* construction in fact, it was possible to verify that typos on city names, are negligible; errors on this specific attribute arise primarily from the exchange between the name of the locality/fraction and the municipality name, a type of error not detectable through comparison functions.
Among the aggregate functions available in *Silk*, the *minimum* function was chosen, changing the threshold from 10% to 100%: this operation also implies that the produced results, with a measure of a global similarity of 10% (in the worst case) and 100% (in the best case), are labeled as matching. A special precautions have been reserved to the house number: in fact, it is not included in all tests.
In order to better recognize tests, a special name has been assigned to each rules, symptomatic of the change from the basic rule, with the following format:

<KCXTnum_Xnum_*ThrasH* >

From this format, it is then possible to learn information about first and second transformation rules, and also on the aggregate function threshold.

- *K* indicates if the specific knowledge of the dataset has been transferred into the rule;
- *C* indicates if the house number has been removed in the rule;
- *X* is the distance measure (l = *Levenshtein*, j =*Jaro*, h =*Jaccard*);
- *T* indicates the token transformation;
- *Num* is the tolerance percentage of the distance measure;
- *ThrasH* is the percentage of correct results, according to the aggregation measure.

In the first tests, the basic rule that uses comparisons on attributes pairs, has been used, together with the *Levensthein* distance; tolerance of the comparison function and threshold of the aggregate function, have been changed by steps of 10 percentage points at a time.

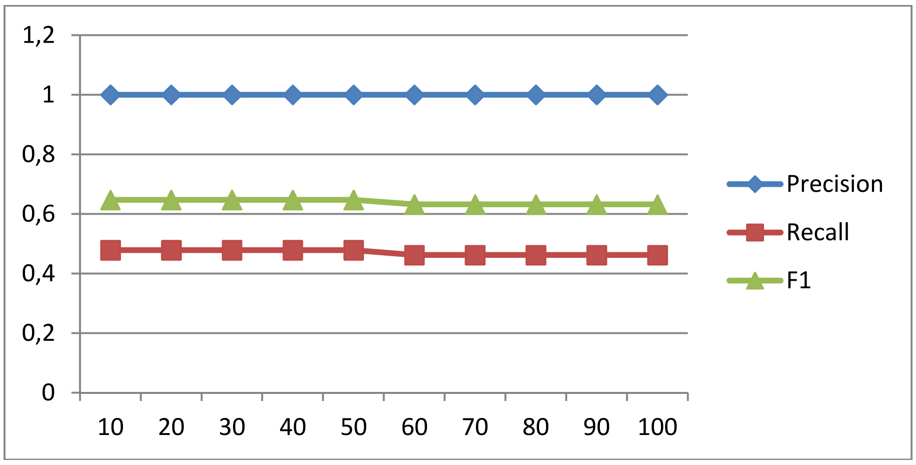


Diagram 1 - Precision, Recall, F1 of the aggregate function with a threshold of 10 to 100, Levenisthein distance of 2, rule cl20_l00_ [10-100]

In the above Diagram 1 the evolution of Precision, Recall and F1 (this latter is also called the F-measure, and it is defined as Harmonic mean of Recall and Precision) is shown, relating to the rule that uses *Levenshtein* distance = 2 for *address* comparison, and *Levenshtein* distance = 0 for *city* comparison. The aggregation function sorts data, based on the minimum score; there are no significant changes in the scores of Precision and Recall. The Precision reaches 100%, while the Recall is maintained close to 50%, that can be translated as half of the data reconcilable were recovered and those recovered are trusted.
In the following diagram it is easy to see that, with the increasing of *Levenshtein* distance, most results are recovered, because also addresses with errors, such as abbreviation, are recovered.

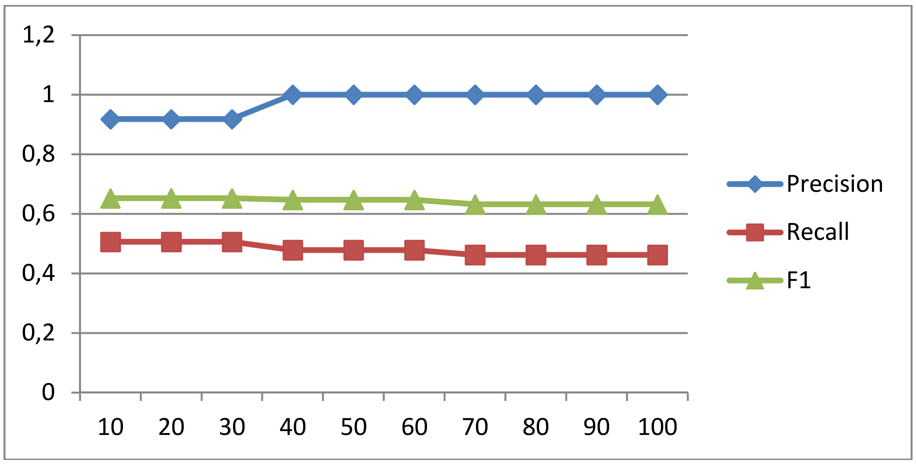


Diagram 2 - Precision, Recall, F1 varying the threshold aggregation function [10-100], Levensthein distance equal to 3; rule cl30_l00_ [10-100]

If the *Levenshtein* distance increases, an improvement in Recall and Precision can be observed, due to the increased incidence of typos. The optimum combination is the aggregate function threshold equal to 50%, with a *Levenshtein* distance equal to 5 characters, and with an aggregate function threshold equal to 70%.

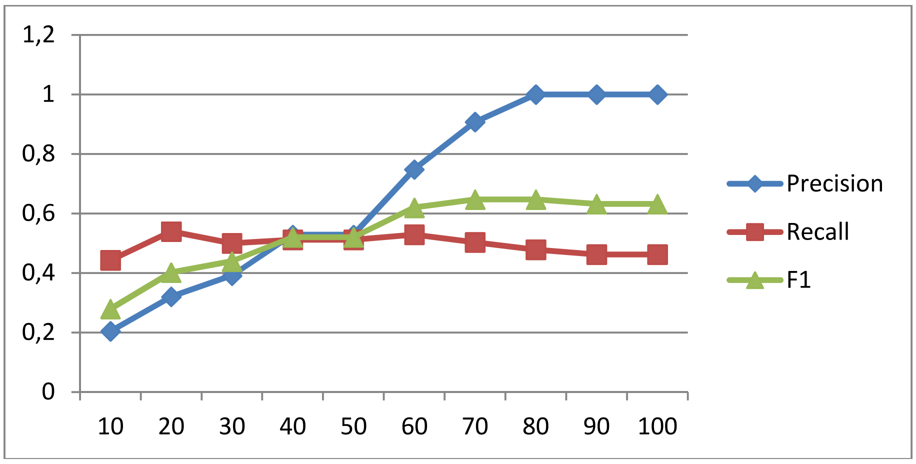


Diagram 3 - Precision, Recall, F1 varying the threshold of the aggregate function [10-100], distance Levensthein equal to 8, rule cl80_l00_ [10-100]

From Diagram 3, it is possible to observe that the rule, obtaining the highest score in terms of F1 measure (0.647), has a threshold of 70%, and it achieves a Precision value of 0.907 and a Recall value of 0.50. This is related to the *Levenshtein* distance tolerance increase, with which also the false positives number increase; so it is necessary to decrease the aggregate function tolerance, considering the results correct, only beyond a 70% threshold.

In the Table 16, the scores obtained from rules variation with a different aggregate function threshold set, are shown.

MOBILITA’ AUTO	VP	FP	VN	FN	Precision	Recall	F1
Levensthein 20	88	0	12	96	1	0,478261	0,647059
Levensthein 30	88	0	12	96	1	0,478261	0,647059
Levensthein 40	89	8	12	87	0,917526	0,505682	0,652015
Levensthein 50	90	7	12	87	0,927835	0,508475	0,656934
Levensthein 80	65	58	11	62	0,528455	0,511811	0,52

Table 16 - Precision, Recall, F1 of the rule with Leveinsthein distance variation [20-70] and an aggregate function threshold equal to 50

The rule with *Levenshtein* distance equal to 5, is the most performing in all tests.
The same tests were then carried out using a different comparison function, that is the *Dice* (d inside the special name format seen above) and the *Jaccard* (h inside the special name format seen above) distance measures have been tested, to determine the corresponding best variables combination.

It is possible to assert that to add knowledge on the dataset and write ad-hoc rules, improves the performance of Silk that achieves a Precision value of 92.59% and a Recall value of 71.43%.

It has also been possible to verify that the rule based on *Dice* distance (distance = 0.3) with a very low aggregate function threshold (20%), is the optimal combination and allows to increases performance against the rule based on *Levensthein* distance (4 percentage points on Precision and 15 percentage points on Recall). For this rule a further transformation has been applied, i.e. the *Tokenization*. This normalization function has allowed to recover errors related to the words order, for example: "*Via Giuseppe Garibaldi*" and "*Via Garibaldi Giuseppe*". This occurs because the *Dice* metric calculates a score based on pairs of elements, within the two strings to transform; this also justifies the low threshold on the aggregation function (results that meet at least 20% are considered correct). Unlike what happens for the rule based on *Levensthein* distance, for the rule based on *Dice* distance, a low threshold does not adversely affect the Recall value.

The first reconciliation process presented in this chapter, that is the SPARQL reconciliation, has been also applied to other types of data for which a reconciliation was necessary, to connect data belonging to different ontology macroclasses; these new reconciliation phase present less complexity respect to services reconciliation; for this reason they will not be discussed in depth.
The other reconciliations performed are the following:

- Traffic sensors were connected via the ObjectProperty *km4c:placedOnRoad* to the *Road* instance, representing the road where they are installed;
- Weather forecasts have been linked to the municipality to which they relate;
- Bus stops were connected to the *Road* instance where they are located thanks to coordinates that geolocated them precisely;
- Sensors installed inside car parks were associated to the *Service* instance representing the same car park;
- Resolutions were connected to the *PA* instance, which has issued them;
- Train stations were connected to the *Road* instance, in which they are localized.

5.6 Phase VI: Data Validation

A validation procedure is required at this point of the work, to test whether the ontology created, allows to model the data correctly, that is to check if data, once inserted inside the triplestore, after having been molded in accordance with the created R2RML models, are interpreted as wanted.
The validation of an ontology refers to errors and/or omissions of concepts/instances that may be contained inside it. There are many ways in which an ontology can be validated in order to improve and expand it. The most important validation, concerns the formal semantics, i.e. concerns the meaning of the constructs (classes and relationships between classes) of an ontology. Some automated tools were built to determine when there are contradictions in an ontology (such as *Protege*) and to identify

a better classification of a concept of an ontology in agreement with the other concepts of the ontology itself.
For this purpose, a first validation phase of the *Km4city* ontology was carried out at the end of its creation and, after each its expansion: exploiting the potential of *Protege* (http://protege.stanford.edu/) was simulated the inference which could be generated due to the ontology defined, thanks to the applied reasoner, that is *HermiT 1.3.8* (http://hermit-reasoner.com/) to verify that the ontology did not contains contradictions or constructs poorly formulated.
A more dynamic validation approach provides instead the study and execution of tests, in order to discover defects, in the absence of which is possible to demonstrate that the system meets the needs for which it was created; this type of procedure is one of the most used for the verification and validation of a systems, and is also the one that allows to obtain more reliable results [Gangemi et al., 2006].

Typical problems that may be encountered are related to: (i) low quality of data, (ii) lack of data that are supposed to arrive in real time, (iii) changes in the data model of the data set, (iv) changes and updates into the data sets (this problem could generate a change into the ontolog,ical model and thus the human intervention is activated for model review), etc.
For example, the *Quality Improvement* process has allowed to add new fields to the ontological model, which in the presence of data clearly not in a correct format, were not foreseen in the first release of the ontology; if the amount of data added, reaches a certain size, a new Validation phase should be performed on that data.
To this end, some periodic verification and validation processes are also needed to be performed by defining a set of *SPARQL* queries on the knowledge base, with the aim of detecting inconsistencies and incompleteness, and verifying the correct status of the model.
For this purpose, a *JAVA* application was designed and developed to automate some parts of the verification and validation process, that has been chosen to be performed on the ontology. The "heart" of this application, is based on some *SPARQL* queries created ad hoc, that will be used to periodically query the triplestore, with the aim to verify if the results obtained are those expected. These periodically executed queries perform a regression testing, every time a new consistent update of ingested data is performed, and also when real time data arrive into the final *RDF* store.
The validation process may lead to identify problems that may be limited to the instances of classes. To this end, the Context macroclass of the *Km4city* ontology assumes a fundamental importance, because has been defined precisely in order to be used during the verification and validation phase: in fact, the fourth information associated with each triple allows to identify the problems but especially the processed datasets to be revised.

The entire process of verification and validation, implemented for the work of this thesis, can be divided into four phases, i.e. (i) design of test cases, (ii) selection of test data, (iii) run tests and, finally, (iv) verification and comparison of the results observed with those expected.

During the first phase, that is the test cases design, an automatic validation process has been designed, that has been initially limited to verifying how many triples and how many instances of each class, are actually loaded into the triplestore; later, thanks to a more detailed analysis of the links that the reasoner should be inferred, according to what defining inside the *Km4city* ontology, a manual validation was also carried out, more targeted to validate the inferred part, for which some ad hoc queries have been defined.

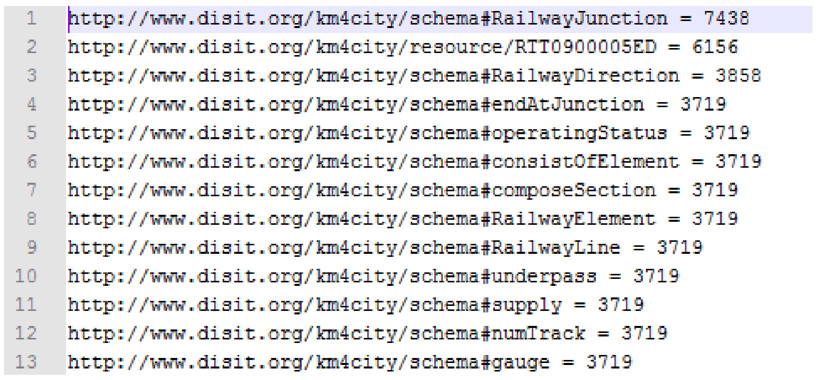
Initially, a check has been performed, to verify if the number of triples, actually loaded on triplestore, is equal to the number of triples contained in the corresponding file, and then, after uploading is complete, a second check is performed, to verify whether the instances number in the input data, for each most importance classes in the ontology, corresponds to the instances number, that are contained inside the triplestore, related to a certain context (that identifies the dataset input).
Information about the context is in fact exploited, to verify that, loading a set of triples generated from a specific dataset, leading to identify the exact instances number of the class in question, related to a context, which specifies precisely its origin.

Finally, some ad hoc queries, specifically developed to verify the inferred triples, thanks to *Sesame*, were formulated and executed on the triplestore.

The *Java* application realized to automate the validation and verification process, is able to perform an analysis on *N3* file, which are created during the mapping phase. The tool, in fact, scans the folders tree in which are stored the *N3* files, after their creation, extracts some useful parameters for each discovered file, such as the file name, the path to which it is stored, the dataset to which it refers, and the class of input data to which it belongs and, finally, it writes this information in a *MySQL* table, created specifically for the verification and validation process, named obviously *Validation*.

Once the operation of scanning all created triples files, is completed, the tool performs a parsing of each *N3* file, that have been inserted within the MySQL table *Validation*, stored inside the *Master* machine. The simple parser implemented, identifies each URI present in the analyzed triples files, that belongs to the *Km4city* ontology, thanks to the use of special *regex* and finally it counts the number of times that each URI is write inside the file. All *Km4city* URIs identified by the parser and the respective count are stored in *text* files specially created, ordered by appearance rate, starting from the most frequently up to the less frequent.

This procedure is repeated for each individual file identified during the previous scan: this means that for each triples file, the validation tool creates the corresponding counting URI file.
In the next Figure, the file created by the parser N3 is represented, which performs a triples counting.



```
http://www.disit.org/km4city/schema#RailwayJunction = 7438
http://www.disit.org/km4city/resource/RTT0900005ED = 6156
http://www.disit.org/km4city/schema#RailwayDirection = 3858
http://www.disit.org/km4city/schema#endAtJunction = 3719
http://www.disit.org/km4city/schema#operatingStatus = 3719
http://www.disit.org/km4city/schema#consistOfElement = 3719
http://www.disit.org/km4city/schema#composeSection = 3719
http://www.disit.org/km4city/schema#RailwayElement = 3719
http://www.disit.org/km4city/schema#RailwayLine = 3719
http://www.disit.org/km4city/schema#underpass = 3719
http://www.disit.org/km4city/schema#supply = 3719
http://www.disit.org/km4city/schema#numTrack = 3719
http://www.disit.org/km4city/schema#gauge = 3719
```

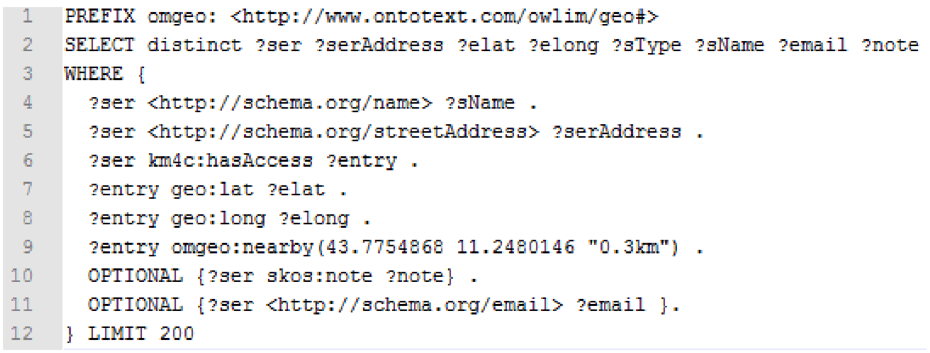
Figure 84 - Triples counting results

After an individual triples file analysis, in most cases manually performed, it was possible to identify, for each file, at least one *Km4city* URI, which assumes significance if validated in the context of that single triples file; that URI has been then stored into the MySQL table *Validation*, inside the row related to the context in which it will be validated. This process, allows to select data on which verification and validation tests must be performed.

The tool then deals with the implementation of verification and validation tests: automatically, for each row in the *Validation* table, the count of the selected URI is extracted, to be verified in its context (i.e. the file to which the row referred), and is compared with the results of a SPARQL query, performed on the fly, that return the same count of the same URI but within the triplestore, limited to the context of the analyzed file.
The two counts must obviously be equal to demonstrate that the data contained in the triples file in question, is correctly entered into the repository.
A further validation on counting triples currently loaded into the repository, can be carried out using information provided by *Owlim-SE* after the uploading of each *N3* file, is complete: *Owlim-SE* in fact returns the number of statements that have correctly read and loaded into the repository, a number that can be compared with the total triples amount of *N3* files, performed by the *Validation* tool, during its parsing phase, and stored within the MySQL table *ProcessManager*.
It was decided to move these two information items within the *processManager* table, because it is also viewable by non-administrator users, and so there is the possibility to more quickly detect any conflicting values, between the two columns containing the

total number of statements counted in the file and the number of statements actually loaded into the repository by *Owlim-SE*.

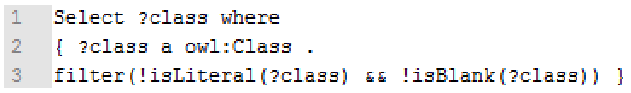
Finally, the verification and validation tests on interconnections created thanks to inferred triples, can be performed executing a couple of query designed and created specifically to verify most significant test cases of interconnection. In the following Figure it is possible to observe a query used to verify the connection beetween the *Point of Interest* and the *Street Guide and Rail Network* Macroclasses, realized thanks to the *Service* reconciliation at street number level.



```
1   PREFIX omgeo: <http://www.ontotext.com/owlim/geo#>
2   SELECT distinct ?ser ?serAddress ?elat ?elong ?sType ?sName ?email ?note
3   WHERE {
4     ?ser <http://schema.org/name> ?sName .
5     ?ser <http://schema.org/streetAddress> ?serAddress .
6     ?ser km4c:hasAccess ?entry .
7     ?entry geo:lat ?elat .
8     ?entry geo:long ?elong .
9     ?entry omgeo:nearby(43.7754868 11.2480146 "0.3km") .
10    OPTIONAL {?ser skos:note ?note} .
11    OPTIONAL {?ser <http://schema.org/email> ?email }.
12  } LIMIT 200
```

Figure 85 - Query to verify connection beetween Point of Interest and the Street Guide and Rail Network Macroclasses

This query, for example, allows to find services that are located within a radius of 300 meters from the point specified at line 9 by the coordinates.
The query in Figure 86 instead, represent a query crucial to work property with the *LOG* (see Section 5.7.2).



```
Select ?class where
{ ?class a owl:Class .
filter(!isLiteral(?class) && !isBlank(?class)) }
```

Figure 86 - A crucial query for the LOG

This query has been included in the validation because, with previous versions of the repository, has happened that did not allow to obtain results, and thus limited the use of the same *LOG* tool.

The Validation tool performs these queries and the others selected to be validate, and stores results in some *MySQL* table, which can then easily checked by an experienced user, who can establish consistency with what is expected from the execution of each query.

5.7 Phase VII: Applications

In Figure 25 is possible to observe the last phase, named *Phase VII*, relating to the use of RDF triplestore generated during the project.
Data access in a triplestore takes place via an access point for SPARQL query execution, also called *SPARQL endpoint*. Thanks to a connection to this *endpoint*, custom queries can be performed, obtaining in a fast and reliable way, their results. The query execution, for example via the *Sesame* interface, requires knowledge of all ontologies used to construct the triplestore, otherwise it's necessary to develop applications that, thanks to the specially developed user interfaces, allow to perform queries without having to know the basic ontology on which the triplestore is realized.
During the project, two applications have been developed, both accessing data using the SPARQL *endpoint*:

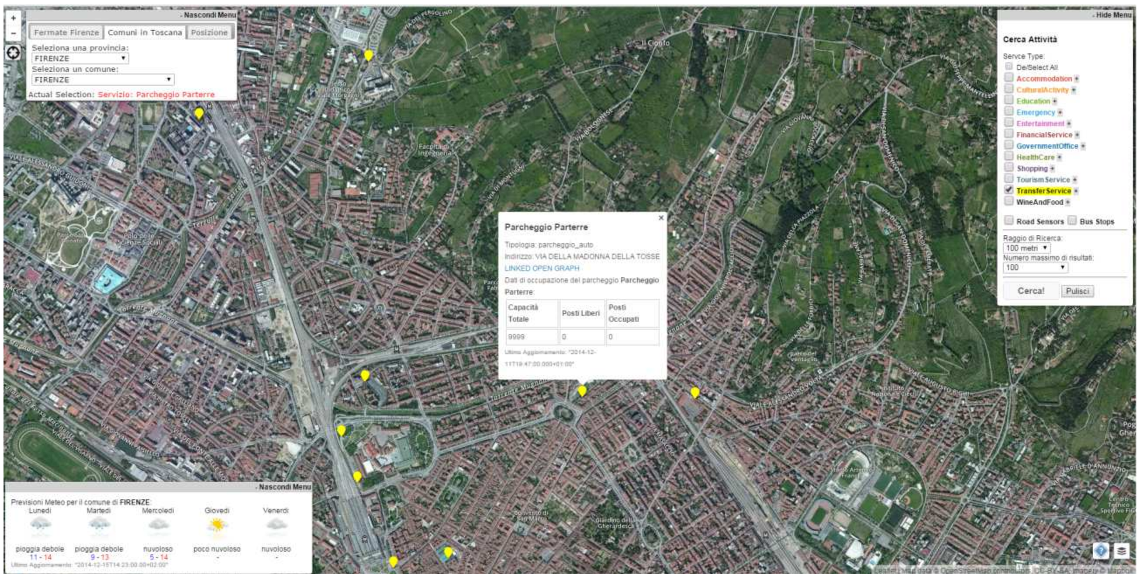
- ServiceMap (http://servicemap.disit.org), a map based application
- Linked Open Graph (http://log.disit.org) for browsing the data from SPARQL/Linked Data sources

The navigation on internet accessible RDF stores is becoming every day more relevant. They are frequently based on local and commonly accepted ontologies and vocabularies to set up large knowledge base to solve specific problems of modelling and reasoning. The growing needs of such structures increased the need of having flexible and accessible tools for RDF store browsing, taking into account multiple SPARQL entry points, to create and analyze reticular structure and scenarios of remote stores.
The *ServiceMap* presented here, provides an example of such innovative services can be integrated on a simple search application for geographic location, through the use of an RDF data store, within which the data are fully interconnected to each other. The main problems encountered with this type of application are related to (i) performance, that is, the difficulty of achieving an architecture that allows to obtain query results in acceptable times; (ii) the implementation of an interface simple, intuitive and user-friendly, to which users can access through both PC (laptop, desktop pc) and mobile devices (smartphone, tablet, etc.); (iii) the identification of real use cases, representing users real-life situations.
The *LOG* tool presented in this paragraph, provides, instead, innovative features solving a number of problems related to graph computation, to cope with high complexity of large LOD graphs with a web based tool. The complexity is mainly managed by providing tools for (i) progressive browsing of the graphs, (ii) allowing graph composition, (iii) providing support to pose specific queries, (iv) allowing the progressive discovering/selection of instances.
The next paragraphs are devoted to the description of the two applications made.

5.7.1 *Service Map*

Due to the high geographic information content of triplestore made in the project, the first application developed is the *ServiceMap*, i.e., a map in which users can view the results of their geo-referenced query, via browser.

Figure 87 shows the *ServiceMap* interface, an application developed in accordance with the main use cases identified on the data currently stored, within the triplestore.



Seleziona una provincia: FIRENZE
Seleziona un comune: FIRENZE
Actual Selection: Servizio: Parcheggio Parterre

Service Type:
- [ ] De/Select All
- [ ] Accommodation
- [ ] CulturalActivity
- [ ] Education
- [ ] Emergency
- [ ] Entertainment
- [ ] FinancialService
- [ ] GovernmentOffice
- [ ] HealthCare
- [ ] Shopping
- [ ] TourismService
- [✓] TransferService
- [ ] WineAndFood

[ ] Road Sensors [ ] Bus Stops

Raggio di Ricerca: 100 metri
Numero massimo di risultati: 100

Parcheggio Parterre

Tipologia: parcheggio_auto
Indirizzo: VIA DELLA MADONNA DELLA TOSSE
LINKED OPEN GRAPH
Dati di occupazione del parcheggio **Parcheggio Parterre**:

Capacità Totale	Posti Liberi	Posti Occupati
9999	0	0

Ultimo Aggiornamento: "2014-12-11T19:47:00.000+01:00"

Previsioni Meteo per il comune di **FIRENZE**:

Lunedì	Martedì	Mercoledì	Giovedì	Venerdì
pioggia debole	pioggia debole	nuvoloso	poco nuvoloso	nuvoloso
11 - 14	9 - 13	5 - 14	-	-

Ultimo Aggiornamento: "2014-12-15T14:23:00.00+02:00"

Figure 87 - ServiceMap interface

The *ServiceMap* is a JSP application (Java Server Page) that, thanks to the Java API of the Sesame framework, allows to retrieve data from a repository.
To create the map, the JavaScript library *Leaflet* (http://leafletjs.com/), has been used, which allows, thanks to script with extremely compact size, to create interactive maps inside a web application. Thanks to the well-documented API and to a long list of third-party plugins easily installable, *Leaflet* can construct a high usable map, which makes use of *OpenStreetMap* (http://www.openstreetmap.org/), as open-source maps.
In Figure 88 a schematization of the process behind the *ServiceMap*, can be viewed. The user connects to the *ServiceMap* Application via web browser and, through the interaction with the map, some requests to the server are formed and sent. The server provides to process requests and convert them to SPARQL query, that will be sent to the project's RDF store. Once query has been processed, the triplestore sends back the resulting triples, to the server, which interprets them and turns them into files HTML or JSON, that can be displayed directly by the browser.

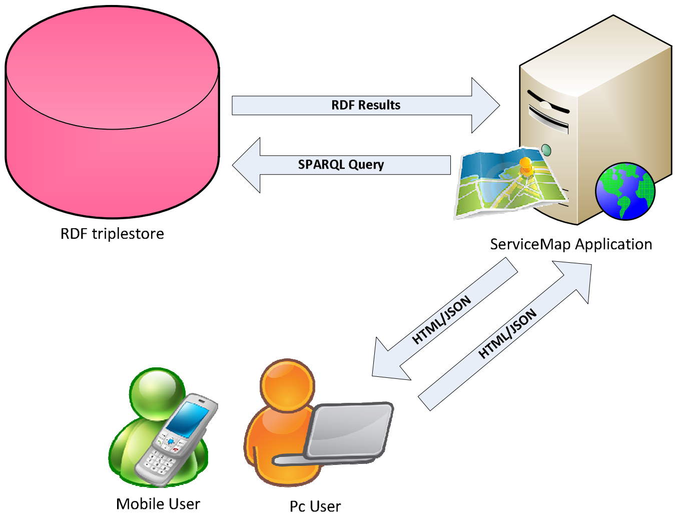
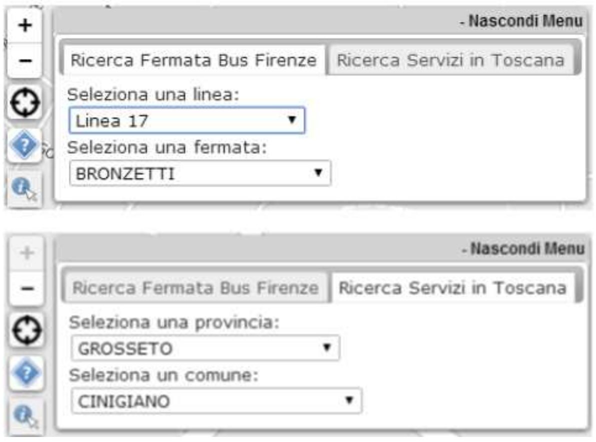


Figure 88 - Schematization of how ServiceMap works

The application has been designed so that the map, that is, its main component, occupy as much space as possible, while the interaction menus are divided into three parts: (i) in the upper left corner, the initial filters of the use cases implemented until now, are located; (ii) in the upper right hand, there are filters that allows to specify the types of services that must be shown, the maximum number of services to be displayed and the maximum research distance from the map center point; (iii) finally, in the bottom left corner there is a context menu that displays additional information, interesting for different use cases, such as weather forecasts or time arrival predictions at a bus stop, or data regarding free places in a given parking, shown according to the type of service selected.



Seleziona una linea: Linea 17
Seleziona una fermata: BRONZETTI

Seleziona una provincia: GROSSETO
Seleziona un comune: CINIGIANO

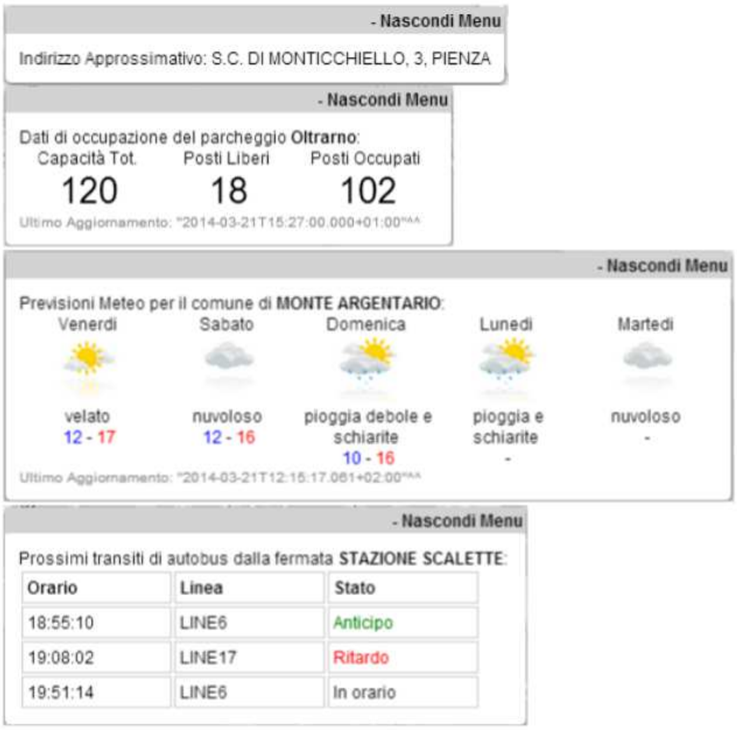
Figure 89 - Use cases menu

Figure 89 shows the two tab menus located in the upper left corner. On the left box side, some buttons have been included that perform the following functions:

- the first two buttons from the top, are native in *Leaflet* and allows to control the map zoom;
- the just below button, instead, allows to activate the user's location research. Once the location has found, thanks to the local network information that a browser can share, the map is centered to the founded coordinates, to which a marker is also placed.
- the fourth button from the top, is a link to the web page that contains information about the developed service, but very closer to a support pages for the application.
- The last button instead, allows to select any point on the map, where a marker will be insert, and to know its coordinates and the corresponding approximate address.

The functions associated with the third and last button, allow to apply the use cases introduced in the following pages, starting from a different point on the map, chosen by the user, or from the user position.

The top right menu (see Figure 87) instead allows, through selecting *checkboxes*, to select which service categories must be retrieved from de triplestore. These categories correspond to subclasses of the *Service* class (e.g. *Entertainment*, *Education*, etc.) defined within the *Km4city* ontology, and their respective associated *ServiceCategory* (e.g. the *Education* subclass, has as *ServiceCategory* "*private infant school*", "*public high school*", etc). An additional checkbox is located under the services menu, specifically for displaying *BusStop* objects.
In the geolocalized search, the maximum radius search can be set (up to 500 meters) together with the maximum number of results to be displayed, to avoid markers overcrowding on the map, which would became unreadable. The two buttons, in the bottom of this menu, i.e. "*Cerca!*" and "*Pulisci*", respectively actuate research, the first one, and the second one, bring the application to the initial state.
The last menu, that is the contextual one, allows the user to know information related to what is displayed/selected on the map; in Figure 90 there is an example of every possible information, that can be displayed on this type of menu.
The top right button of the context menu, i.e. "*Nascondi Menu*", as might guess, allows to hide the entire menu and to display a greater portion of the map.



Indirizzo Approssimativo: S.C. DI MONTICCHIELLO, 3, PIENZA

Dati di occupazione del parcheggio **Oltrarno**:

Capacità Tot.	Posti Liberi	Posti Occupati
120	18	102

Ultimo Aggiornamento: "2014-03-21T15:27:00.000+01:00"^^

Previsioni Meteo per il comune di **MONTE ARGENTARIO**:

Venerdi	Sabato	Domenica	Lunedi	Martedi
velato	nuvoloso	pioggia debole e schiarite	pioggia e schiarite	nuvoloso
12 - 17	12 - 16	10 - 16	-	-

Ultimo Aggiornamento: "2014-03-21T12:15:17.061+02:00"^^

Prossimi transiti di autobus dalla fermata **STAZIONE SCALETTE**:

Orario	Linea	Stato
18:55:10	LINE6	Anticipo
19:08:02	LINE17	Ritardo
19:51:14	LINE6	In orario

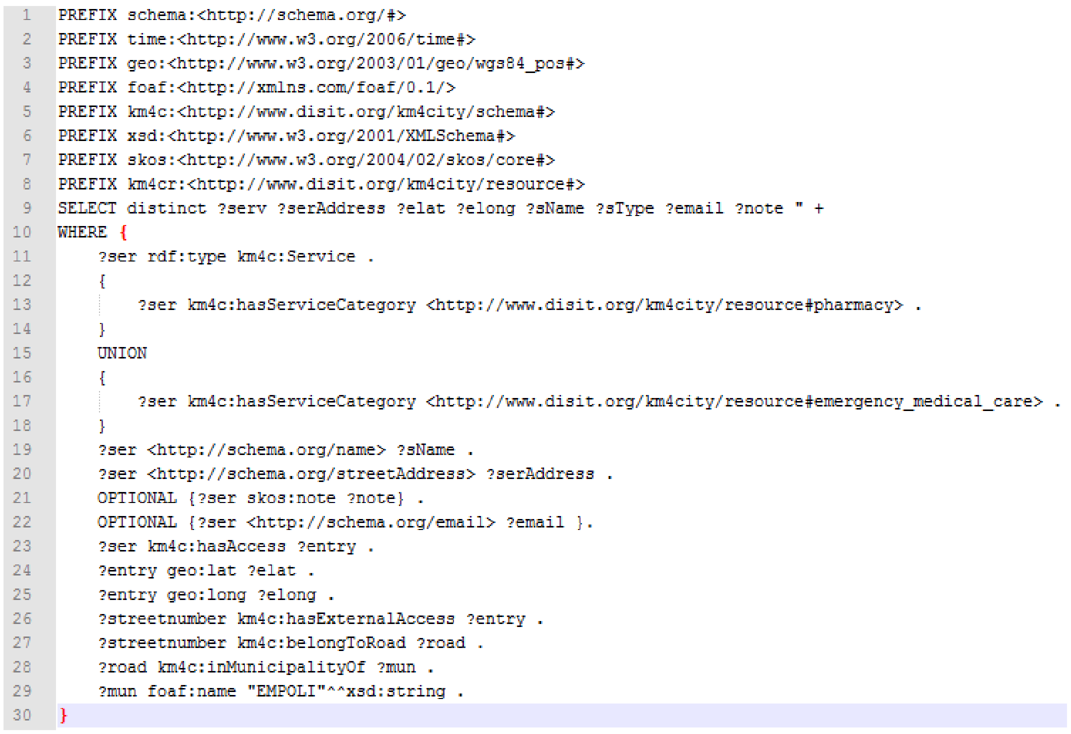
Figure 90 - Real Time data possible views

Thanks to the high performance RDF engine implemented in *OWLIM-SE* and through the use of extremely powerful geospatial indices, is possible to obtain results from queries with a very short time response, despite the substantial amount of triple georeferenced inside the project triplestore.
In the next paragraphs, three use cases will be analyzed as an example.

Use Case 1: Search for services belonging to a specified municipality

The first use case concerns search for services belonging to a given Tuscan municipality. The search type used in this case is not geographical, in fact the results are simply filtered according to the selected municipality.
To achieve the desired results, initially the province to which the municipality belonging must be selected and then the municipality name, thanks to the two *dropdown* named "*Seleziona una provincia*" and "*Seleziona un comune*". If the province to which the sought municipality belonging is not known, in the first dropdown choice is possible to select "*Tutte le province*", which allows to access, via the second dropdown, to the list of all 285 municipalities of Tuscany region.
Once the municipality has been selected, the service categories to be displayed, must be selected inside the special filtering menu (see Figure 87); as soon as the search radius and the maximum number of provided results are selected, just press the search button,

the search begins. In Figure 91 an example of SPARQL query generated from the application, has shown; it search for "*Guardia medica*" and "*Farmacia*" in Empoli, while the next figure, shows the obtained results.



```
PREFIX schema:<http://schema.org/#>
PREFIX time:<http://www.w3.org/2006/time#>
PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX km4c:<http://www.disit.org/km4city/schema#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
PREFIX km4cr:<http://www.disit.org/km4city/resource#>
SELECT distinct ?serv ?serAddress ?elat ?elong ?sName ?sType ?email ?note " +
WHERE {
    ?ser rdf:type km4c:Service .
    {
        ?ser km4c:hasServiceCategory <http://www.disit.org/km4city/resource#pharmacy> .
    }
    UNION
    {
        ?ser km4c:hasServiceCategory <http://www.disit.org/km4city/resource#emergency_medical_care> .
    }
    ?ser <http://schema.org/name> ?sName .
    ?ser <http://schema.org/streetAddress> ?serAddress .
    OPTIONAL {?ser skos:note ?note} .
    OPTIONAL {?ser <http://schema.org/email> ?email }.
    ?ser km4c:hasAccess ?entry .
    ?entry geo:lat ?elat .
    ?entry geo:long ?elong .
    ?streetnumber km4c:hasExternalAccess ?entry .
    ?streetnumber km4c:belongToRoad ?road .
    ?road km4c:inMunicipalityOf ?mun .
    ?mun foaf:name "EMPOLI"^^xsd:string .
}
```

Figure 91- SPARQL query to find a pharmacy in Empoli

ServiceAddress	Elat	Elong	ServiceName	Service Type
PIAZZA SAN ROCCO, 10	43,7193405	10,9393343	NUOVA DOTTOR VALOROSI	farmacia
PIAZZA DELLA VITTORIA, 26	43,7202826	10,9489318	BIZZARRI	farmacia
VIA DEI CAPPUCCINI, 18	43,7143452	10,9475482	COMUNALE	farmacia
VIA DEL PAPA, 20	43,7192948	10,9480072	CASTELLANI GIUSEPPE	farmacia
VIA VAL D'ORME, 83	43,7004727	10,95287	BOLOGNESI	farmacia

Table 17 - Results of the previous SPARQL query

As previously stated, the recovered data in RDF format are then converted in JSON format, by the *ServiceMap*; this choice is related to the format compatibility with *Leaflet*, which otherwise would not be able to place markers on the map.

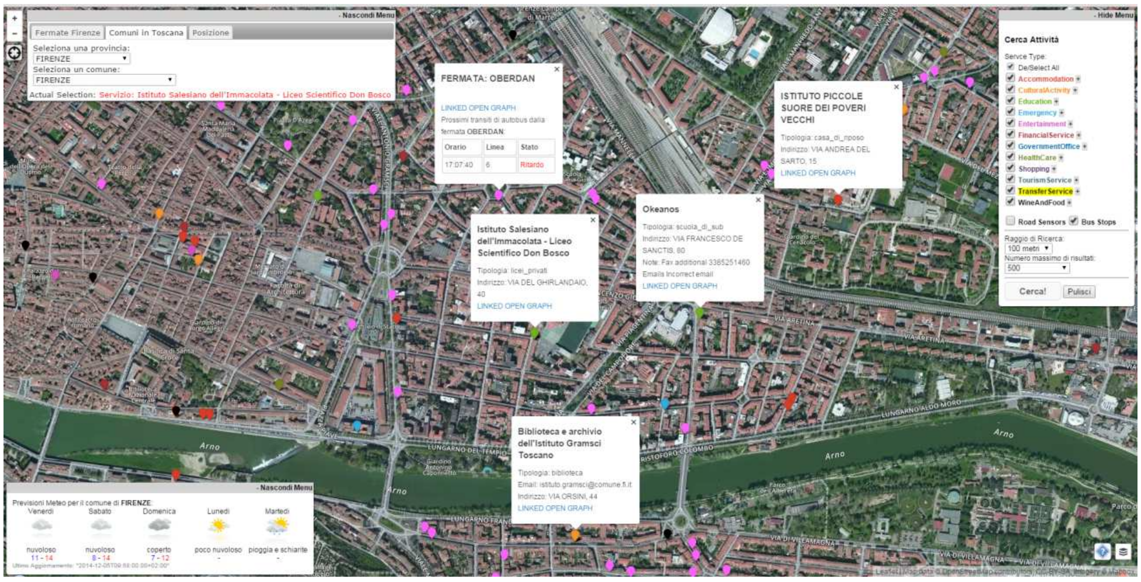

FERMATA: OBERDAN
Istituto Salesiano dell'Immacolata - Liceo Scientifico Don Bosco
Okeanos
ISTITUTO PICCOLE SUORE DEI POVERI VECCHI
Biblioteca e archivio dell'Istituto Gramsci Toscano
Cerca Attività
Cerca!
Pulisci
Arno

Figure 92- Results on the map

The following figure represents a sequence diagram between client side and server side.

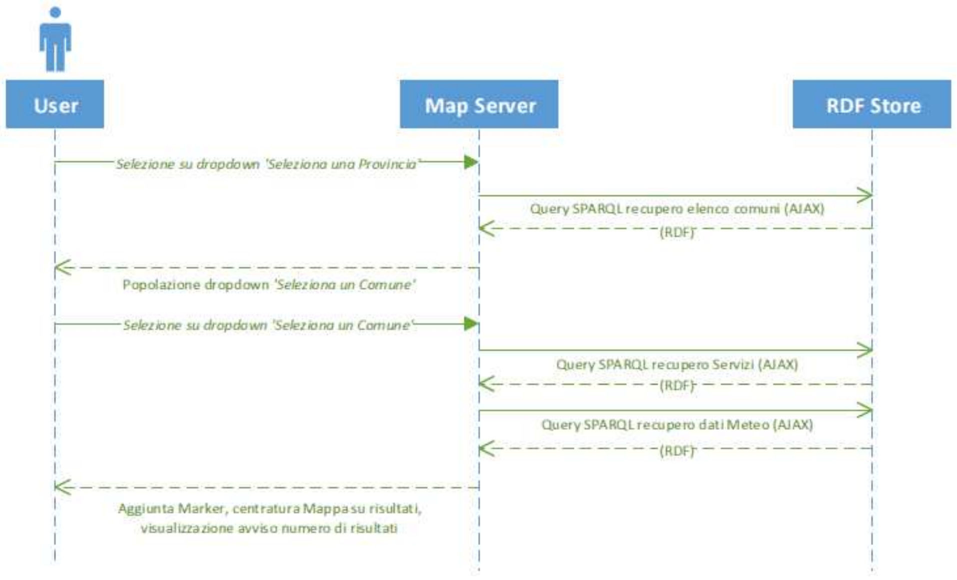
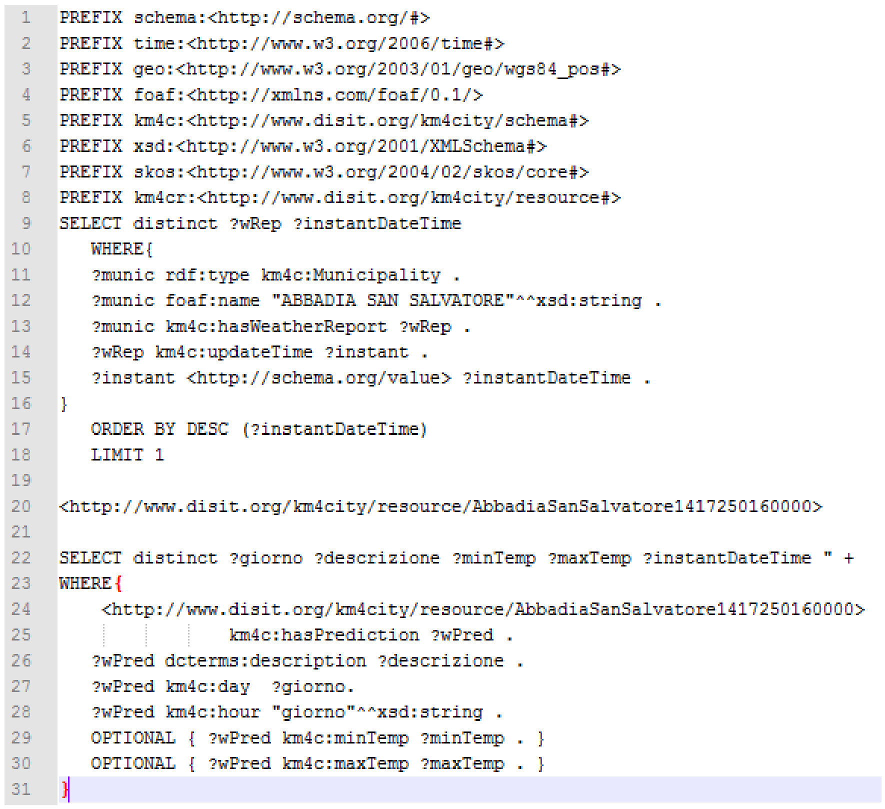

User
Map Server
RDF Store
Selezione su dropdown 'Seleziona una Provincia'
Query SPARQL recupero elenco comuni (AJAX)
(RDF)
Popolazione dropdown 'Seleziona un Comune'
Selezione su dropdown 'Seleziona un Comune'
Query SPARQL recupero Servizi (AJAX)
(RDF)
Query SPARQL recupero dati Meteo (AJAX)
(RDF)
Aggiunta Marker, centratura Mappa su risultati, visualizzazione avviso numero di risultati

Diagram 4 - UseCase 1 sequence diagram

Simultaneously to the query that is seeking the services within the selected municipality, a second query is launched, to retrieve the latest weather forecast of the same municipality. The search procedure is composed by two sequential queries: the first one extracts the most recent *WeatherReport* (lines 1-14), while the second one extracts, from the reports obtained as a results of the previous query, the five daily forecast, following today's date (lines 18-31); queries are shown in Figure 93

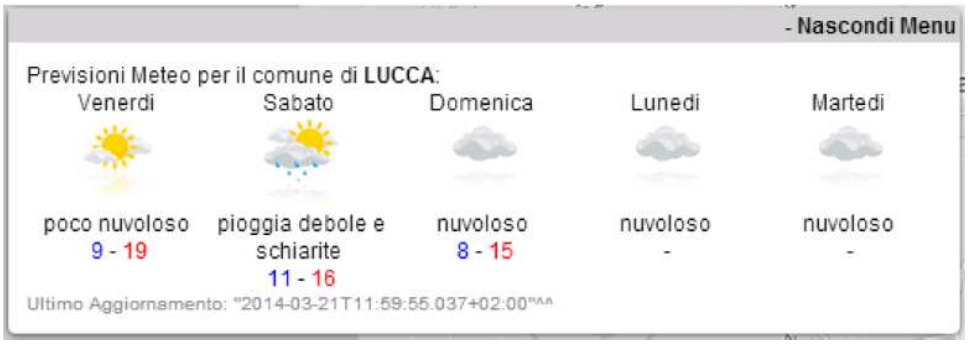


```
 1  PREFIX schema:<http://schema.org/#>
 2  PREFIX time:<http://www.w3.org/2006/time#>
 3  PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
 4  PREFIX foaf:<http://xmlns.com/foaf/0.1/>
 5  PREFIX km4c:<http://www.disit.org/km4city/schema#>
 6  PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
 7  PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
 8  PREFIX km4cr:<http://www.disit.org/km4city/resource#>
 9  SELECT distinct ?wRep ?instantDateTime
10     WHERE{
11     ?munic rdf:type km4c:Municipality .
12     ?munic foaf:name "ABBADIA SAN SALVATORE"^^xsd:string .
13     ?munic km4c:hasWeatherReport ?wRep .
14     ?wRep km4c:updateTime ?instant .
15     ?instant <http://schema.org/value> ?instantDateTime .
16  }
17     ORDER BY DESC (?instantDateTime)
18     LIMIT 1
19
20  <http://www.disit.org/km4city/resource/AbbadiaSanSalvatore1417250160000>
21
22  SELECT distinct ?giorno ?descrizione ?minTemp ?maxTemp ?instantDateTime " +
23  WHERE{
24      <http://www.disit.org/km4city/resource/AbbadiaSanSalvatore1417250160000>
25                  km4c:hasPrediction ?wPred .
26     ?wPred dcterms:description ?descrizione .
27     ?wPred km4c:day  ?giorno.
28     ?wPred km4c:hour "giorno"^^xsd:string .
29     OPTIONAL { ?wPred km4c:minTemp ?minTemp . }
30     OPTIONAL { ?wPred km4c:maxTemp ?maxTemp . }
31  }
```

Figure 93 - SPARQL query for weather predictions

At line 16, the value of *?WeatherReport*, resulting from the first query, is used to retrieve elements of type *?WeatherPrediction*; even from the first query, the *?instantDateTime* value is extracted, which corresponds to the report creation time. The ?*updateTime* value is instead extracted thanks to line 11 and it represents the report updating date, in *xsd:dateTime* format; according to this last value extracted, at line 14, the resulted records are sorted, and then only the first one is selected, that is the most recent.

In the second query, instead, lines 26-27-29-30 the following prediction attributes are extract:*?Days* (representing the day of week), *?Description* (the literal description of the forecast), *?MinTemp* and *?MaxTemp* (minimum and maximum temperature provided, respectively). These last two variables are associated with *OPTIONAL* SPARQL operator, because sometimes their values are null and without using this operator, the lines that do not show temperature values would be excluded from the results. Finally, at line 28, predictions are filtered by choosing only those having *?day* equal to "*giorno*", which represent the daily forecasts associated to a less granularity, but almost present for the 5 days following the today's date. The query results are then formatted to create the menu in Figure 94.



- Nascondi Menu

Previsioni Meteo per il comune di **LUCCA**:

Venerdi	Sabato	Domenica	Lunedi	Martedi
poco nuvoloso	pioggia debole e schiarite	nuvoloso	nuvoloso	nuvoloso
9 - 19	11 - 16	8 - 15	-	-

Ultimo Aggiornamento: "2014-03-21T11:59:55.037+02:00"^^

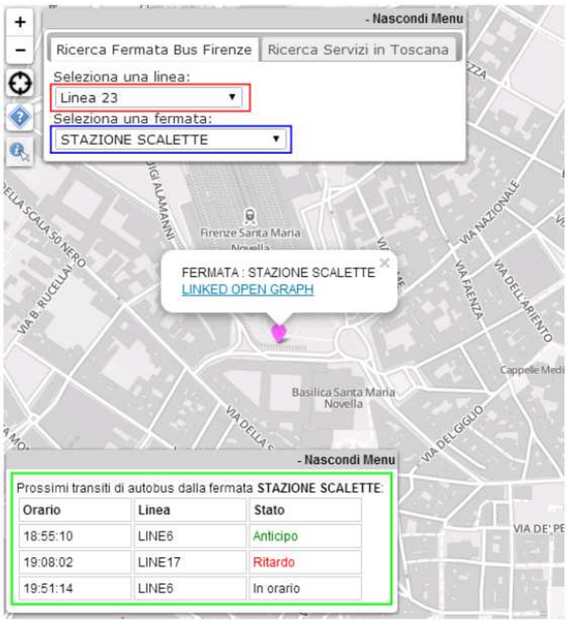
Figure 94 - Results of the previous SPARQL query on the map

Use Case 2: Search for services near a Bus Stop

The second use case concerns research of services nearby a bus stop, within the metropolitan area of Florence.

To enable this use case a bus line must be selected via the menu "*Seleziona una linea*" and then, in the dropdown below, a BusStop must be chosen; similarly to the previous case, in the first dropdown, the value "*tutte le linee*" can be selected, that allows to access the list of all bus stops.

If one bus stop is selected, the map will be center on its coordinates and a small pink marker, associated with the bus stop, will be displayed; simultaneously the system will load the context menu containing data providing by AVM systems, installed on vehicles, related to the bus transit time on that particular stop. In Figure 95 is possible to observe what has been described so far.

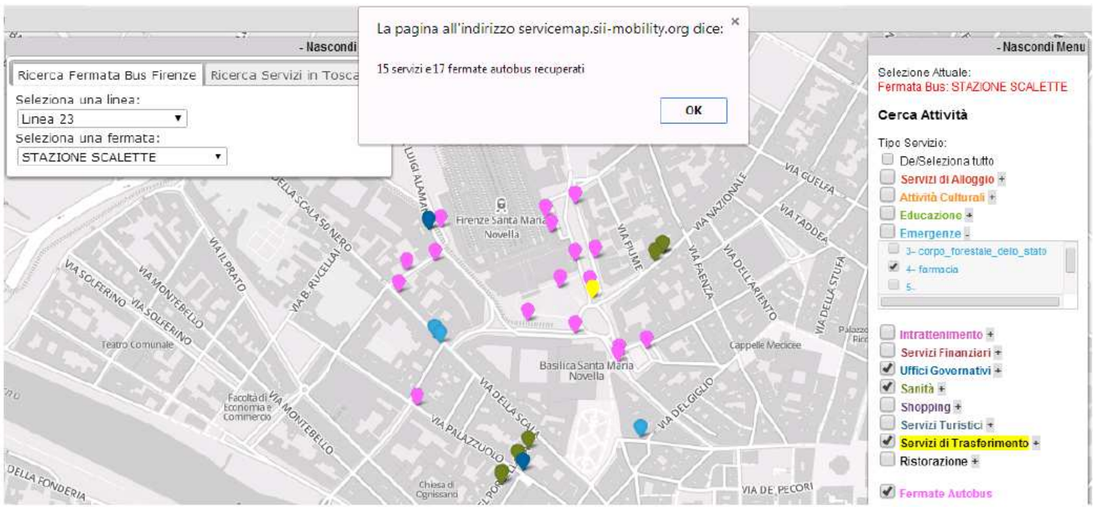


Prossimi transiti di autobus dalla fermata STAZIONE SCALETTE:

Orario	Linea	Stato
18:55:10	LINE6	Anticipo
19:08:02	LINE17	Ritardo
19:51:14	LINE6	In orario

Figure 95 - How to search a Bus Stop

As soon as the stop is selected, all interest services placed within a certain radius from the marker, can be searched: this is possible thanks to the menu located at the top right, where the service categories and their subcategories are selected, together with the search radius and the maximum number of results to display. Finally, clicking the "*Cerca!*" button, the requested services are obtained; the system displays a warning message that informs the user of how many services and many bus stops were recovered (Figure 96).


La pagina all'indirizzo servicemap.sii-mobility.org dice:

15 servizi e 17 fermate autobus recuperati

Ricerca Fermata Bus Firenze

Seleziona una linea: Linea 23

Seleziona una fermata: STAZIONE SCALETTE

Selezione Attuale:
Fermata Bus: STAZIONE SCALETTE

Cerca Attività

Tipo Servizio:
- [ ] De/Seleziona tutto
- [ ] Servizi di Alloggio +
- [ ] Attività Culturali +
- [ ] Educazione +
- [ ] Emergenze -
  - [ ] 3- corpo_forestale_dello_stato
  - [✓] 4- farmacia
  - [ ] 5-
- [ ] Intrattenimento +
- [ ] Servizi Finanziari +
- [✓] Uffici Governativi +
- [✓] Sanità +
- [ ] Shopping +
- [ ] Servizi Turistici +
- [✓] Servizi di Trasferimento +
- [ ] Ristorazione +
- [✓] Fermate Autobus

Figure 96 - Services near a Bus Stop

The query in Figure 97 is the one made by the system, based on the choices made in this second use case.

```
PREFIX schema:<http://schema.org/#>
PREFIX time:<http://www.w3.org/2006/time#>
PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX km4c:<http://www.disit.org/km4city/schema#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
PREFIX km4cr:<http://www.disit.org/km4city/resource#>
SELECT distinct ?ser ?serAddress ?elat ?elong ?sType ?sName ?email ?note
WHERE {
  ?ser km4c:hasServiceCategory <http://www.disit.org/km4city/resource#pharmacy> .
  ?ser <http://schema.org/name> ?sName .
  ?ser <http://schema.org/streetAddress> ?serAddress .
  ?ser km4c:hasAccess ?entry .
  ?entry geo:lat ?elat .
  ?entry geo:long ?elong .
  ?entry omgeo:nearby(43.7754868 11.2480146 "0.3km") .
  OPTIONAL {?ser skos:note ?note} .
  OPTIONAL {?ser <http://schema.org/email> ?email }.
} LIMIT 200
```

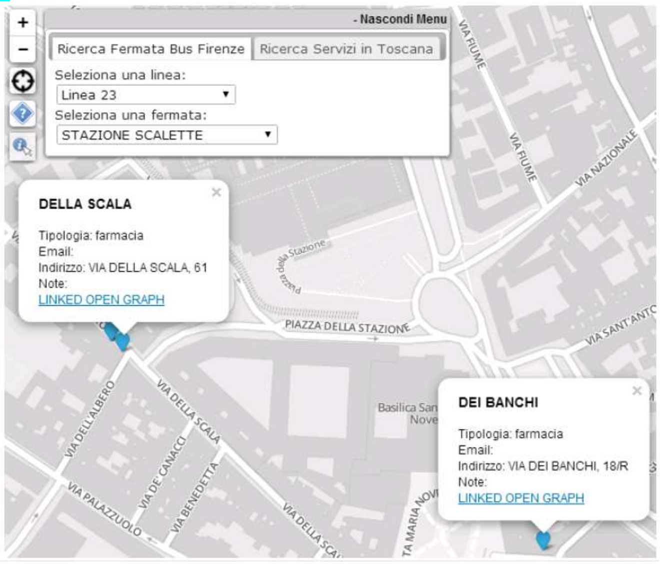
Figure 97 - SPARQL query to find services near a Bus Stop

Line 7 of the query, specifies which information, about the services, that fulfill the requirements selected, must be contained in the results, that is: the URI of the service (*?Ser*), its address (*?SerAddress*), its coordinates (*?Elat* and *?Elong*), the name of the service (*?sName*), its email address (*?email*) and *?notes* that contains all other information related to the specified service.
At line 10 the selected service categories are specified and, among all the services found, only those associated with an instance of the class *Entry* of the *Km4City* ontology, will be used later. At line 16, in fact the instruction *Omgeo:nearby* allows to filter out the data based on the distance from a central point, in that case the bus station "*Stazione Scalette*". The last parameter of *Omgeo:nearby* command, is the search radius. Finally, thanks to the *LIMIT* operator, the maximum number of results can be set; the results obtained with this query are shown in below in tabular format (Table 18).

ServiceAddress	Elat	Elong	ServiceName	Service Type
VIA DEI BANCHI, 18/R	43,773609	11,2505591	DEI BANCHI	farmacia
VIA DELLA SCALA, 61	43,7752329	11,2459294	DELLA SCALA	farmacia
VIA DELLA VIGNA NUOVA, 54/R	43,77118	11,2492657	SAN GIORGIO	farmacia
PIAZZA DEGLI OTTAVINI	43,7726956	11,2494195	CAMILLI	farmacia

Table 18 - Results of the previous SPARQL query

The results are then inserted into the map thanks to the *Leaflet*. The query result is shown in Figure 98.



Seleziona una linea: Linea 23
Seleziona una fermata: STAZIONE SCALETTE

DELLA SCALA
Tipologia: farmacia
Email:
Indirizzo: VIA DELLA SCALA, 61
Note:
LINKED OPEN GRAPH

DEI BANCHI
Tipologia: farmacia
Email:
Indirizzo: VIA DEI BANCHI, 18/R
Note:
LINKED OPEN GRAPH

Figure 98 - Results on the map

The sequence diagram between the client side and server side of the use case just described, is the following:

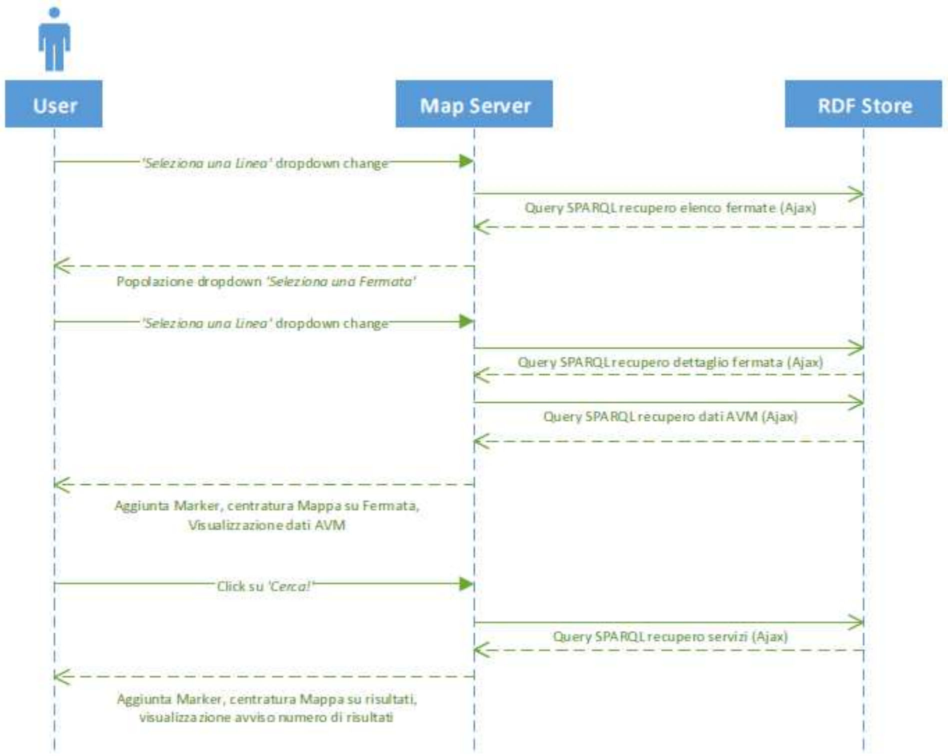
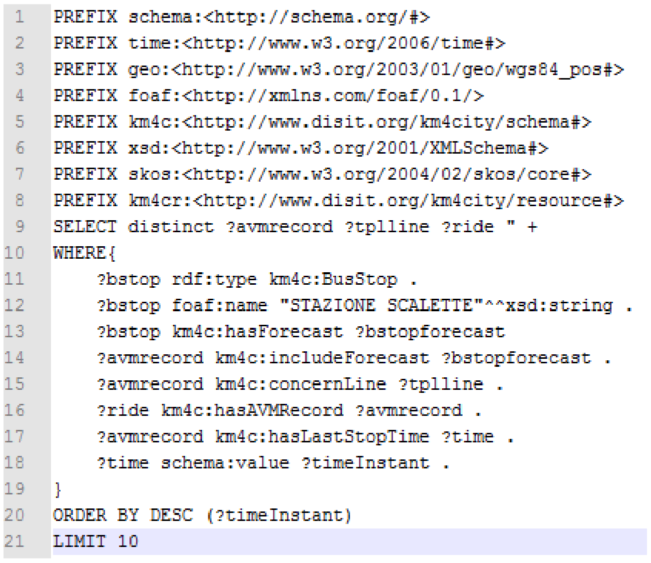

User
Map Server
RDF Store
'Seleziona una Linea' dropdown change
Query SPARQL recupero elenco fermate (Ajax)
Popolazione dropdown 'Seleziona una Fermata'
'Seleziona una Linea' dropdown change
Query SPARQL recupero dettaglio fermata (Ajax)
Query SPARQL recupero dati AVM (Ajax)
Aggiunta Marker, centratura Mappa su Fermata, Visualizzazione dati AVM
Click su 'Cerca!'
Query SPARQL recupero servizi (Ajax)
Aggiunta Marker, centratura Mappa su risultati, visualizzazione avviso numero di risultati

Diagram 5 -Use Case 2 sequence diagram

As mentioned above, when the user selects a bus stop, a query to retrieve the relevant context information to show in the menu, is performed; that query is quite complex, since it has to deal with Real-time data relating to the AVM system, and is shown in Figure 99:



```
PREFIX schema:<http://schema.org/#>
PREFIX time:<http://www.w3.org/2006/time#>
PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX km4c:<http://www.disit.org/km4city/schema#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
PREFIX km4cr:<http://www.disit.org/km4city/resource#>
SELECT distinct ?avmrecord ?tplline ?ride " +
WHERE{
    ?bstop rdf:type km4c:BusStop .
    ?bstop foaf:name "STAZIONE SCALETTE"^^xsd:string .
    ?bstop km4c:hasForecast ?bstopforecast
    ?avmrecord km4c:includeForecast ?bstopforecast .
    ?avmrecord km4c:concernLine ?tplline .
    ?ride km4c:hasAVMRecord ?avmrecord .
    ?avmrecord km4c:hasLastStopTime ?time .
    ?time schema:value ?timeInstant .
}
ORDER BY DESC (?timeInstant)
LIMIT 10
```

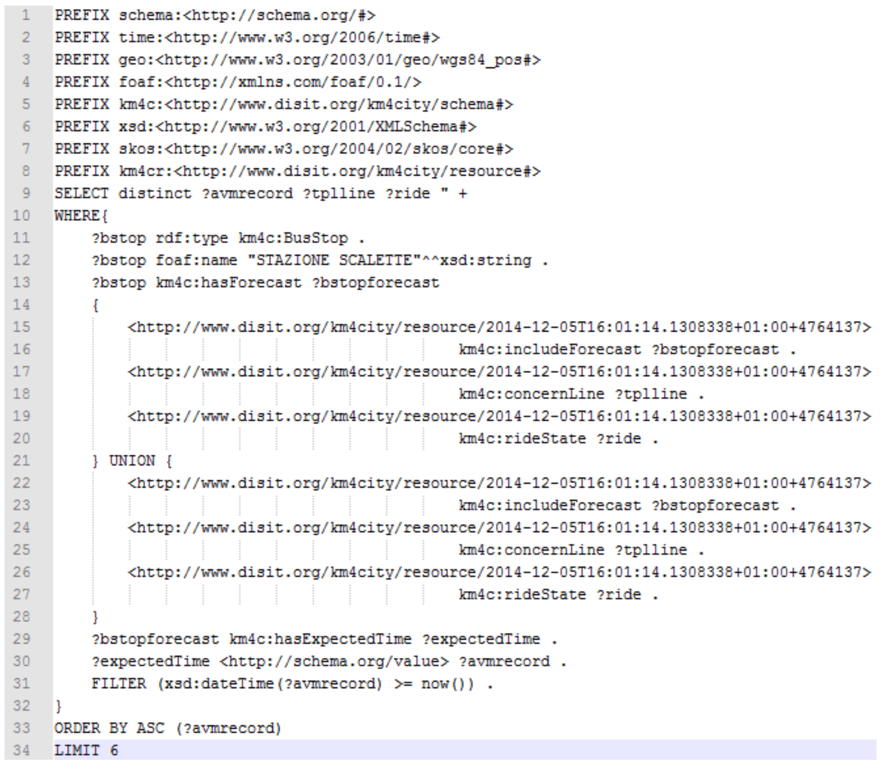
Figure 99 - SPARQL query for AVM

The query returns the latest 10 unique combinations of values *?AvmRecord*, *?Line* and *?Ride*, relative to the selected bus stop. By accessing *PublicTransport* and *RealTime* (only the part devoted to the AVM) macroclasses of the ontology *Km4City*, and taking advantage of their interconnections, the correct objects are extracted.
Through the last two lines (14-15) dedicated to data filtering, the temporal information, relating to the *AVMrecord* generation date, is also withdrawn. Finally, lines 17-18 are concerned with the reorganization of the recovered data, timely based. The result of this query is shown in Table 19.

AVM record	Line	Ride
2014-03-07t17:58:25.4584782+01:00+4737229	LINE 23	4737229
2014-03-07t17:58:25.4584782+01:00+4764073	LINE 17	4764073
2014-03-07t17:58:25.4584782+01:00+4764311	LINE 17	4764311
2014-03-07t17:58:25.4584782+01:00+4737165	LINE 23	4737165
2014-03-07t17:58:25.4584782+01:00+4764055	LINE 17	4764055
2014-03-07t17:58:25.4574781+01:00+4737239	LINE 23	4737239
2014-03-07t17:58:25.4584782+01:00+4737032	LINE 23	4737032
2014-03-07t17:58:25.4584782+01:00+4764259	LINE 23	4737114
2014-03-07t17:58:25.4574781+01:00+4764259	LINE 17	4764259
2014-03-07t17:50:25.2114583+01:00+4764311	LINE 17	4764311

Table 19 - Results of AVM SPARQL query

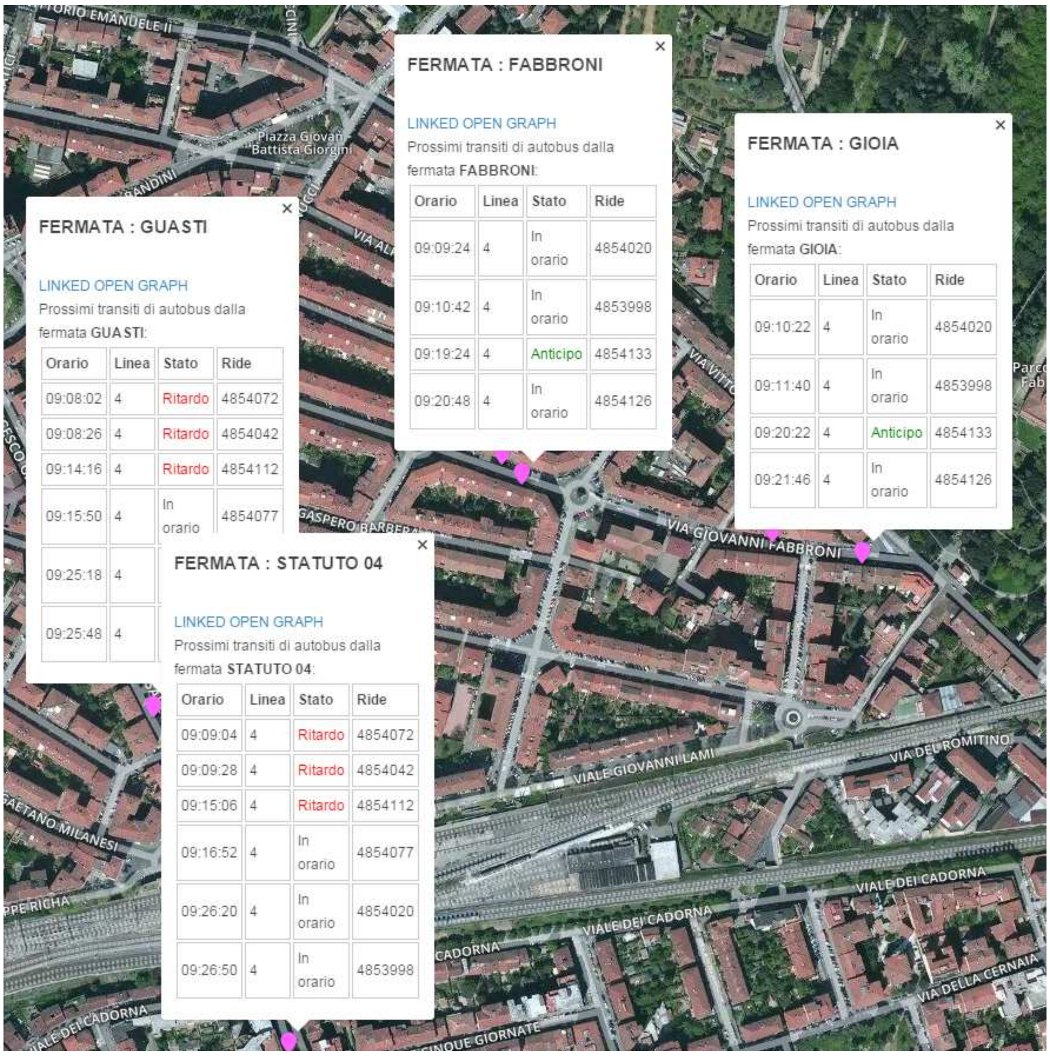
A second type of query is then composed, by exploiting the results of the first query, to detect the transit time of buses, contained in those *AVMRecord*. As it is possible see from the Figure 100, the second query is considerably more complicated.



```
1  PREFIX schema:<http://schema.org/#>
2  PREFIX time:<http://www.w3.org/2006/time#>
3  PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
4  PREFIX foaf:<http://xmlns.com/foaf/0.1/>
5  PREFIX km4c:<http://www.disit.org/km4city/schema#>
6  PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
7  PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
8  PREFIX km4cr:<http://www.disit.org/km4city/resource#>
9  SELECT distinct ?avmrecord ?tplline ?ride " +
10 WHERE{
11     ?bstop rdf:type km4c:BusStop .
12     ?bstop foaf:name "STAZIONE SCALETTE"^^xsd:string .
13     ?bstop km4c:hasForecast ?bstopforecast
14     {
15         <http://www.disit.org/km4city/resource/2014-12-05T16:01:14.1308338+01:00+4764137>
16                                     km4c:includeForecast ?bstopforecast .
17         <http://www.disit.org/km4city/resource/2014-12-05T16:01:14.1308338+01:00+4764137>
18                                     km4c:concernLine ?tplline .
19         <http://www.disit.org/km4city/resource/2014-12-05T16:01:14.1308338+01:00+4764137>
20                                     km4c:rideState ?ride .
21     } UNION {
22         <http://www.disit.org/km4city/resource/2014-12-05T16:01:14.1308338+01:00+4764137>
23                                     km4c:includeForecast ?bstopforecast .
24         <http://www.disit.org/km4city/resource/2014-12-05T16:01:14.1308338+01:00+4764137>
25                                     km4c:concernLine ?tplline .
26         <http://www.disit.org/km4city/resource/2014-12-05T16:01:14.1308338+01:00+4764137>
27                                     km4c:rideState ?ride .
28     }
29     ?bstopforecast km4c:hasExpectedTime ?expectedTime .
30     ?expectedTime <http://schema.org/value> ?avmrecord .
31     FILTER (xsd:dateTime(?avmrecord) >= now()) .
32 }
33 ORDER BY ASC (?avmrecord)
34 LIMIT 6
```

Figure 100 – Second SPARQL query for AVM

The query in the above figure, uses only 2 of the 10 results obtained (shown in lines 11-15 and 17-21) thanks to the first query, but for demonstration purposes, does not make much sense to use them all; this is the reason why the second query is made with just two results. *?forecast* elements are extracted using the *dinstinct* operator which eliminates the risk of duplicate data relating to the same ride; some cascade data filtering are then performed: at line 24 forecasts are filtered based on the current date and time, while the lines 26-27, reorder results and takes only the first four.
The recovered data until this point, can then be used to populate the context menu as shown in Figure 101.



FERMATA : FABBRONI

LINKED OPEN GRAPH
Prossimi transiti di autobus dalla fermata **FABBRONI**:

Orario	Linea	Stato	Ride
09:09:24	4	In orario	4854020
09:10:42	4	In orario	4853998
09:19:24	4	Anticipo	4854133
09:20:48	4	In orario	4854126

FERMATA : GIOIA

LINKED OPEN GRAPH
Prossimi transiti di autobus dalla fermata **GIOIA**:

Orario	Linea	Stato	Ride
09:10:22	4	In orario	4854020
09:11:40	4	In orario	4853998
09:20:22	4	Anticipo	4854133
09:21:46	4	In orario	4854126

FERMATA : GUASTI

LINKED OPEN GRAPH
Prossimi transiti di autobus dalla fermata **GUASTI**:

Orario	Linea	Stato	Ride
09:08:02	4	Ritardo	4854072
09:08:26	4	Ritardo	4854042
09:14:16	4	Ritardo	4854112
09:15:50	4	In orario	4854077
09:25:18	4		
09:25:48	4		

FERMATA : STATUTO 04

LINKED OPEN GRAPH
Prossimi transiti di autobus dalla fermata **STATUTO 04**:

Orario	Linea	Stato	Ride
09:09:04	4	Ritardo	4854072
09:09:28	4	Ritardo	4854042
09:15:06	4	Ritardo	4854112
09:16:52	4	In orario	4854077
09:26:20	4	In orario	4854020
09:26:50	4	In orario	4853998

Figure 101 - Results of the second AVM SPARQL query on the map

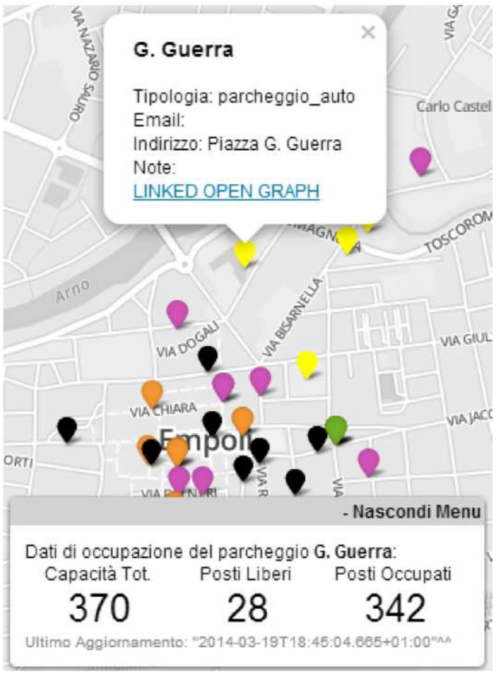
The services search, thanks to the special buttons located below the zoom buttons, can be generalized to research facilities nearby any point, such as the user GPS position detected thanks to the appropriate button in the upper left, or any other marker present on the map, or any other point, after using the button that allows to find the approximate address (always positioned at the top left).
This services search is very similar to what seen in the previous section, also the queries that the web service forms and sends to the server, because the only difference lies in the central point from which the search begins.

Use Case 3: How to display all the contextual menus

The last use case is devoted to other contextual searches that allow to display different information in the menu on the bottom left, and that have not been described, yet.
Specifically, the two most important research are the following:

- Control of occupation of a specific car park;
- Associating an approximate address to a generic point, selected on the map, thanks to procedures of reverse geocoding.

The first type of contextual information can be obtained by simply looking for a car park in the metropolitan area of Florence and clicking on the marker associated. After that, the system prompts data to the repository, relating to the most recent status update of the selected parking, and provide information to the user, as shown in Figure 102.



G. Guerra

Tipologia: parcheggio_auto
Email:
Indirizzo: Piazza G. Guerra
Note:
LINKED OPEN GRAPH

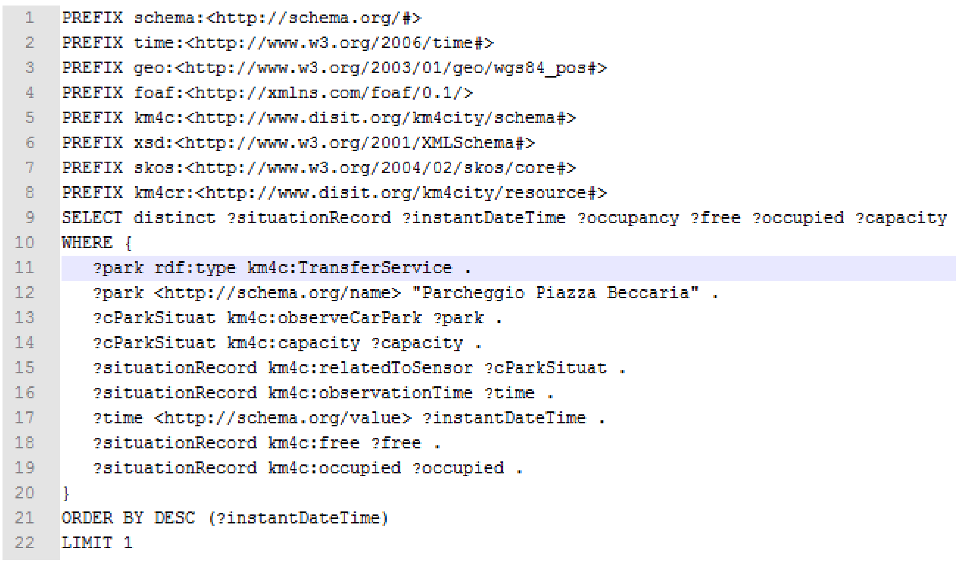
Dati di occupazione del parcheggio **G. Guerra**:

Capacità Tot.	Posti Liberi	Posti Occupati
370	28	342

Ultimo Aggiornamento: "2014-03-19T18:45:04.665+01:00"^^

Figure 102 - Parking Real Time information on the map

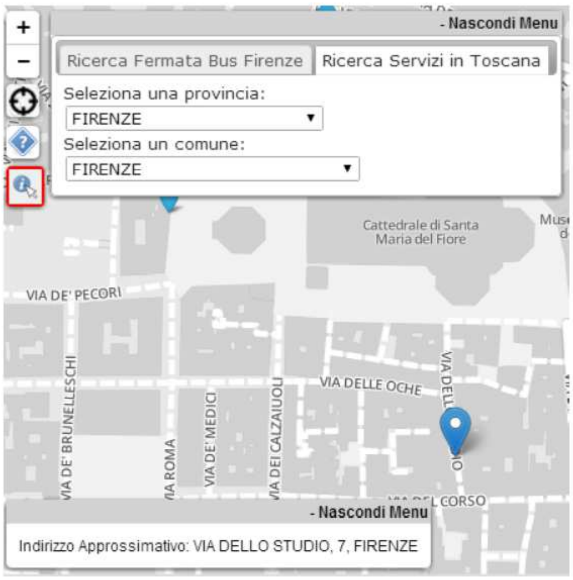
Below the query that allows to obtain this information is shown.



```
PREFIX schema:<http://schema.org/#>
PREFIX time:<http://www.w3.org/2006/time#>
PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX km4c:<http://www.disit.org/km4city/schema#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
PREFIX km4cr:<http://www.disit.org/km4city/resource#>
SELECT distinct ?situationRecord ?instantDateTime ?occupancy ?free ?occupied ?capacity
WHERE {
   ?park rdf:type km4c:TransferService .
   ?park <http://schema.org/name> "Parcheggio Piazza Beccaria" .
   ?cParkSituat km4c:observeCarPark ?park .
   ?cParkSituat km4c:capacity ?capacity .
   ?situationRecord km4c:relatedToSensor ?cParkSituat .
   ?situationRecord km4c:observationTime ?time .
   ?time <http://schema.org/value> ?instantDateTime .
   ?situationRecord km4c:free ?free .
   ?situationRecord km4c:occupied ?occupied .
}
ORDER BY DESC (?instantDateTime)
LIMIT 1
```

Figure 103 - SPARQL query for parking occupancy

The query retrieves the URI of the last *?SituationRecord* associated with the selected parking, the instant in which has been created (*?InstantDateTime*) and other data on its status, i.e. *?free*, *?occupied* and *?capacity*, respectively, the number of free places, occupied places and the total number of places.



Seleziona una provincia: FIRENZE

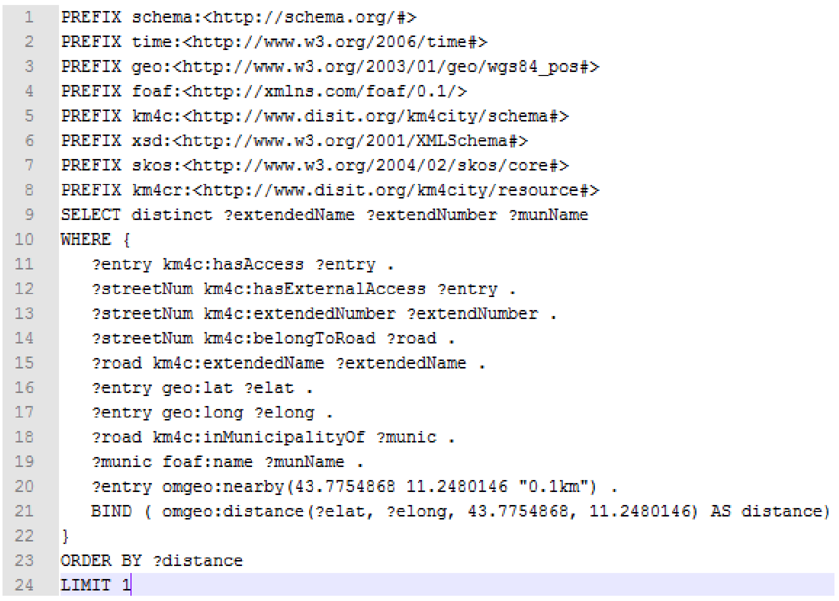
Seleziona un comune: FIRENZE

Indirizzo Approssimativo: VIA DELLO STUDIO, 7, FIRENZE

Figure 104 - Use Case of approximate address

In relation to the second type of contextual information displayable, i.e. the address associated with a selected point on the map, the user must first click on the function button (in the top left corner), and then on the point to which he wants to associate an

address and, only at this point, the operation of reverse geocoding is enabled; in Figure 104 an example of this use case, can be observe.
The SPARQL query, used to associate an approximate address to a pair of coordinates, is very similar to the query used during the bus stops reconciliation with elements of the *Road* class (see Section 5.5.1), and it is shown in Figure 105.



```
PREFIX schema:<http://schema.org/#>
PREFIX time:<http://www.w3.org/2006/time#>
PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX km4c:<http://www.disit.org/km4city/schema#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
PREFIX km4cr:<http://www.disit.org/km4city/resource#>
SELECT distinct ?extendedName ?extendNumber ?munName
WHERE {
   ?entry km4c:hasAccess ?entry .
   ?streetNum km4c:hasExternalAccess ?entry .
   ?streetNum km4c:extendedNumber ?extendNumber .
   ?streetNum km4c:belongToRoad ?road .
   ?road km4c:extendedName ?extendedName .
   ?entry geo:lat ?elat .
   ?entry geo:long ?elong .
   ?road km4c:inMunicipalityOf ?munic .
   ?munic foaf:name ?munName .
   ?entry omgeo:nearby(43.7754868 11.2480146 "0.1km") .
   BIND ( omgeo:distance(?elat, ?elong, 43.7754868, 11.2480146) AS distance)
}
ORDER BY ?distance
LIMIT 1
```

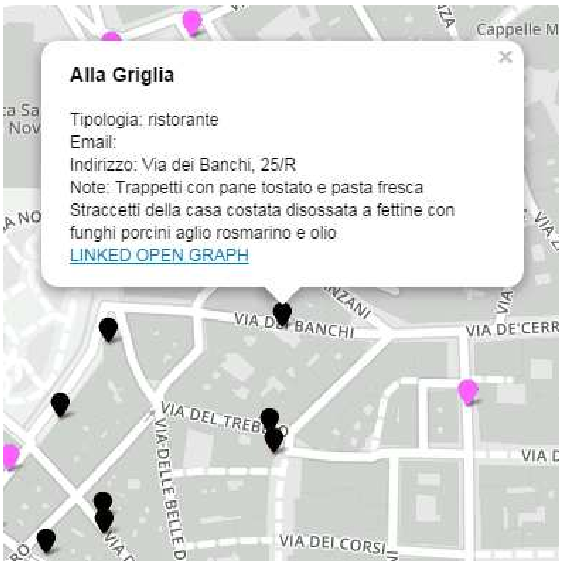
Figure 105 - SPARQL query to find the approximate address

Similarly to what is done during the bus stops reconciliation, even in this query the nearest neighbor method is used to search within a radius of 100 meters. To obtain the desired information, the values retrieved are the follow:

- *?ExtendedName* of the *Road* element,
- *?ExtendedNumber* that is the house number and
- the town name (*?MunicipalityName*) is recovered, concatenated and finally it is displayed in the context menu in the lower left.

5.7.2 *Linked Open Graph*

From the ServiceMap described in the previous section, by clicking on one of the markers corresponding to a service or a bus stop, a small dialog box is open as shown in Figure 106. Furthermore, clicking on the "*Linked Open Graph*" link, which is located inside the dialog box, is also possible display information linked to the object of interest, through a Faceted Graph.



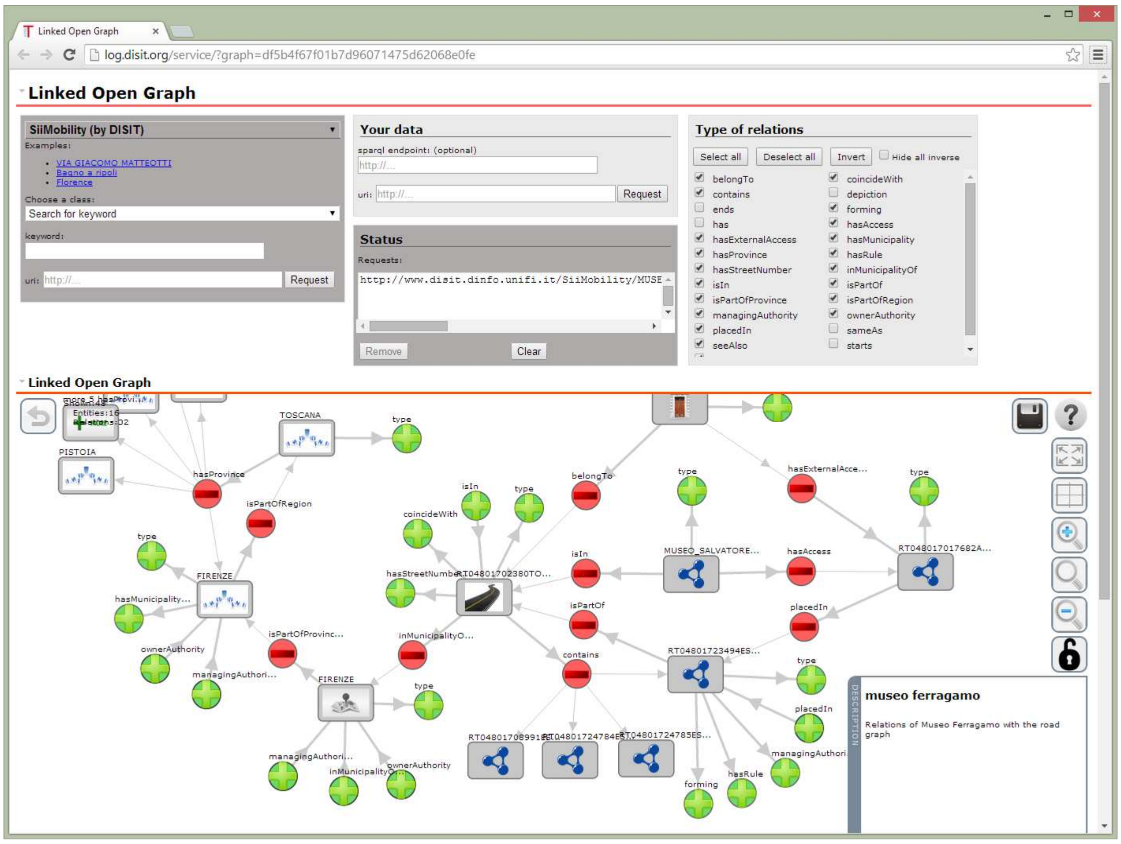
Alla Griglia

Tipologia: ristorante
Email:
Indirizzo: Via dei Banchi, 25/R
Note: Trappetti con pane tostato e pasta fresca Straccetti della casa costata disossata a fettine con funghi porcini aglio rosmarino e olio
LINKED OPEN GRAPH

Figure 106 - LOG button on ServiceMap

The *Linked Open Graph*, *LOG* [Bellini, Nesi, Venturi, 2014], is a web tool for collaborative browsing and navigation on multiple SPARQL entry points. The LOG tool is free to be used, and it has been adopted in multiple projects as *ECLAP* [Bellini et al., 2013B] for cultural heritage (http://www.eclap.eu), to create the *Social Graph* of the social network, *Sii-Mobility* for smart city and *ICARO* for smart cloud ontology analysis. It has been validated using multiple public accessible RDF stores such as: *dbPedia*, *Europeana*, *Getty Vocabulary*, *Camera and Senato*, *GeoLocation*, etc., putting in evidence the different cases and usage of *LOG* tools in the different scenarios, with a specific stress on the analysis of multiple RDF stores on the same graph.
A tool for browsing LD/LOD selecting relationships among URI elements and their attributes, can be a solution for developer for data and knowledge engineers. Therefore, services that allow to insert URI of LOD to navigate on their structure, are very important and the graph may bring to other connected RDF stores, via their definitions in terms of LD.
Technically, not all ontologies and RDF models and stores have been developed by using the same methods, since they have been developed by different teams, using different styles, in different periods, and exploiting different vocabularies. This implies that different approaches to model the same entities and patterns may be possible, as well as different usage of "*sameAs*", "equivalent class", blank nodes, reuse of vocabulary and concepts.
The access and browse to a RDF store via the SPARQL entry point is a way to understand the knowledge base and the relationships among the included entities. In some cases, the entities/URIs (*URI(a)*, *URI(b)*) of different RDF stores (accessible via different SPARQL entry points: *URL(a)* and *URL(b)*) may be connected each other. Typically the connection can be via URI representing classes of common ontologies and

definitions. The visualization of graphs associated with *URL(a),URI(a)* and *URL(b),URI(b)* on the same screen may allow to put in evidence the relationships among these two graphs. They may be the basis for (i) integrating the two ontologies, for federating RDF storage, (ii) understand differences and relationships, and/or (iii) for creating additional connections. For example, by creating an *owl:sameAs* relationship among two entities that represent the same concept in the two models. In some cases, they have not been intentionally defined by using the same vocabulary since they are different for somehow, while in other context they should be the same, otherwise deductions in the knowledge base would not take into account all needed facts.


Linked Open Graph

log.disit.org/service/?graph=df5b4f67f01b7d96071475d62068e0fe

SiiMobility (by DISIT)

Examples:
- VIA GIACOMO MATTEOTTI
- Bagno a ripoli
- Florence

Choose a class: Search for keyword

keyword:

uri: http://...

Your data

sparql endpoint: (optional) http://...

uri: http://...

Status

Requests:

```
http://www.disit.dinfo.unifi.it/SiiMobility/MUSE
```

Type of relations

Hide all inverse ☐

✓ belongTo	✓ coincideWith
✓ contains	☐ depiction
☐ ends	✓ forming
☐ has	✓ hasAccess
✓ hasExternalAccess	✓ hasMunicipality
✓ hasProvince	✓ hasRule
✓ hasStreetNumber	✓ inMunicipalityOf
✓ isIn	✓ isPartOf
✓ isPartOfProvince	✓ isPartOfRegion
✓ managingAuthority	✓ ownerAuthority
✓ placedIn	☐ sameAs
✓ seeAlso	☐ starts

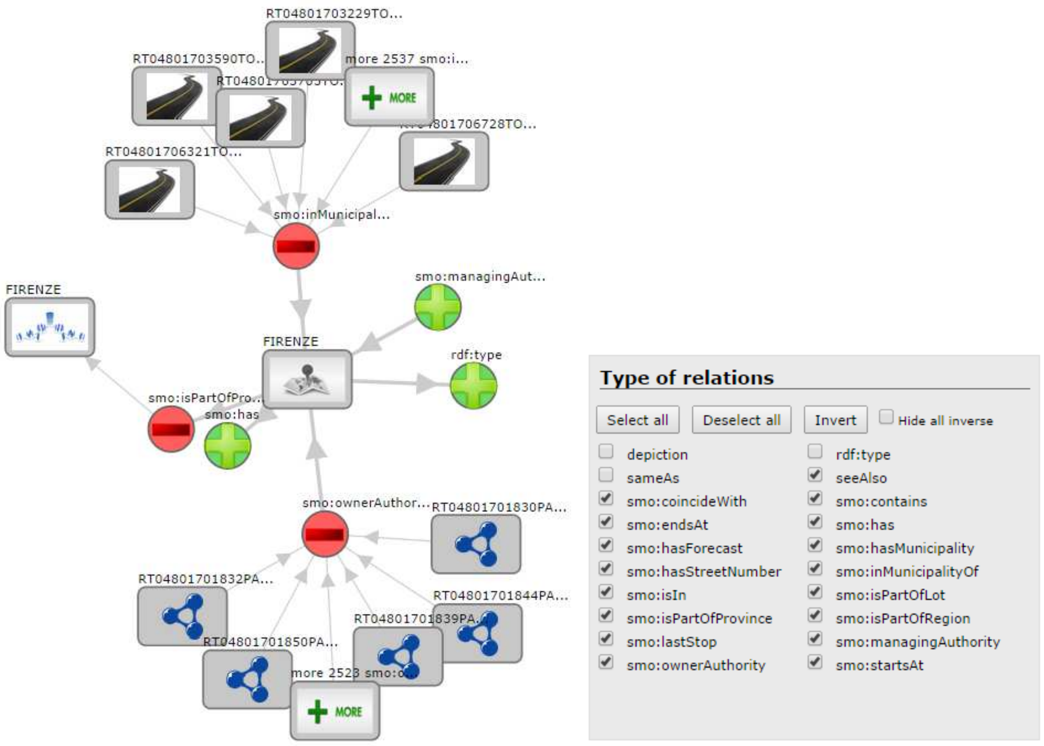
Linked Open Graph

museo ferragamo

Relations of Museo Ferragamo with the road graph

Figure 107 - LOG interface

The graph is populated by nodes and edges: nodes can be of two types, which may represent entities (rectangular shaped nodes) as content, terms, users, etc. or relations between resources (circular shaped nodes); through directed edges the elements and their relationships are linked. Examples of relations are shown in Figure.



Type of relations

Select all | Deselect all | Invert | ☐ Hide all inverse

☐ depiction	☐ rdf:type
☐ sameAs	☑ seeAlso
☑ smo:coincideWith	☑ smo:contains
☑ smo:endsAt	☑ smo:has
☑ smo:hasForecast	☑ smo:hasMunicipality
☑ smo:hasStreetNumber	☑ smo:inMunicipalityOf
☑ smo:isIn	☑ smo:isPartOfLot
☑ smo:isPartOfProvince	☑ smo:isPartOfRegion
☑ smo:lastStop	☑ smo:managingAuthority
☑ smo:ownerAuthority	☑ smo:startsAt

Figure 108 - Relationships filtering, i.e. Hide/show types of relations to reduce the graph complexity

The LOG.disit.org service is not a simple browsing of related resources; the visual browsing of SPARQL entry points is not a simple task, especially if this work is performed by a Web Application. In particular, specific algorithms are needed to cope with complexity of obtaining and processing complex reticular structures with web based applications, removing duplications, managing multiple entry points, generating complex SPARQL queries, etc. Furthermore it has a high number of demanded features that transform the LOG into a great competitor, if comparing them with representative state of the art solutions. Here below, a list of its most desirable features is reported:

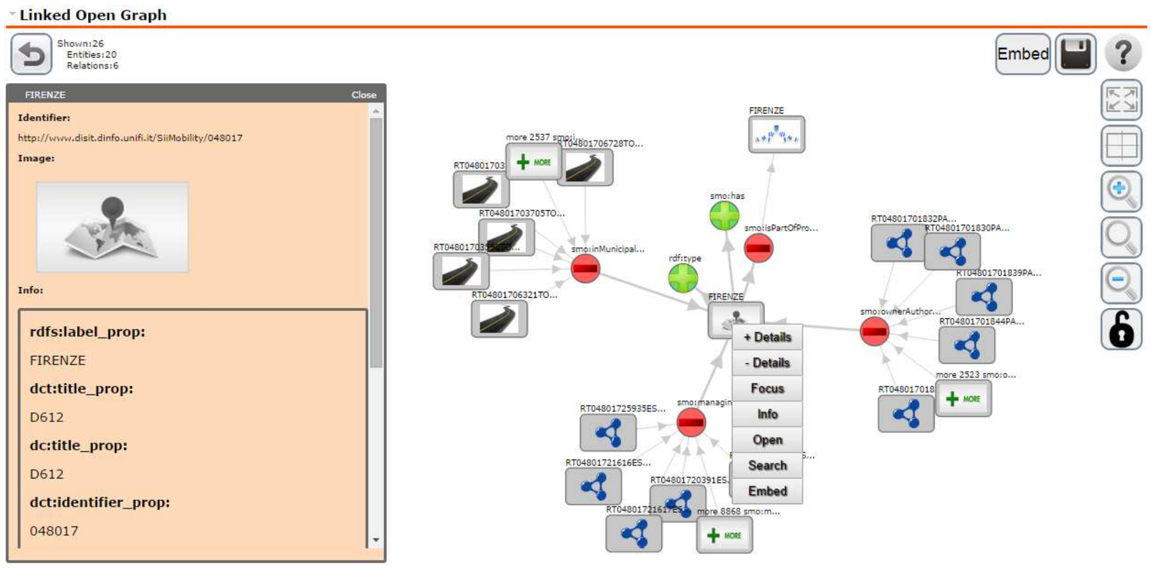
- **Access and rendering of LD**: the visual tool should be capable to represent a LD which is publically accessible as a URI, providing a set of triples.
- **Access and rendering URI from SPARQL entry point**: a visual tool for browsing SPARQL entry points extract the results by using a couple {*URL(i), Q*}, where Q is the semantic query or an URI.
- **Managing Entry Points with different URL in URI**: the visual tool has to be capable to accept to start browsing from the couple URI, URL having different domains.
- **Multiple SPARQL entry points**: the visualization of graphs associated with URL(a),URI(a) and URL(b),URI(b) on the same screen may allow to put in evidence the relationships among these two graphs.

- **Making keyword based query**: in order to identify a starting URI for RDF graph rendering it could be possible to pose a keyword-based query on the RDF store. This feature is not always available on the RDF store (SPARQL entry points), and may be implemented in several different manners.
- **Inspecting entry point for searching classes**: a textual search can be performed on the instances of one or more of those classes, in order to get back a list of entities/URI from which the graph visual browsing can start.
- **Showing relationships, turning on/off, singularly or globally**: once the first URI and related URIs are shown several relationships may be present in the graph, maybe hundreds or thousands. In any case, the users should be enabled to turn on/off some of the relationship categories to make the graph more readable and focused on the entities and relationships under analysis.
- **Representing relationships (managing complexity)**: in the rendering of the RDF graph, a large number of entities (URI) and the relationships among them may be present. The high number of graphical elements can be reduced allowing closing/opening, expanding/compressing relations, filtering some relationships from the visualization and may be also graphically representing entities and relationships by using coded styles.
- **Discovering inbound/outbound relationships, URI and queries**: in some tool, the contextualized text of the query declined for a specific entity is accessible. It can be very useful for training the users in using the SPARQL and for shortening the data exploitation in external applications accessing to the SPARQL entry point API.
- **Undo actions performed, "back"**: in the RDF visual graph manipulation, the possibly of undoing the actions performed with a back buttons may be very useful, together with the possibility of saving the reached status.
- **Save and Load LOD graphs**: a very valuable feature is the possibility of saving the status of the graph with all its linked URIs, and the relationships exploded (taking into account their on/off status). This graphical context should be the starting point for further analysis and not a simple image snapshot.
- **Share and collaborative LOD graphs**: among the major tools detectable on the web, only LOG.DISIT provide this collaborative feature on LOD RDF graphs. LOG.DISIT allows to share the RDF graph as web data on the cloud, in read and read/write modalities
- **URI attributes (showing info or an URI)**: a number of attributes/values (literal) may be associated with the URI. These data should be accessible without involving graph representation.
- **URL to resources**: an URI may have among its attributes some URL to external digital resources. These URL should be accessible for opening the digital resources into the browser or for download.
- **Representing entities**: in complex LOD graph the fast identification of URI type is very important. The URI can be represented by using specific icons on

the basis of their: (i) type (problems in the case of multiple types), (ii) information and attributes, (iii) specific icon associated with the URI (e.g., image of the person for dc:author), (iv) specific case, for example to represent the Blank nodes.

Thanks to this tool the *Km4city* based triplestore can be browse, starting from a resource and exploring the entire graph, by following the various related resources.
By analyzing the working environment, in the upper left corner of the main screen, there is the *BACK* button, which allows to go back to previous states of the graph (e.g. after a focus). On the other hand however, all the main buttons are collected, that corresponds to the user utility: starting from the top is possible to find the *EMBED* button, which opens the dialog box in Figure 111 that allows to retrieve the code to integrate the LOG in any site; the next button is the *SAVE* button, useful to save the Linked Open Graph status and share it by providing a valid email address on which, in a short time, a mail will be receive containing a link that could allow to access at the LOG and share it with friends. The round button marked with a question mark, instead, allows to access to the tools *Help*, while the button just below enables/disables switching to the full screen mode. The button similar to the four rectangles, allows to re-center the whole graph and the last three buttons, are the zoom controls, respectively, zoom in button, default zoom button and zoom out button.
Moreover, clicking the mouse right button on a node, a navigation menu is displayed, which allows to perform the following functions:

- **Expand** an entity node with its relations adding them to the graph;
- **Focus** on an entity, in this case the graph is cleared and only the focused node is shown with its relations;
- **Open**, that is the play of the page or content associated with the node (e.g. associated DataProperties to a specific class of *Km4City* ontology);
- **Zoom/Pan** the view;
- A special node is the "**More**" node that is presented when in a relation are present many nodes (e.g., the content associated with a group).
- **Save** and **share** graphs and share with other colleagues, avoid duplicated links, **explore inbound and outbound** relationships, **navigating on OD and LOD** in a transparent manner.
- Work on preferred **relationships**.
- **Search**, that allows to search the preferred entities and URI into a set of SPARQL databases;



FIRENZE Close

Identifier:

http://www.disit.dinfo.unifi.it/SiiMobility/048017

Image:

Info:

rdfs:label_prop:

FIRENZE

dct:title_prop:

D612

dc:title_prop:

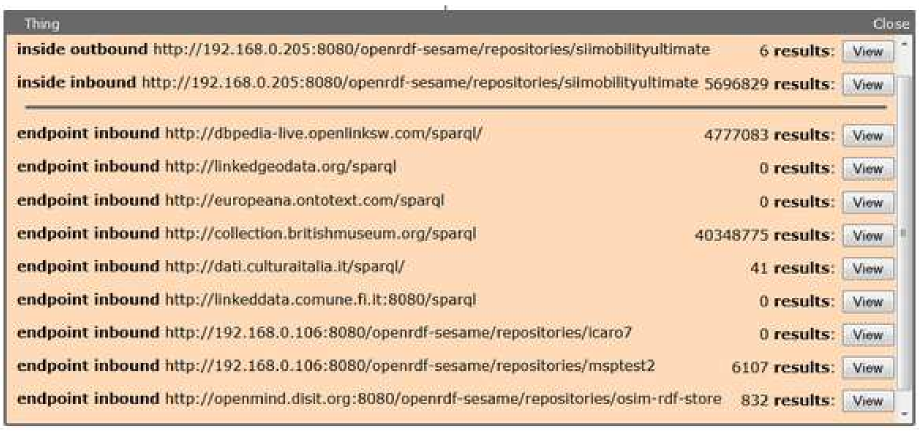
D612

dct:identifier_prop:

048017

Figure 109 - Info Tab for an entity

The following figure reports an example of a *Search* for preferred entities and URI into a set of SPARQL databases with billions and billions of triples.

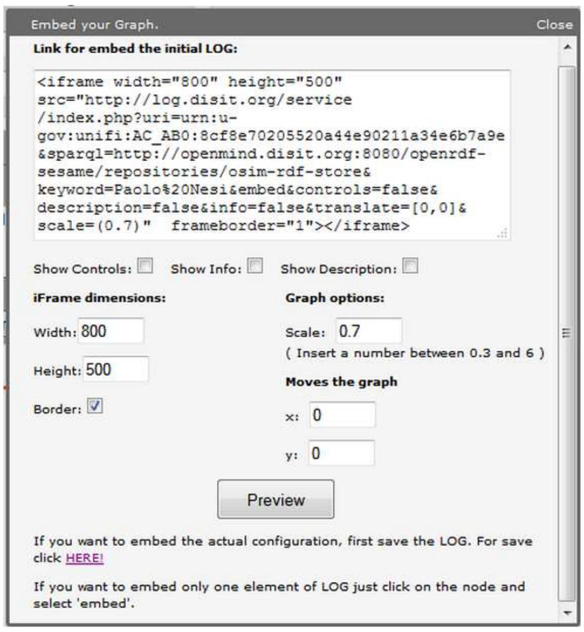


Thing Close

inside outbound http://192.168.0.205:8080/openrdf-sesame/repositories/siimobilityultimate	6 **results:**	View
inside inbound http://192.168.0.205:8080/openrdf-sesame/repositories/siimobilityultimate	5696829 **results:**	View
endpoint inbound http://dbpedia-live.openlinksw.com/sparql/	4777083 **results:**	View
endpoint inbound http://linkedgeodata.org/sparql	0 **results:**	View
endpoint inbound http://europeana.ontotext.com/sparql	0 **results:**	View
endpoint inbound http://collection.britishmuseum.org/sparql	40348775 **results:**	View
endpoint inbound http://dati.culturaitalia.it/sparql/	41 **results:**	View
endpoint inbound http://linkeddata.comune.fi.it:8080/sparql	0 **results:**	View
endpoint inbound http://192.168.0.106:8080/openrdf-sesame/repositories/icaro7	0 **results:**	View
endpoint inbound http://192.168.0.106:8080/openrdf-sesame/repositories/msptest2	6107 **results:**	View
endpoint inbound http://openmind.disit.org:8080/openrdf-sesame/repositories/osim-rdf-store	832 **results:**	View

Figure 110 - Search preferred entities tab

LOG can also be embedded into a WEB pages. On this regard it is possible customize the buttons and the actions allowed at users on the embedded version of the *LOG.disit.org* tool instance, and on which segment of graph starting.


Embed your Graph. Close

Link for embed the initial LOG:

```
<iframe width="800" height="500"
src="http://log.disit.org/service
/index.php?uri=urn:u-
gov:unifi:AC_AB0:8cf8e70205520a44e90211a34e6b7a9e
&sparql=http://openmind.disit.org:8080/openrdf-
sesame/repositories/osim-rdf-store&
keyword=Paolo%20Nesi&embed&controls=false&
description=false&info=false&translate=[0,0]&
scale=(0.7)"  frameborder="1"></iframe>
```

Show Controls: ☐ Show Info: ☐ Show Description: ☐

iFrame dimensions:

Width: 800

Height: 500

Border: ☑

Graph options:

Scale: 0.7

(Insert a number between 0.3 and 6)

Moves the graph

x: 0

y: 0

Preview

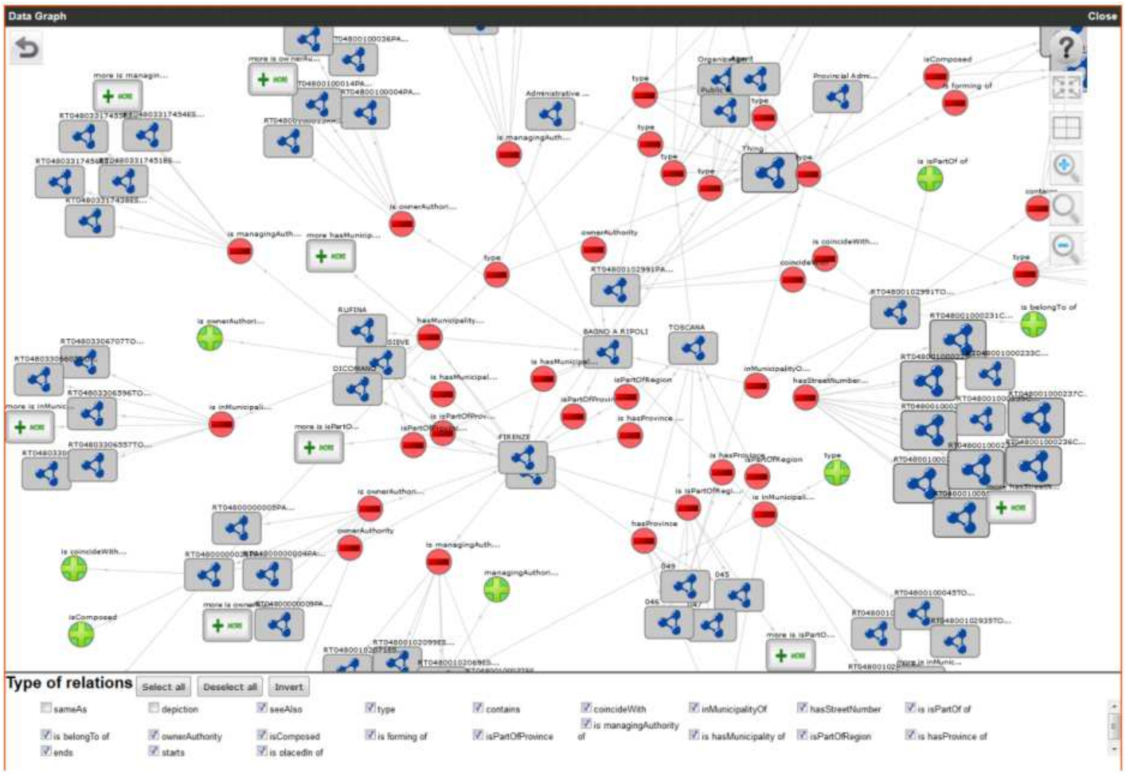
If you want to embed the actual configuration, first save the LOG. For save click HERE!

If you want to embed only one element of LOG just click on the node and select 'embed'.

Figure 111 - Embed code for the LOG

to verify which the type of operation, users operate on LOG tools, a user's interaction analysis of the *ECLAP* [Bellini, Nesi, Serena, 2014] social graph and of the whole portal, has been carried out. Its results shown that only the 5.8% of the unique users interacted with the social graph, and the most requested operation are to *Open* a node (43%, for example to access at a recommendation, to see the content of other users), then to *Expand* a node (29%, mainly a media object 17%) and then to see the *More* related content (18%), the *Focus* operation is at about 10% on the operations requested since the social graph was activated (2013-01-29) until the mid of September 2013.

Concluding, the LOG can be very useful to understand the differences interactively studying the RDF store from remote, to learn and to explore the possibility of reusing and connecting them each other. In fact, the visual browsing of SPARQL entry point can help to analyze the RDF store reticular structure, that is at the basis of the ontology and the related instances of predicates contained. The *LOG.disit* tool, with its additional features, with respect to the state of the art browsing tools such as *LodLive* (http://lodlive.it/) and *Gruff* (http://franz.com/agraph/gruff/), can be a very useful tool for: analyzing RDF stores and models, comparing and discovering connections and relationships among RDF stores and models, discovering eventual problems in accessible knowledge base for their future reuse and connection.



Type of relations Select all Deselect all Invert

☐ sameAs ☐ depiction ☑ seeAlso ☑ type ☑ contains ☑ coincideWith ☑ inMunicipalityOf ☑ hasStreetNumber ☑ is isPartOf of
☑ is managingAuthority of
☑ is belongTo of ☑ ownerAuthority ☑ isComposed ☑ is forming of ☑ isPartOfProvince ☑ is hasMunicipality of ☑ isPartOfRegion ☑ is hasProvince of
☑ ends ☑ starts ☑ is placedIn of

Figure 112 - A portion of the Road Graph views with LOG

Moreover, despite the first impression, the representation of an RDF reticular structure and thus its access are not a simple neither superficial task.

The *Linked Open Graph* allows to display and browse the structure and relations among the RDF entities, for this reason, within the project presented in this thesis, a tool like the *LOG*, can be useful, for example, to (i) discover and understand the model and the information associated to a given service in the city, (ii) discover connections and similarities among different open data set of public administration, (iii) study the integration of open data with geographic information.

Chapter 6

6. System Evaluation

The system evaluation process aims at assessing the efficacy and effectiveness of the experiment in achieving the objective of offering services and tools for the efficient ingestion and querying of SmartCity data, in order to allow an easy interconnection of this large amount of information into an RDF store and its exploitation. The evaluation is distinguished in nine parts:

- *Qualitative Evaluation*: initially problems encountered on datasets will be examined, mainly related to their data quality.
- *Quality Improvement metrics*: this assessment is to verify the Quality Improvement phase impact (Phase II), implemented within the architecture presented in this dissertation, and in trying to measure how much benefit actually entailed its implementation.
- *Quality metrics*: in this section will be evaluated some metrics that help to define the quality of available datasets, before and after the Quality Improvement phase.
- *Reconciliation evaluation*: as seen in Section 5.5, two approaches to data reconciliation at geographical level, have been tested; in this section the results of both approaches will be compared to determine if the two methods are interchangeable.
- *Triples loading assessment*: this section is devoted to verify the number of triples, properly read by the RDF management system, in order to identify any problems on data or errors in the mapping process.
- *Validation results*: the validation process is a means by which it is possible to check correctness of data and of the chosen ontological model; this section is dedicated to verify the correct interpretation of data through the ontology Km4City.
- *Interconnection evaluation*: one of the research project purposes, is to semantically interconnect the ingested dataset; so, this section is dedicated to verify the interconnections, that have been created between datasets, thanks to the implemented architecture.
- *Volume measure*: as mentioned earlier, the processed volume of data and their difference in size and speed, allows us to place the project in the Big Data field;

it is evident, therefore, that an analysis on triplestore size and its growth, have playing an important part.

- *Time response evaluation*: the last part of the chapter, is devoted to presentation of execution times measured for processes related to Ingestion and Mapping phases (respectively Phase I and III).

6.1 Qualitative Evaluation

The most used tools in the field of qualitative data analysis, are observation and content analysis. This evaluation section is dedicated to the content analysis of part of the datasets involved in the research project, presented in this dissertation.
To design processes that realize the quality improvement of various attributes belonging to services entity, it was necessary to carry out a detailed analysis of datasets to be treated; below the main problems found inside these datasets, are listed:

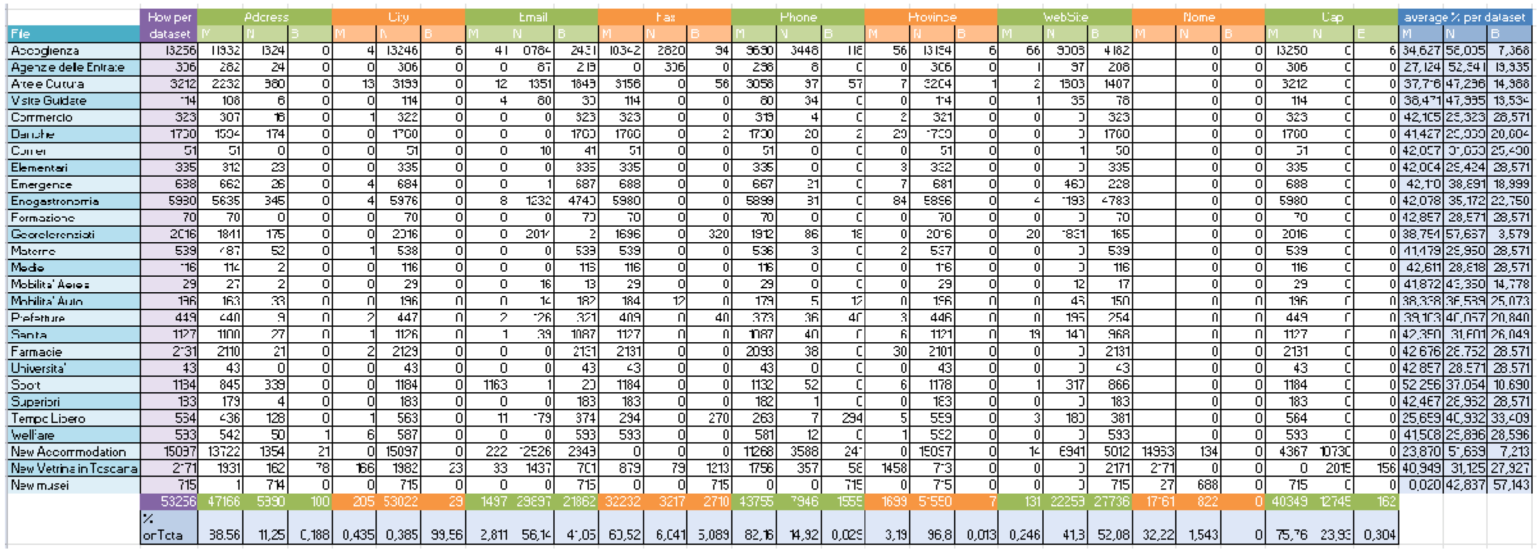
- Presence of alphabetic/special characters in number field (such as / - blank spaces);
- Numbers written in accordance with the exponential notation;
- Lack of 0 before the regional/provincial prefix in telephone/fax numbers, lack of international prefix 0039 or +39;
- Presence of multiple telephone numbers (entire number or only suffixes extension number);
- Partial/incomplete numbers;
- Address and house number stored in one field (comma separated or blank space separated);
- Address and house number stored in one field with additional text;
- Generic text without any information;
- Address with double house number (terminating with a character, or comma separated);
- Address double house number (without a comma);
- Information regarding Locality or Fraction stored in a wrong field;
- Lack of/incomplete suffix (it,com,...) after dot in website address or email address;
- Lack of/incomplete suffix (it,com,...) and the dot preceding it;
- Lack of @ or double @ in email address;
- Illegal Characters (blank space included);
- Triple /, double http://, lack of domain, : after www, double dot after www in website address;

- False positive email address or web site address (correct syntax but not existing);
- Presence of accented letters, @, apostrophe, / or : before domain in website address.

6.2 Quality Improvement metrics

One of the objectives, defined during the design phase, is to perform the data ingestion by limiting the amount of lost data; to this purpose, in fact, the Quality Improvement phase has been also implemented. To verify the efficiency of this phase, that is, to measure the quantity of data lost, due to errors that can afflict the input data (see Section 5.2), datasets relating to services in Tuscany was selected, because they are evidently affected by a number of errors.
This dataset was then subjected to a double mapping in triple: the first made with data previously submitted to the Quality Improvement process (Phase II in Figure 25), and the second made instead on data as they have been received, without subjecting them to any correction process, that is, directly at the end of Phase I (Figure 25).
This study had the objective of evaluating the number of errors that the system is able to identify thanks to Quality Improvement process and the number of additional triples that such corrections allow to generate.
To complete this first phase of system evaluation, a Kettle transformation was then made, which takes data directly from the first HBase repository of the architecture, i.e. the repository where data are inserted at the end of Phase I (see Figure 25), and applied on them the mapping process, corresponding to Phase III.
This transformation then saves triples created in a different location than where the architecture saves triples after Phase III, in order to keep well separated the experiments. The two different N3 triples files created, are then compared to verify if there are some differences in the total number of triples created.
Both methods were applied to services data set, consisting of 27 different Open Datasets coming from the Open Data portal of the Tuscany region. Data collected through the Quality Improvement phase, are collected in the following table.
As it is possible to see from Table 20, in which are counted all actions implemented by the quality improvement processes on each datasets examined, the data set with the largest number of empty cells, that is the dataset with the highest number of missing data, is related to museums. Instead, the data set in which the highest number of corrections were made, is the dataset related to sports facilities. The most complete data set, that is, the one with the lowest percentage of missing data, is the georeferenced accommodation dataset and, finally, the data set on which the step of Quality Improvement acted a smaller number of times, that is the dataset containing the most correct data, concerning accommodation.

Table 20 - Results of the Quality Improvement process



	How per	Address			City			Email			Fax			Phone			Province			WebSite			Nome			Cap			average % per dataset		
File	dataset	M	N	B	M	N	B	M	N	B	M	N	B	M	N	B	M	N	B	M	N	B	M	N	B	M	N	E	M	N	B
Accoglienza	13256	11932	1324	0	4	13246	6	41	10784	2431	10342	2820	94	9690	3448	118	56	13194	6	66	9008	4182		0	0	13250	0	6	34,627	58,005	7,368
Agenzie delle Entrate	306	282	24	0	0	306	0	0	87	219	0	306	0	298	8	0	0	306	0	1	97	208		0	0	306	0	0	27,124	52,941	19,935
Aree Cultura	3212	2232	980	0	13	3199	0	12	1351	1849	3156	0	56	3058	97	57	7	3204	1	2	1803	1407		0	0	3212	0	0	37,776	47,236	14,988
Visite Guidate	114	108	6	0	0	114	0	4	80	30	114	0	0	80	34	0	0	114	0	1	35	78		0	0	114	0	0	38,471	47,995	13,534
Commercio	323	307	16	0	1	322	0	0	0	323	323	0	0	319	4	0	2	321	0	0	0	323		0	0	323	0	0	42,105	29,323	28,571
Banche	1700	1504	174	0	0	1700	0	0	0	1700	1700	0	2	1700	20	2	20	1700	0	0	0	1700		0	0	1700	0	0	41,427	[illegible]	20,004
Corrieri	51	51	0	0	0	51	0	0	10	41	51	0	0	51	0	0	0	51	0	0	1	50		0	0	51	0	0	42,057	[illegible]	25,490
Elementari	335	312	23	0	0	335	0	0	0	335	335	0	0	335	0	0	3	332	0	0	0	335		0	0	335	0	0	42,004	29,424	28,571
Emergenze	688	662	26	0	4	684	0	0	1	687	688	0	0	667	21	0	7	681	0	0	460	228		0	0	688	0	0	42,110	38,891	18,999
Enogastronomia	5980	5635	345	0	4	5976	0	8	1232	4740	5980	0	0	5899	31	0	84	5896	0	4	1193	4783		0	0	5980	0	0	42,078	35,172	22,750
Formazione	70	70	0	0	0	70	0	0	0	70	70	0	0	70	0	0	0	70	0	0	0	70		0	0	70	0	0	42,857	28,571	28,571
Georeferenziati	2016	1841	175	0	0	2016	0	0	2014	2	1696	0	320	1912	86	18	0	2016	0	20	1831	165		0	0	2016	0	0	38,754	57,667	3,579
Materne	539	487	52	0	1	538	0	0	0	539	539	0	0	536	3	0	2	537	0	0	0	539		0	0	539	0	0	41,479	29,950	28,571
Medie	116	114	2	0	0	116	0	0	0	116	116	0	0	116	0	0	0	116	0	0	0	116		0	0	116	0	0	42,611	28,818	28,571
Mobilita' Aerea	29	27	2	0	0	29	0	0	16	13	29	0	0	29	0	0	0	29	0	0	12	17		0	0	29	0	0	41,872	43,350	14,778
Mobilita' Auto	196	163	33	0	0	196	0	0	14	182	184	12	0	179	5	12	0	196	0	0	46	150		0	0	196	0	0	38,338	36,589	25,073
Prefetture	449	440	9	0	2	447	0	2	126	321	409	0	40	373	36	40	3	446	0	0	195	254		0	0	449	0	0	39,103	40,057	20,840
Santa	1127	1100	27	0	1	1126	0	1	39	1087	1127	0	0	1087	40	0	6	1121	0	19	140	968		0	0	1127	0	0	42,350	31,601	26,049
Farmacie	2131	2110	21	0	2	2129	0	0	0	2131	2131	0	0	2093	38	0	30	2101	0	0	0	2131		0	0	2131	0	0	42,676	28,752	28,571
Universita'	43	43	0	0	0	43	0	0	0	43	43	0	0	43	0	0	0	43	0	0	0	43		0	0	43	0	0	42,857	28,571	28,571
Sport	1184	845	339	0	0	1184	0	1163	1	20	1184	0	0	1132	52	0	6	1178	0	1	317	866		0	0	1184	0	0	52,256	37,054	10,690
Superiori	183	179	4	0	0	183	0	0	0	183	183	0	0	182	1	0	0	183	0	0	0	183		0	0	183	0	0	42,467	28,962	28,571
Tempo Libero	564	436	128	0	1	563	0	11	179	374	294	0	270	263	7	294	5	559	0	3	180	381		0	0	564	0	0	25,659	40,932	33,409
Welfare	593	542	50	1	6	587	0	0	0	593	593	0	0	581	12	0	1	592	0	0	0	593		0	0	593	0	0	41,508	29,896	28,596
New Accommodation	15097	13722	1354	21	0	15097	0	222	12526	2349	0	0	0	11268	3588	241	0	15097	0	14	6941	5012	14963	134	0	4367	10730	0	23,870	57,639	7,213
New Vetrina in Toscana	2171	1931	162	78	166	1982	23	33	1437	701	879	79	1213	1756	357	58	1458	713	0	0	0	2171	2171	0	0	0	2015	156	40,949	31,125	27,927
New musei	715	1	714	0	0	715	0	0	0	715	0	0	715	0	0	715	0	715	0	0	0	715	27	688	0	715	0	0	0,020	42,837	57,143
	53256	47166	5390	100	205	53022	29	1497	29897	21862	32232	3217	2710	43755	7946	1555	1699	51550	7	131	22259	27736	17761	822	0	40349	12745	162			
	% on Tota	88,56	11,25	0,188	0,435	0,385	99,56	2,811	56,14	41,05	60,52	6,041	5,089	82,16	14,92	0,029	3,19	96,8	0,013	0,246	41,8	52,08	32,22	1,543	0	75,76	23,93	0,304			

Furthermore, in order to make more understandable the numbers reported in the table, it is necessary to make some observations:

- The number of *address* fields modified is high because the Quality Improvement phase involves the separation of the house number from the name of the street, in order to store these two values in two separate fields, to be able to carry out a more simply geolocalization process at street number level.
- *Website* and *email* fields are the ones that most have no values; only a very low services percentage presents this information.
- The modified *telephone numbers* are several, mainly due to the incorrect data export format chosen by who has prepared data: in fact the digit 0 at the start of prefix is not present in almost all the phone numbers; same thing happens to *fax numbers*.
- The amount of modified *cap* by the Quality Improvement process, corresponds to 100% because in no dataset was present such value, and so it was necessary to include the correct value, to complete the address.

As previously mentioned, after the creation of the two triples sets, a comparison was made, and in Table 21 the results obtained has been reported.

File	**New Triples**	**Old Triples**	**Unique Elements**	**%**
Accoglienza	186.217	151.482	13.256	0,187
Agenzie entrate	3.954	3.120	306	0,211
Arte e cultura	41.621	21.979	3.212	0,472
Banche	21.211	17.540	1.768	0,173
Commercio	3.932	3.025	323	0,231
Corrieri	647	520	51	0,196
Elementari	4.022	3.160	335	0,214
Emergenze	8.752	6.889	688	0,213
enogastronomia	77.744	47.609	5.980	0,388
Farmacie	25.648	11.790	2.131	0,540
Formazione	858	700	70	0,184
Georeferenz	33.321	2.287	2.016	0,931
Materne	6.436	3.559	539	0,447
Medie	1.410	1.110	116	0,213
Mobilita' aerea	390	43	29	0,890
Mobilita' auto	2.422	1.979	196	0,183
Prefetture	5.656	4.534	449	0,198
Salute	13.759	10.802	1.127	0,215
Sport	15.252	13.807	1.184	0,095
Superiori	2.214	1.820	183	0,178
Tempo Libero	6.836	6.218	564	0,090
Universita'	532	344	43	0,353

Visite guidate	1.516	1.012	114	0,332
Welfare	7.325	805	593	0,890
Totale	471.675	316.134	35.273	**0,330**

Table 21- Triples counting before and after the Quality Improvement processing

% triple increase	
Media	0,334
Errore standard	0,050
Mediana	0,214
Moda	#N/D
Deviazione standard	0,247
Varianza campionaria	0,061
Curtosi	1,562
Asimmetria	1,589
Intervallo	0,841
Minimo	0,090
Massimo	0,931
Somma	8,025
Conteggio	24
Più grande(1)	0,931
Più piccolo(1)	0,090
Livello di confidenza(95,0%)	0,104

Table 22 - Descriptive statistics of triples increase (%)

Analyzing data in Table 21 it is possible to note that, thanks to the Quality Improvement phase, the triples number increased by 155.541 units, which corresponds to a triples increase of 33%, a high number considering that services corresponds to only 0.6% of all triples contained inside the repository.
Furthermore, knowing that the unique elements of all the datasets are 35.273, it is obtained that, on average, for each individual service, thanks to the Quality Improvement phase, 4,40 triples were added; in fact, from an average of 8,962 triples held for each service without QI, an average of 13,372 triples for each service has been achieve, thanks to Phase II, which means that about 4,40 information has been recovered from the initial dataset.

In conclusion, the impact of the Quality Improvement phase has an average of 33,4%, a median of 21,4% and a variance of 0,061%

6.3 Quality metrics

Another evaluation that can be made on available datasets, involves the measurement of *Completeness*, *Accuracy* and *Consistency*.
Following the method used in [Bellini, Nesi, 2013] for Meta Data assessment, will be briefly presented and applied to the Open Data used during the project.

Commonly the *Completeness* is related to the empty field in a dataset and is generically defined as the degree to which value are present in the attributes that require them [Piprani and Ernst, 2008]. In [Ochoa and Duval, 2006], instead, the definition for the *Completeness* is presented as the degree to which data contains all the information needed to have an ideal representation of the described object. Following the idea of this definition, it is therefore necessary to define, for each class of ontology, which is the information that, ideally, every instance should possess. For example, considering the *km4c:Service* class; in the following table there are fields that ideally we would like that *Service* each instance own, to make information complete as much as possible.

Service DataProperty	Status	Weight
Name	Mandatory	1
Street addess	Mandatory	1
House number	Recommended	0,75
Postal code	Recommended	0,75
City/Locality	Mandatory	1
Province	Optional	0,25
Phone	Recommended	0,75
Fax	Optional	0,25
Email	Optional	0,25
Website	Optional	0,25

Table 23 - Ideal composition of a Km4city:Service instance

To each row in the table has been also associated a weight, which indicates how the information is important for the considered class: *1* indicates that the attribute is Mandatory (because information associated with this DataProperty cannot be retrieved from other sources, if missing), *0.75* indicates instead that it is Recommended (it is possible in some way compensate for the lack of this DataProperty) and finally *0.25* indicates that the attribute is Optional (only information little useful to users or that can be easily retrieved through other sources). In summary, the *Completeness* dimension became function of the weight assigned to the field, according to the importance that it possesses in relation to the realized data interconnection process, and it can be calculated as shown below:

$$f(x) = \begin{cases} 0, if\ the\ field\ is\ empty \\ 1, otherwise \end{cases}$$

$$ComR(y) = \frac{\sum_{i=1}^{nField(y)} f\big(x_i(y)\big) * w_i}{\sum_{j=1}^{nField(y)} w_j}$$

Moreover, as defined by Bruce and Hillman [Bruce and Hilmann, 2004] data should be accurate in the sense of high quality editing, thus it is possible to consider accurate a record when: i) there are not typographical errors in the free text fields; ii) the value in the field are in the format expected. The *Accuracy* in practical, measures how the analyzed field, contains what is expected to contain.

$$g(x) = \begin{cases} 0, if\ an\ accuracy\ issue\ was\ detected \\ 1, no\ problem\ found \end{cases}$$

$$AccR(y) = 1 - \frac{\sum_{i=1}^{nField(y)} g\big(x_i(y)\big) * w_i}{\sum_{j=1}^{nFieldCom(y)} w_j}$$

The Consistency dimension, instead, has to address the logical error; the results of a missed consistency control can affect several fields. Examples are:

- Some services present in the *City* field, the address locality value;
- Inside the *website* field there are an email address.

Some of the Consistency cases are difficult to be detected automatically or required notable computing efforts; for this fact, the calculation of this third quality measure, has been limited to a smaller number of datasets.

$$h(x) = \begin{cases} 0, if\ an\ consistency\ issue\ was\ detected \\ \ \ 1, no\ problem\ found \end{cases}$$

$$ConR(y) = \frac{\sum_{i=1}^{nField(y)} h\big(x_i(y)\big) * w_i}{\sum_{j=1}^{nFieldCom(y)} w_j}$$

After defining the three metrics to be applied, the services dataset were assessed, considering data in Table 23 as an ideal dataset composition (with the corresponding weights).

Dataset	Average	Std Dev	Variance	Min	Max
Accoglienza	0,927	0,111	0,012	0,685	1
Agenzie delle Entrate	0,850	0,276	0,076	0,284	1
Arte e Cultura	0,860	0,213	0,045	0,424	1
Visite Guidate	0,897	0,209	0,044	0,316	1
Commercio	0,792	0,397	0,157	0	1
Banche	0,790	0,396	0,157	0	1
Corrieri	0,814	0,356	0,127	0,020	1
Elementari	0,789	0,396	0,157	0	1
Emergenze	0,858	0,302	0,091	0,001	1
Enogastronomia	0,834	0,316	0,100	0,200	1
Formazione	0,797	0,399	0,159	0	1
Georeferenziati	0,958	0,066	0,004	0,829	1
Materne	0,786	0,395	0,156	0	1

Medie	0,795	0,398	0,158	0	1
Mobilita' Aerea	0,838	0,250	0,063	0,414	1
Mobilita' Auto	0,799	0,334	0,111	0,071	1
Prefetture	0,849	0,249	0,062	0,285	1
Sanita'	0,805	0,361	0,130	0,035	1
Farmacie	0,798	0,399	0,159	0	1
Universita'	0,798	0,399	0,159	0	1
Sport	0,892	0,230	0,053	0,269	1
Superiori	0,795	0,398	0,158	0	1
Tempo Libero	0,740	0,281	0,079	0,324	1
Wellfare	0,782	0,395	0,156	0	1
New Accommodation	0,929	0,112	0,013	0,668	1
New Vetrina in Toscana	0,788	0,314	0,098	0	1
New musei	0,573	0,474	0,225	0	1

Table 24 - Average, standard deviation, variance, minimum and maximum of the Completeness for each dataset

As can be seen from Table 24, the dataset with the smallest *Completeness* is the dataset *Tempo Libero* with a value of 0.740; the most complete file is instead on georeferenced services, with a *Completeness* value equal to 0.958.
In general the fields *Name*, *PostalCode*, *City* and *Province* are those mostly present in each analyzed dataset; on the contrary *WebSite* and *Email* are those most frequently empty.
The two graphs below show the *Completeness* trend of each DataProperty, for both previously mentioned datasets, which have shown the best results.

Tempo Libero	
Media	0,740
Errore standard	0,094
Mediana	0,871
Moda	1
Deviazione standard	0,296
Varianza campionaria	0,088
Curtosi	-1,896
Asimmetria	-0,419
Intervallo	0,676
Minimo	0,324
Massimo	1
Somma	7,402
Conteggio	10
Livello di confidenza(95,0%)	0,212

Georeferenced Service	
Media	0,958
Errore standard	0,022
Mediana	1,000
Moda	1
Deviazione standard	0,070
Varianza campionaria	0,005
Curtosi	0,183
Asimmetria	-1,369
Intervallo	0,171
Minimo	0,829
Massimo	1
Somma	9,578
Conteggio	10
Livello di confidenza(95,0%)	0,050

Table 25 - Descriptive statistics of Completeness distribution for data below to Tempo Libero and Georeferenced Service datasets

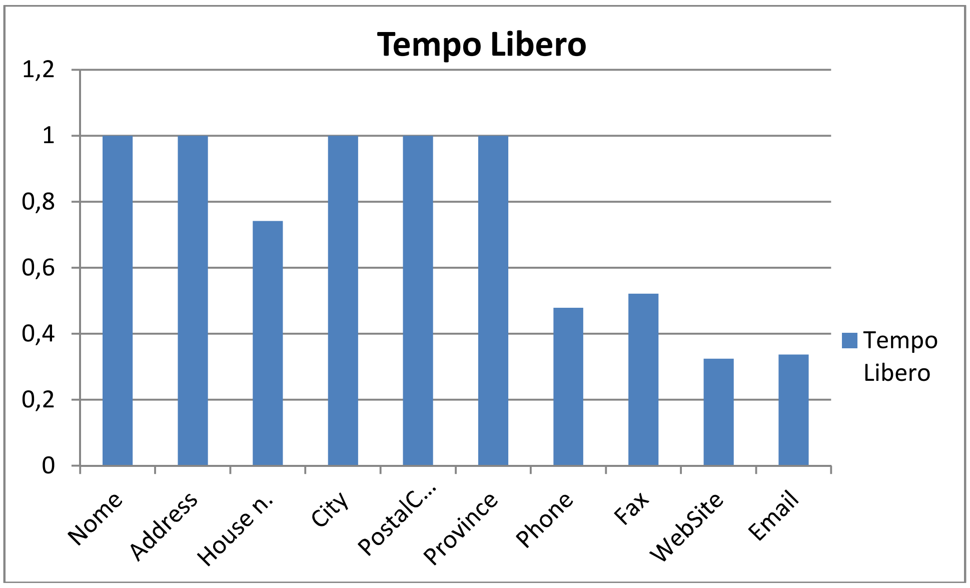


Diagram 6 - Completeness distribution for dataProperties of the "Tempo Libero" dataset

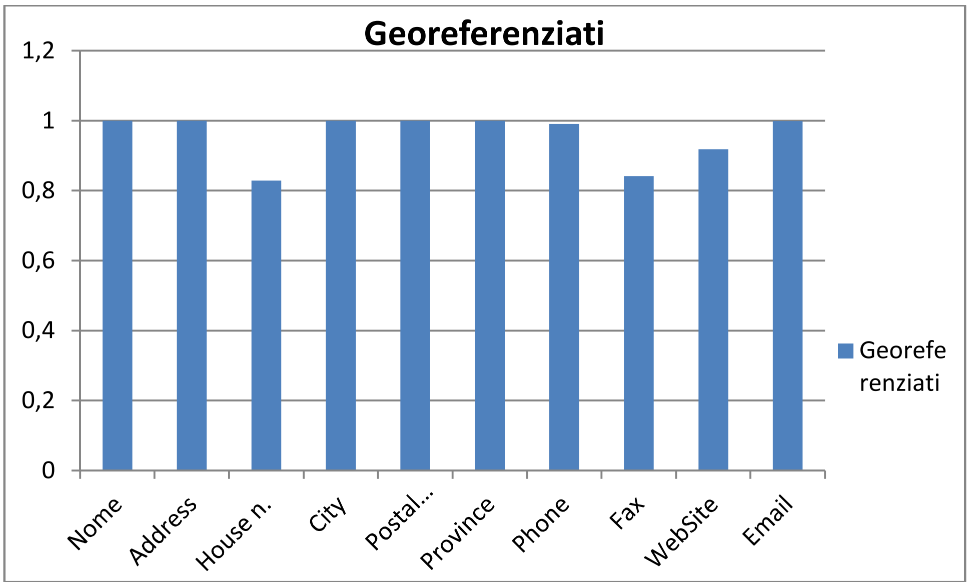


Diagram 7 - Completeness distribution for dataProperties of the "Georeferenziati" dataset

Table 26 is instead dedicated to the *Accuracy* analysis.

Dataset	Average	Std Dev	Variance	Min	Max
Accoglienza	0,167	0,252	0,064	0	0,750
Agenzie delle Entrate	0,162	0,290	0,084	0	0,750
Arte e Cultura	0,225	0,278	0,077	0	0,750
Visite Guidate	0,163	0,253	0,064	0	0,750

Commercio	0,183	0,290	0,084	0	0,750
Banche	0,191	0,286	0,082	0	0,750
Corrieri	0,175	0,297	0,088	0	0,750
Elementari	0,187	0,291	0,084	0	0,750
Emergenze	0,180	0,288	0,083	0	0,750
Enogastronomia	0,185	0,289	0,084	0	0,750
Formazione	0,175	0,297	0,088	0	0,750
Georeferenziati	0,183	0,281	0,079	0	0,750
Materne	0,192	0,288	0,083	0	0,750
Medie	0,178	0,295	0,087	0	0,750
Mobilita' Aerea	0,187	0,291	0,084	0	0,750
Mobilita' Auto	0,196	0,273	0,075	0	0,750
Prefetture	0,164	0,271	0,073	0	0,750
Sanita'	0,177	0,289	0,084	0	0,750
Farmacie	0,176	0,293	0,086	0	0,750
Universita'	0,175	0,297	0,088	0	0,750
Sport	0,247	0,268	0,072	0	0,750
Superiori	0,178	0,294	0,086	0	0,750
Tempo Libero	0,164	0,227	0,051	0	0,750
Wellfare	0,189	0,286	0,082	0	0,750
New Accommodation	0,094	0,169	0,029	0	0,560
New Vetrina in Toscana	0,109	0,174	0,030	0	0,607
New musei	0,250	0,387	0,150	0	0,999

Table 26 - Average, standard deviation, variance, minimum and maximum of the Accuracy for each dataset

In relation to *Accuracy*, dataset that has a lower value of accuracy, that is, presents a greater number of errors, is that related to the museums with an *Accuracy* equal to 0.250, while the dataset with fewer errors is the one with the new data relating to *Accomodation*, with an average accuracy of 0.094. The most accurate dataProperty are *Name*, *City* and *Province*, while, on the contrary, the field *Phone* is the less accurate.

New Musei	
Media	0,250
Errore standard	0,129
Mediana	0
Moda	0
Deviazione standard	0,408
Varianza campionaria	0,166
Curtosi	-0,659
Asimmetria	1,148
Intervallo	0,999
Minimo	0
Massimo	0,999

New Accommodation	
Media	0,094
Errore standard	0,056
Mediana	0,002
Moda	0
Deviazione standard	0,178
Varianza campionaria	0,032
Curtosi	6,068
Asimmetria	2,419
Intervallo	0,560
Minimo	0
Massimo	0,560

Somma	2,498	Somma	0,938
Conteggio	10	Conteggio	10
Livello di confidenza(95,0%)	0,292	Livello di confidenza(95,0%)	0,127

Table 27 - Descriptive statistics of Accuracy distribution for data below to New Musei and New Accommodation datasets

The following graphs shows the *Accuracy* distributions for each DataProperty of the two above mentioned datasets.

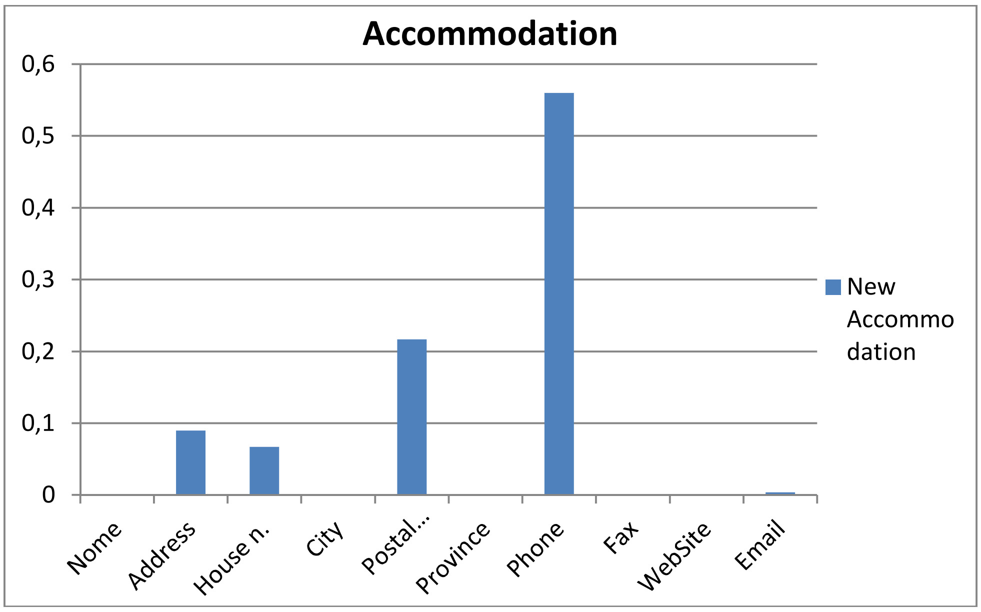


Diagram 8 - Accuracy for each dataProperty of the "Accommodation" dataset

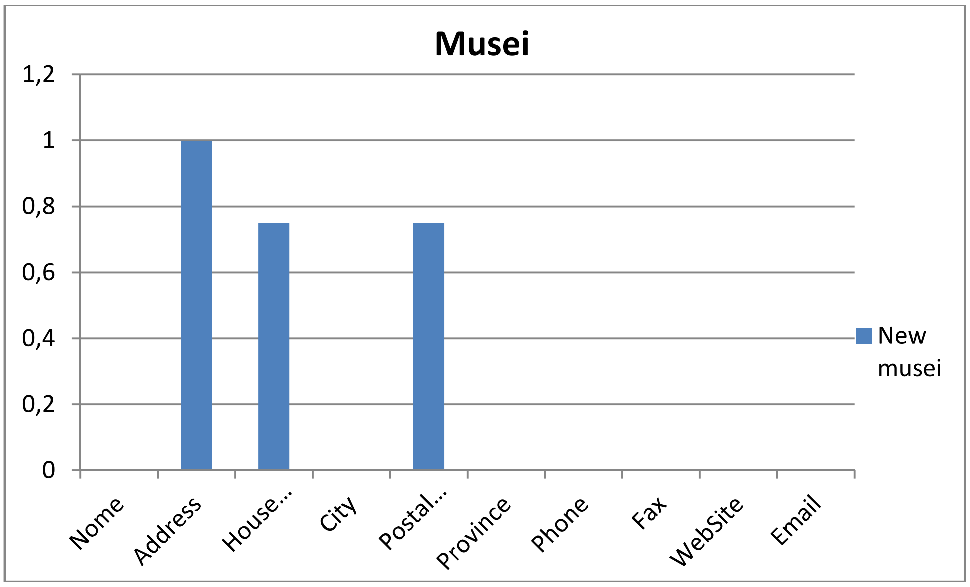


Diagram 9 - Accuracy for each dataProperty of the "Musei" dataset

The following table, show the *Consistency* value obtained, only for the four dataset chosen, calculated both before and after the Quality Improvement phase.

Dataset	Average	Std dev	Variance	Min	Max
Corrieri	0,200	0,350	0,123	0	1
Formazione	0,203	0,348	0,121	0	1
Università	0,200	0,350	0,123	0	1
Mobilità aerea	0,208	0,346	0,119	0	1
		After QI	Phase		
Corrieri	0	0	0	0	0
Formazione	0,010	0,022	0	0	0,071
Università	0	0	0	0	0
Mobilità aerea	0,058	0,148	0,022	0	0,500

Table 28 - Consistency values before and after QI

As is possible to see from Table 28, the Quality Improvement Phase help to reduce the *Consistency* value in most cases. In the following, descriptive statistics for the datasets *Formazione* and *Mobilità aerea*, are shown.

The following diagram show the consistency values calculated for each attribute, both before and after the application of the QI, for datasets *Formazione* and *Mobilità aerea*.

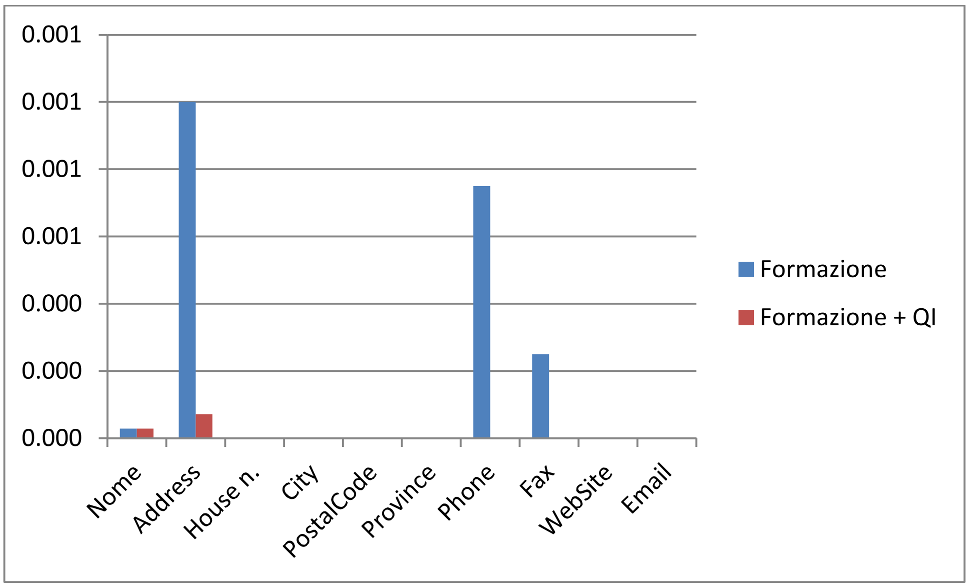


Diagram 10 - Consistency distribution for dataProperties of "Formazione", before and after QI

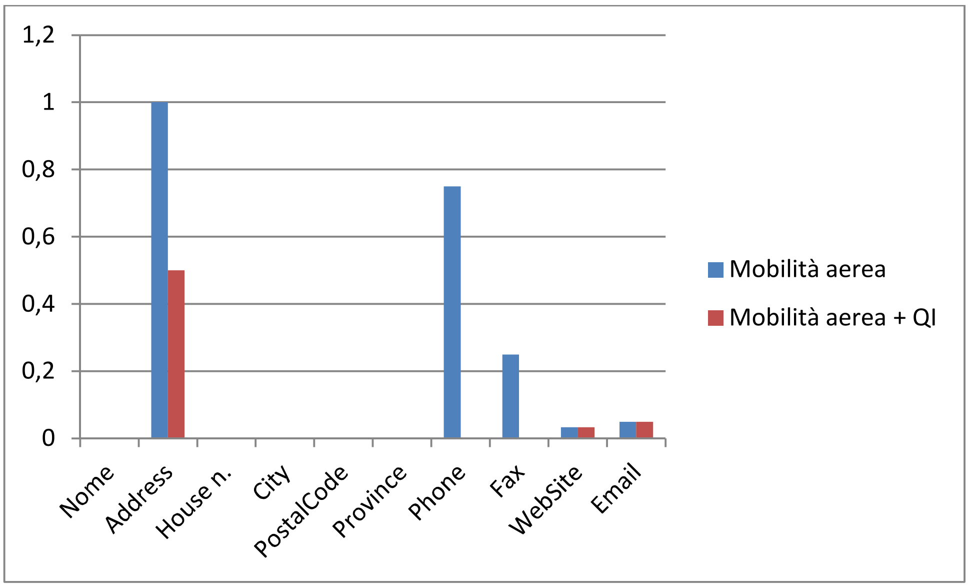


Diagram 11 - Consistency distribution for dataProperties of "Mobilità Aerea", before and after QI

6.4 Reconciliation evaluation

In this Section, a comparison between the two reconciliation methods presented in Section 5.5, is provided, to see which allows to obtain a greater number of better reconciliated services, both at street level and at street number level.
In fact, after being loading and indexing into the RDF store, a dataset may be connected with the other, if their entities refer to the same triples; the term reconciliation refers to

the process of verification and link RDF data that represent the same object on two different dataset, but which are not connected, due to some inconsistencies in their representation.
For the work of this thesis, we particularly focused on interconnecting services to the *Street Guide and Rail Network* macroclass of the Km4City Ontology.

Table 29 shows the list of Open Data that were ingested, until the writing time of this thesis, but new ingestion processes dedicated to other Open Data sets, are under development (for more details see Section 7.1).

Name	Format	Name	Format
Arte e cultura	Csv	Strutture ricettive	csv
Banche	Csv	Strutture ricettive georeferenziate	csv
Corrieri espresso	Csv	tempo libero	csv
Emergenze	Csv	uffici vati	csv
Enogastronomia	Csv	ubiversita' e conservatori	csv
Farmacie	Csv	visite guidate	csv
Imprese del Commercio	Csv	welfare	csv
Infrastrutture aeree	Csv	Accessi sportello suolo pubblico e taxi	csv
Scuale dell'infanzia	Csv	Delibere	csv
Scuola elementare	Csv	Arrivi turistici	csv
Scuola media	Csv	Ataf	csv
Scuola superiore	Csv	Linee Tram	kmz
Corsi di Lingue e di formazione	Csv	Sinistri per via	csv
Sport	Csv	Veicoli circolanti	csv
Previsioni meteo	Csv	Vetrina toscana - botteghe	csv
Salute e sanita'	Csv	Strutture ricettive nuovo dataset	csv
Servizi epr il trasporto su strada	Csv	musei	xml
Servizi vari	Csv	POI dell'osservatorio dei trasporti	csv
Luoghi Freschi a Firenze	Kmz	gate ZTL	kmz
Punti vendita biglietti ATAF	Kmz		

Table 29 - Open Data already ingested

In order to compare the two reconciliation methods applied, namely SPARQL reconciliation and Silk reconciliation, a count of reconciliation triples has been first made, created thanks to the two methods applied.
In Table 30 the results obtained in the different SPARQL reconciliation steps, are reported.
For the validation, a total amount of services/points of interest inserted into the repository has been of 30.182 instances. Among these, 13.185 have been reconciled at street number-level, while the number of elements reconciled at street-level has been 21.207.

No. Step	Method	No. hasAccess Triple created	No. isIn Triple created

1st Reconciliation Step	Exact Search	5.639	8.349
2nd Reconciliation Step	Exact Search + Support Table	1.743	6.996
3rd Reconciliation Step	Last Word Search	5.206	(some duplicates) 5.435
4th Reconciliation Step	Google GeoCoding API	597	527
	Total	13.185	(no duplicates) 21.207

Table 30 - Results of all SPARQL reconciliation steps applied

In the collected data sets, an average of about the 15% are automatically connected entities since they refer to perfectly consistent locations (i.e., perfect match in terms of location, street and civic number) in the MIIC with respect to the description reported in the service data set. In the total of location entities ingested, 5,75% of locations are wrong and not reconcilable due to (i) the presence of wrong values for streets and/or locations, (ii) the lack of a consistent reference location into the MIIC geographical model.

Relating instead to the automatic reconciliation with Silk, as a result of the tests, to verify the best combination of variables for each distance rule that it was decided to test, in order to compare the results of the two reconciliation approaches analyzed, a work of reverse engineering has been done, to assess how many triples represent the result of a manual verification and delete them.
An additional test requires that rules, taking into account all the knowledge relating to possible errors; ad-hoc rules have been created to remove accents, dots, unnecessary words (i.e. *location*), and any other type of error detected, thanks to manual checking. For this special rule, the letter *k* (representing the word knowledge) has been added in the format name.

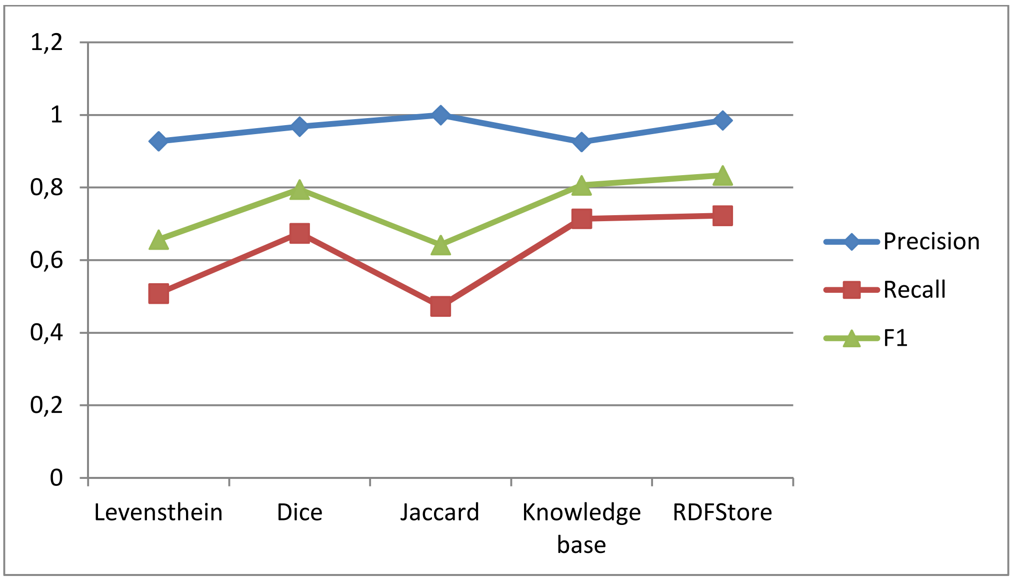


Diagram 12 - How Precision, Recall, F1 change when the comparison function changing; rules [cl50_l00_50, ctd03_l00_20, cth20_l00_50, k_cl20_l00_50]

Method	Precision	Recall	F1
SPARQL –based reconciliation	1,00	0,69	0,820
SPARQL -based reconciliation + manual action	0,985	0,722	0,833
Link discovering – Leveisthein	0,927	0,508	0,656
Link discovering – Dice	0,968	0,674	0,794
Link discovering – Jaccard	1,000	0,472	0,642
Link discovering - Knowledge base + Leveisthein	0,925	0,714	0,806

Table 31 - Reconciliation comparison results

The obtained results are reported in Table 31. The table reports the results assessed in terms of Precision, Recall and F1 score [Powers, 2011], in identifying the correct entities to be reconciliated.
The first two lines refer to the *SPARQL* approach, with and without manual intervention as described in Section 5.5.1. The manual intervention has strongly improved the Recall. On the other hand, the *SPARQL* approach is very time intensive for the programmers since a set of specific queries have to be produced for each data set, to be reconciled.
The second part of Table 31 reported the results related to different implementations of link discovering *SILK* based solutions, by using different string distances (i.e., *Leveisthein*, *Dice*, and *Jaccard*), with the above mentioned (see Chapter 5.5.2) values for their parameters. Other distance models have been also used without obtaining significant results. The last link discovering solution has been coded by using an additional knowledge about all the specific strings coding problems as previously reported.

These tests were repeated on data to which the Quality Improvement process has been applied: tests were carried out on a sample composed of two datasets, chosen among the 27 services datasets provided by the Tuscany region. Datasets involved are the same previously used in other tests, in order to verify, if QI affects rules to be applied for reconciliation through links discovering approach; in fact it is necessary re-determine the optimum combination of values for the thresholds associated with different distances measure.

Method	Precision	Recall	F1
Link discovering – Dice	0,968	0,674	0,794
Link discovering – Jaccard	1,000	0,472	0,642
Link discovering - Knowledge base + Leveisthein	0,925	0,714	0,806
Link discovering + QI – Dice	0,945	0,779	0,854
Link discovering + QI – Jaccard	1,000	0,588	0,740
Link discovering - Knowledge base + QI + Leveisthein	0,892	0,839	0,865

Table 32 - Precision, Recall and F1 comparison with and without QI

Table 32 shows the values obtained in the new test just described, with the data submitted to Quality Improvement, compared to the previous tests without QI (first three rows of the table) and with respective distance measurement of Dice, Jaccard and Levensthein.
These values shown that, for each distance measure applied to the dataset samples, thanks to QI, an improvement in Recall and F1 is obtained, which means that a higher number of reconcilable services are reconciled, compared to a situation which uses a dataset that has not been pre-processed by QI phase. It is therefore clear that the Quality Improvement phase, assumes a greater importance because positively affects the semi-automatic reconciliation performed thanks to the used link discovering techniques.

Looking closely at the results obtained in these tests, it is possible to state that an automated approach to reconciliation, done after a thorough analysis to see which is the best distance function and the respective applicable threshold values, can replace the SPARQL method, which instead requires a greater knowledge of the input data, in order to determine the ad-hoc queries that can be process to obtain a reconciliation triple.

6.5 Triples loading assessment

As we have seen in Section 5.6, a way to check if the process consisting of Phase I, II and III, i.e. the process from data ingestion to triples mapping, is successful, concerns to verify if all triples provided to OWLIM-SE, are properly loaded on the triplestore. This type of verification, also allows to identify use cases not considered in the design phase of the individual process steps.

In Table 33, it is possible to observe an extract of the *processManager* table following the execution of the java validation and verification tool, also designed to perform the triples counting in each file resulting from the mapping phase (Phase III). The analysis showed that a percentage of 99,99% of triples are correctly loading.

Process	Triples	TriplesValidation	Difference
Accessi_sportello_suolo_pubblico_e_taxi_csv	246	246	0
Arrivi_turistici_csv	112	112	0
Arte_e_cultura_csv	44.785	44.785	0
Ataf_csv	275	275	0
Banche_csv	22.806	22.806	0
Corrieri_espresso_csv	676	676	0
Corsi_di_lingue_e_Scuole_di_formazione_csv	909	909	0
Delibere_csv	1.026	1.026	0
Emergenze_csv	9.416	9.416	0
Enogastronomia_csv	83.613	83.613	0
Farmacie_csv	27.679	27.679	0
Grafo_stradale_Arezzo	4.911.197	4.911.197	0
Grafo_stradale_Firenze	10.919.228	10.919.228	0

Grafo_stradale_Grosseto	5.871.753	5.871.753	0
Grafo_stradale_Livorno	7.273.891	7.273.891	0
Grafo_stradale_Lucca	9.786.001	9.786.001	0
Grafo_stradale_Massa_e_Carrara	7.175.667	7.175.667	0
Grafo_stradale_Pisa	11.484.307	11.484.307	0
Grafo_stradale_Pistoia	10.648.352	10.648.352	0
Grafo_stradale_Prato	10.784.844	10.784.844	0
Grafo_stradale_Siena	14.363.127	14.363.127	0
Imprese_del_commercio_csv	4.181	4.181	0
Infrastrutture_aeree_csv	403	403	0
Musei_csv	8.930	8.930	0
Salute_e_sanita_csv	14.860	14.860	0
Scuola_dell_infanzia_csv	6.952	6.952	0
Scuola_elementare_csv	4.332	4.332	0
Scuola_media_csv	1.506	1.506	0
Scuola_superiore_csv	2.374	2.374	0
Servizi_per_il_trasporto_su_strada_csv	2.594	2.594	0
Servizi_vari_csv	4.235	4.235	0
Sinistri_per_via_csv	55.027	55.027	0
Sport_in_Toscana_csv	16.420	16.420	0
Strutture_ricettive_con_georeferenziazione_csv	31.288	31.288	0
Strutture_ricettive2_csv	238.913	238.919	-6
Tempo_libero_csv	7.380	7.380	0
Uffici_vari_csv	6.086	6.086	0
Universita_e_conservatori_csv	559	559	0
Veicoli_circolanti_csv	280	280	0
Vetrina_Toscana_ristoranti_e_botteghe_csv	33.878	33.878	0
Visite_guidate_csv	1.614	1.614	0
Welfare_csv	7.897	7.897	0
Grafo Ferroviario	138.017	138.017	0

Table 33 - Triples counting (Validation Phase)

In fact, only triples, belonging to the second dataset of accommodation facilities in all of Tuscany (*Strutture_ricettive2_csv*), was not loaded. Following a manual analysis of all triples, has been possible to identify the reason for failed loading, due in all six cases, to triple created by Karma with *undefined* value.
Below are the 6 identified triples, are reported; this fact occurred because Karma does not create triple if the cell corresponding to data inside the database is *NULL*, otherwise, if the cell is only empty, Karma creates a triple using the *undefined* value.

```
<http://www.disit.org/km4city/resource/4e0040ce7e1e45481d03652e19d49771>
<http://www.w3.org/2003/01/geo/wgs84_pos#lat>
"undefined"^^<http://www.w3.org/2001/XMLSchema#float> .
<http://www.disit.org/km4city/resource/4e0040ce7e1e45481d03652e19d49771>
<http://www.w3.org/2003/01/geo/wgs84_pos#long>
"undefined"^^<http://www.w3.org/2001/XMLSchema#float> .
```

```
<http://www.disit.org/km4city/resource/8913e143e1a0f7acfb46937c59148eaf>
<http://www.w3.org/2003/01/geo/wgs84_pos#lat>
"undefined"^^<http://www.w3.org/2001/XMLSchema#float> .
<http://www.disit.org/km4city/resource/8913e143e1a0f7acfb46937c59148eaf>
<http://www.w3.org/2003/01/geo/wgs84_pos#long>
"undefined"^^<http://www.w3.org/2001/XMLSchema#float> .

<http://www.disit.org/km4city/resource/fd6dc8e2ff1af3f7279ea7617f2bd38d>
<http://www.w3.org/2003/01/geo/wgs84_pos#long>
"undefined"^^<http://www.w3.org/2001/XMLSchema#float> .
<http://www.disit.org/km4city/resource/fd6dc8e2ff1af3f7279ea7617f2bd38d>
<http://www.w3.org/2003/01/geo/wgs84_pos#lat>
"undefined"^^<http://www.w3.org/2001/XMLSchema#float> .
```

Thanks to this discovery has been possible to further improve the ingestion phase, by inserting an additional check on missing values, to ensure that the corresponding value entered on HBase is always *NULL* and not a blank cell.

6.6 Validation results

As seen in Section 5.6, a validation procedure is required to check if data, once inserted inside the triplestore, after having been molded in accordance with the created R2RML models, are interpreted as wanted.
With regard to the second part of the data validation process presented in Section 5.6, in the following table, results achieved after the instances number count of each class, chosen to be validated, are presented, limited to a certain context. As is possible to verify in Table 34, counting of loaded instances corresponds to the number of instances resulting from a triplestore interrogation, for all datasets that do not belong to the Tuscany region Street Graph.

id	File	tipo1	count1	tipo1count
1	Luoghi_freschi_a_Firenze_kmz.n3	km4c:FreshPlace	25	25
2	Grafo_stradale_Arezzo_cippo.n3	km4c:Milestone	1.248	1.248
3	Grafo_stradale_Arezzo_com.n3	km4c:Municipality	39	39
4	Grafo_stradale_Arezzo_eleroad.n3	km4c:RoadElement	40.643	40.643
5	Grafo_stradale_Arezzo_entry.n3	km4c:Entry	174.597	174.597
6	Grafo_stradale_Arezzo_estesa.n3	km4c:AdministrativeRoad	9.887	9.887
7	Grafo_stradale_Arezzo_giunz.n3	km4c:Node	32.741	32.741
8	Grafo_stradale_Arezzo_manov.n3	km4c:Maneuver	1.445	1.445
9	Grafo_stradale_Arezzo_prov.n3	km4c:Province	1	1
10	Grafo_stradale_Arezzo_reg_acc.n3	km4c:EntryRule	21	21
11	Grafo_stradale_Arezzo_strnum.n3	km4c:StreetNumber	139.898	139.898
12	Grafo_stradale_Arezzo_topon.n3	km4c:Road	10.067	9.956
13	Grafo_stradale_Firenze_cippo.n3	km4c:Milestone	1.287	1.287
14	Grafo_stradale_Firenze_com.n3	km4c:Municipality	44	44
15	Grafo_stradale_Firenze_eleroad.n3	km4c:RoadElement	100.568	101.041
16	Grafo_stradale_Firenze_entry.n3	km4c:Entry	352.827	352.827
17	Grafo_stradale_Firenze_estesa.n3	km4c:AdministrativeRoad	27.584	17.889
18	Grafo_stradale_Firenze_giunz.n3	km4c:Node	81.022	48.759
19	Grafo_stradale_Firenze_manov.n3	km4c:Maneuver	4.611	4.611
20	Grafo_stradale_Firenze_prov.n3	km4c:Province	2	2
21	Grafo_stradale_Firenze_reg_acc.n3	km4c:EntryRule	16.049	17.667

22	Grafo_stradale_Firenze_strnum.n3	km4c:StreetNumber	347.340	347.340
23	Grafo_stradale_Firenze_topon.n3	km4c:Road	27.992	18.889
24	Grafo_stradale_Grosseto_cippo.n3	km4c:Milestone	1.702	1.702
25	Grafo_stradale_Grosseto_com.n3	km4c:Municipality	28	28
26	Grafo_stradale_Grosseto_eleroad.n3	km4c:RoadElement	124.613	125.086
27	Grafo_stradale_Grosseto_entry.n3	km4c:Entry	106.249	106.249
28	Grafo_stradale_Grosseto_estesa.n3	km4c:AdministrativeRoad	36.418	8.834
29	Grafo_stradale_Grosseto_giunz.n3	km4c:Node	100.193	19.171
30	Grafo_stradale_Grosseto_manov.n3	km4c:Maneuver	956	956
31	Grafo_stradale_Grosseto_prov.n3	km4c:Province	3	3
32	Grafo_stradale_Grosseto_reg_acc.n3	km4c:EntryRule	0	0
33	Grafo_stradale_Grosseto_strnum.n3	km4c:StreetNumber	106.351	106.351
34	Grafo_stradale_Grosseto_topon.n3	km4c:Road	37.015	9.217
35	Grafo_stradale_Livorno_cippo.n3	km4c:Milestone	260	260
36	Grafo_stradale_Livorno_com.n3	km4c:Municipality	20	20
37	Grafo_stradale_Livorno_eleroad.n3	km4c:RoadElement	151.406	151.925
38	Grafo_stradale_Livorno_entry.n3	km4c:Entry	141.575	141.575
39	Grafo_stradale_Livorno_estesa.n3	km4c:AdministrativeRoad	43.561	7.167
40	Grafo_stradale_Livorno_giunz.n3	km4c:Node	121.846	21.708
41	Grafo_stradale_Livorno_manov.n3	km4c:Maneuver	2.453	2.453
42	Grafo_stradale_Livorno_prov.n3	km4c:Province	4	4
43	Grafo_stradale_Livorno_reg_acc.n3	km4c:EntryRule	0	0
44	Grafo_stradale_Livorno_strnum.n3	km4c:StreetNumber	134.456	134.456
45	Grafo_stradale_Livorno_topon.n3	km4c:Road	44.182	7.458
46	Grafo_stradale_Lucca_cippo.n3	km4c:Milestone	253	253
47	Grafo_stradale_Lucca_com.n3	km4c:Municipality	35	35
48	Grafo_stradale_Lucca_eleroad.n3	km4c:RoadElement	198.517	199.207
49	Grafo_stradale_Lucca_entry.n3	km4c:Entry	190.461	190.461
50	Grafo_stradale_Lucca_estesa.n3	km4c:AdministrativeRoad	55.572	12.110
51	Grafo_stradale_Lucca_giunz.n3	km4c:Node	160.940	39.261
52	Grafo_stradale_Lucca_manov.n3	km4c:Maneuver	1.628	1.628
53	Grafo_stradale_Lucca_prov.n3	km4c:Province	5	5
54	Grafo_stradale_Lucca_reg_acc.n3	km4c:EntryRule	2	2
55	Grafo_stradale_Lucca_strnum.n3	km4c:StreetNumber	188.890	188.890
56	Grafo_stradale_Lucca_topon.n3	km4c:Road	56.317	12.503
57	Grafo_stradale_Massa_e_Carrara_cippo.n3	km4c:Milestone	589	589
58	Grafo_stradale_Massa_e_Carrara_com.n3	km4c:Municipality	17	17
59	Grafo_stradale_Massa_e_Carrara_eleroad.n3	km4c:RoadElement	216.318	217.150
60	Grafo_stradale_Massa_e_Carrara_entry.n3	km4c:Entry	54.381	54.381
61	Grafo_stradale_Massa_e_Carrara_estesa.n3	km4c:AdministrativeRoad	62.294	6.803
62	Grafo_stradale_Massa_e_Carrara_giunz.n3	km4c:Node	175.932	15.162
63	Grafo_stradale_Massa_e_Carrara_manov.n3	km4c:Maneuver	959	959
64	Grafo_stradale_Massa_e_Carrara_prov.n3	km4c:Province	6	6
65	Grafo_stradale_Massa_e_Carrara_reg_acc.n3	km4c:EntryRule	0	0
66	Grafo_stradale_Massa_e_Carrara_strnum.n3	km4c:StreetNumber	54.713	54.713
67	Grafo_stradale_Massa_e_Carrara_topon.n3	km4c:Road	63.068	6.890
68	Grafo_stradale_Pisa_cippo.n3	km4c:Milestone	874	874
69	Grafo_stradale_Pisa_com.n3	km4c:Municipality	39	39
70	Grafo_stradale_Pisa_eleroad.n3	km4c:RoadElement	262.845	264.694
71	Grafo_stradale_Pisa_entry.n3	km4c:Entry	184.428	184.428
72	Grafo_stradale_Pisa_estesa.n3	km4c:AdministrativeRoad	77.666	15.873
73	Grafo_stradale_Pisa_giunz.n3	km4c:Node	213.590	38.821
74	Grafo_stradale_Pisa_manov.n3	km4c:Maneuver	1.734	1.734
75	Grafo_stradale_Pisa_prov.n3	km4c:Province	7	7
76	Grafo_stradale_Pisa_reg_acc.n3	km4c:EntryRule	15	17
77	Grafo_stradale_Pisa_strnum.n3	km4c:StreetNumber	184.319	184.319
78	Grafo_stradale_Pisa_topon.n3	km4c:Road	78.563	16.378
79	Grafo_stradale_Pistoia_cippo.n3	km4c:Milestone	216	216
80	Grafo_stradale_Pistoia_com.n3	km4c:Municipality	22	22
81	Grafo_stradale_Pistoia_eleroad.n3	km4c:RoadElement	291.386	293.828

82	Grafo_stradale_Pistoia_entry.n3	km4c:Entry	121.473	121.473
83	Grafo_stradale_Pistoia_estesa.n3	km4c:AdministrativeRoad	83.520	6.096
84	Grafo_stradale_Pistoia_giunz.n3	km4c:Node	236.929	23.969
85	Grafo_stradale_Pistoia_manov.n3	km4c:Maneuver	936	936
86	Grafo_stradale_Pistoia_prov.n3	km4c:Province	8	8
87	Grafo_stradale_Pistoia_reg_acc.n3	km4c:EntryRule	0	0
88	Grafo_stradale_Pistoia_strnum.n3	km4c:StreetNumber	120.200	120.200
89	Grafo_stradale_Pistoia_topon.n3	km4c:Road	84.449	6.377
90	Grafo_stradale_Prato_cippo.n3	km4c:Milestone	46	46
91	Grafo_stradale_Prato_com.n3	km4c:Municipality	7	7
92	Grafo_stradale_Prato_eleroad.n3	km4c:RoadElement	304.689	307.815
93	Grafo_stradale_Prato_entry.n3	km4c:Entry	113.483	113.483
94	Grafo_stradale_Prato_estesa.n3	km4c:AdministrativeRoad	87.583	4.387
95	Grafo_stradale_Prato_giunz.n3	km4c:Node	247.391	11.193
96	Grafo_stradale_Prato_manov.n3	km4c:Maneuver	793	793
97	Grafo_stradale_Prato_prov.n3	km4c:Province	9	9
98	Grafo_stradale_Prato_reg_acc.n3	km4c:EntryRule	26	26
99	Grafo_stradale_Prato_strnum.n3	km4c:StreetNumber	113.728	113.728
100	Grafo_stradale_Prato_topon.n3	km4c:Road	88.443	4.466
101	Grafo_stradale_Siena_cippo.n3	km4c:Milestone	1.398	1.398
102	Grafo_stradale_Siena_com.n3	km4c:Municipality	36	36
103	Grafo_stradale_Siena_eleroad.n3	km4c:RoadElement	389.711	393.887
104	Grafo_stradale_Siena_entry.n3	km4c:Entry	125.611	125.611
105	Grafo_stradale_Siena_estesa.n3	km4c:AdministrativeRoad	132.979	46.019
106	Grafo_stradale_Siena_giunz.n3	km4c:Node	318.160	71.988
107	Grafo_stradale_Siena_manov.n3	km4c:Maneuver	1.659	1.659
108	Grafo_stradale_Siena_prov.n3	km4c:Province	10	10
109	Grafo_stradale_Siena_reg_acc.n3	km4c:EntryRule	21	21
110	Grafo_stradale_Siena_strnum.n3	km4c:StreetNumber	118.312	118.312
111	Grafo_stradale_Siena_topon.n3	km4c:Road	132.921	46.833
112	Arte_e_cultura_csv.n3	km4c:Service	3.212	3.212
113	Banche_csv.n3	km4c:Service	1.768	1.768
114	Corrieri_espresso_csv.n3	km4c:Service	51	51
115	Corsi_di_lingue_e_Scuole_di_formazione_csv.n3	km4c:Service	70	70
116	Emergenze_csv.n3	km4c:Service	688	688
117	Enogastronomia_csv.n3	km4c:Service	5.980	5.980
118	Farmacie_csv.n3	km4c:Service	2.131	2.131
119	Imprese_del_commercio_csv.n3	km4c:Service	323	323
120	Infrastrutture_aeree_csv.n3	km4c:Service	29	29
121	Musei_csv.n3	km4c:Service	715	715
122	Salute_e_sanita_csv.n3	km4c:Service	1.127	1.127
123	Scuola_dell_infanzia_csv.n3	km4c:Service	539	539
124	Scuola_elementare_csv.n3	km4c:Service	335	335
125	Scuola_media_csv.n3	km4c:Service	116	116
126	Scuola_superiore_csv.n3	km4c:Service	183	183
127	Servizi_per_il_trasporto_su_strada_csv.n3	km4c:Service	196	196
128	Servizi_vari_csv.n3	km4c:Service	306	306
129	Sport_in_Toscana_csv.n3	km4c:Service	1.184	1.184
130	Strutture_ricettive2_csv.n3	km4c:Service	15.143	15.143
131	Strutture_ricettive_con_georeferenziazione_csv.n3	km4c:Service	2.016	2.016
132	Tempo_libero_csv.n3	km4c:Service	564	564
133	Uffici_vari_csv.n3	km4c:Service	449	449
134	Universita_e_conservatori_csv.n3	km4c:Service	43	43
135	Vetrina_Toscana_ristoranti_e_botteghe_csv.n3	km4c:Service	2.260	2.260
136	Visite_guidate_csv.n3	km4c:Service	114	114
137	Welfare_csv.n3	km4c:Service	593	593
138	Accessi_sportello_suolo_pubblico_e_taxi_csv_stat.n3	km4c:StatisticalData	36	36
140	Arrivi_turistici_csv_stat.n3	km4c:StatisticalData	16	16
141	Ataf_csv_stat.n3	km4c:StatisticalData	40	40
142	Delibere_csv_delibere.n3	km4c:Resolution	7.901	7.901

143	Sinistri_per_via_csv_sinistri.n3	km4c:StatisticalData	7.861	7.861
144	Veicoli_circolanti_csv_stat.n3	km4c:StatisticalData	40	40

Table 34 - Counting of loaded instances (Validation Phase)

Concerning to the Street Graph, it is instead necessary to distinguish three cases:

- The counting of loaded instances corresponds to the number of instances resulting from a triplestore interrogation;
- The counting of loaded instances is lower than the number of instances resulting from a triplestore interrogation;
- The counting of loaded instances is greater than to the number of instances resulting from a triplestore interrogation.

An in-depth data analysis was then carried out, in order to understand the reasons for the differences found in the two counts.
As is possible to see from Table 34, almost all datasets related to *Node*, *Road* and *AdministrativeRoad*, have a higher count within the triplestore; to explain why this happens, an example will be used. Consider the Province of Florence and the ranks of its Node. Within the *Node* file there are 48.759 different elements belonging to the Province of Florence, which are correctly loaded into the repository, with context <*http://www.disit.org/km4city/GrafoStradale/Grafo_Stradale_Firenze*>. It is easy to understand that there are *RoadElement* shared by more than one province, i.e. they have a starting node and ending node belonging to two different provinces. Then, within the file relating to *RoadElement*, there are triples relative to nodes of other provinces, to which, however, being present in a file related to the province of Florence, OWLIM also associate the context of the Province of Florence. So, counting *Node* instances in triplestore related context <*http://www.disit.org/km4city/GrafoStradale/Grafo_Stradale_Firenze*>, in reality, a greater number of instances will be counted, compared to those belonging to the province of Florence.

As regards the case of a lower count inside the triplestore, it is possible to observe from Table 34 that this case is limited to *RoadElement* and *EntryRule* instances (this second option affects only the province of Pisa). Thanks to a further data analysis, it was possible to verify that *RoadElement* information are slightly lower number within the repository, compared to the loaded number due to an error present in triples. In fact, the missing *RoadElement* are inferred as instances of the EntryRule class of the KM4City Ontology, due to an incorrect R2RML model that generate an ObjectProperty triple with swapped domain and range. Checking on missing *EntryRule*, instead, has identified a problem in data source, which was promptly communicated to the data supplier.

However, thanks to the validation phase has been possible to identify and correct the problem and regenerate a new triplestore devoid of these detected inconsistencies.

6.7 Interconnection evaluation

In Section 5.6 has been also taken into analysis, the verification of interconnections that could be created between data, belonging to different datasets. In fact, to check whether the data loaded into triplestore behave as expected, that is, if the applied modeling allows to interconnect the data as initially planned, some ad-hoc SPARQL queries were performed.
To better clarify the used methodology, some interest areas in the ontology were chosen as sample, for design the appropriate queries to be executed, in order to verify which interconnections has been correctly created.
In Table 35, the results obtained from the following query, are shown:

```
SELECT distinct ?ser ?serAddress ?elat ?elong ?sType ?sName ?email
?note
WHERE {
      ?ser <http://schema.org/name> ?name .
      ?ser <http://schema.org/streetAddress>  ?serAddress .
      ?ser km4c:has Access ?entry .
      ?entry geo:lat ?elat .
      ?entry geo:long ?elong .
      ?entry omgeo:nearby (43.7754868 11.2480146 “0.3mm”) .
      OPTIONAL {?ser skos:note ?note} .
      OPTIONAL {?ser <http://schema.org/email> ?email} .
} LIMIT 200
```

Ser	**SerAddress**	**Elat**	**Elong**	**SName**	**Email**	**Note**
<http://www.disit.org/km4city/resource/0f9f2f012684bad736e5a84676843753>	"VIA PANZANI "^^xsd:string	"43.773981688334459"^^xsd:float	"11.251067621724776"^^xsd:float	"Giglio Rosso"^^xsd:string	"info@ristorantegigliorosso.com"^^xsd:string	"Phone additional: 055211795"^^xsd:string
<http://www.disit.org/km4city/resource/4d2f8a71134d0e16d32d9cc71c3efd7f>	"VIA PANZANI "^^xsd:string	"43.773981688334459"^^xsd:float	"11.251067621724776"^^xsd:float	"Rostorante Giglio Rosso"^^xsd:string	"info@ristorantegigliorosso.com"^^xsd:string	"Phone additional: 055211795"^^xsd:string
<http://www.disit.org/km4city/resource/0d993404cb2427c43230bdd27166c293>	"VIA FIUME"^^xsd:string	"43.77709708015724"^^xsd:float	"11.250142322350097"^^xsd:float	"ALBANI"^^xsd:string	"info.flo@albanihotels.com"^^xsd:string	
<http://www.disit.org/km4city/resource/25d7445b5daf76dafd7697b0e826fd99>	"VIA FIUME"^^xsd:string	"43.776905611527056"^^xsd:float	"11.250264056781193"^^xsd:float	"JOLY"^^xsd:string	"info@hoteljoly.it"^^xsd:string	
<http://www.disit.org/km4city/resource/3a54ff1082a4e4c17f699873c67e652f>	"VIA FIUME"^^xsd:string	"43.776905611527056"^^xsd:float	"11.250264056781193"^^xsd:float	"LOMBARDI"^^xsd:string	"hotel.lombardi@dada.it"^^xsd:string	
<http://www.disit.org/km4city/resource/fb98cd2d2dca58ac61db740707194d8e>	"VIA NAZIONALE"^^xsd:string	"43.776427111549665"^^xsd:float	"11.250788174287324"^^xsd:float	"VANESSA"^^xsd:string	"weidan.zhu@yahoo.it"^^xsd:string	"general information: (FIRENZE)"^^xsd:string

<http://www.disit.org/km4city/resource/1f12ac7183f600afe9dfb8716ca6c244>	"VIA FIUME"^^xsd:string	"43.77763962999515"^^xsd:float	"11.249794558517685"^^xsd:float	"CELLINI"^^xsd:string		
<http://www.disit.org/km4city/resource/2286d211298721fc25969f9aeea58355>	"VIA FIUME"^^xsd:string	"43.77763962999515"^^xsd:float	"11.249794558517685"^^xsd:float	"DESIREE"^^xsd:string	"info@desireehotel.com"^^xsd:string	
<http://www.disit.org/km4city/resource/4bdb2a00e2eaa54c3224ece0f14a81e5>	"VIA FIUME"^^xsd:string	"43.77763962999515"^^xsd:float	"11.249794558517685"^^xsd:float	"SERENA"^^xsd:string	"info@albergoserena.it"^^xsd:string	
<http://www.disit.org/km4city/resource/5492c0ab37dec9cab90bc73896a6b5e3>	"VIA FIUME"^^xsd:string	"43.77763962999515"^^xsd:float	"11.249794558517685"^^xsd:float	"FIORITA"^^xsd:string	"info@hotelfiorita.com"^^xsd:string	"Phone additional: 0552654376"^^xsd:string
<http://www.disit.org/km4city/resource/537e38ed969b4025f7d33f06dfc280d5>	"VIA FIUME"^^xsd:string	"43.777560768127806"^^xsd:float	"11.249684427529154"^^xsd:float	"ERINA"^^xsd:string	"info@hotelerina.it"^^xsd:string	
<http://www.disit.org/km4city/resource/5d4a001e753d2469e255024449173b9f>	"VIA FIUME"^^xsd:string	"43.777560768127806"^^xsd:float	"11.249684427529154"^^xsd:float	"SOGGIORNO_ISABELLA_DE'_MEDICI"^^xsd:string	"info@isabellademedici.com"^^xsd:string	
<http://www.disit.org/km4city/resource/66118578e01e1d61a1d56b2753e6682f>	"VIA FIUME"^^xsd:string	"43.777560768127806"^^xsd:float	"11.249684427529154"^^xsd:float	"DUCA_D'AOSTA"^^xsd:string	"info@hotelducadaosta.eu"^^xsd:string	
<http://www.disit.org/km4city/resource/cb16944181be95d5c8a9848f88adbe5f>	"VIA FIUME"^^xsd:string	"43.777560768127806"^^xsd:float	"11.249684427529154"^^xsd:float	"STELLA_MARY"^^xsd:string	"info@hotelstellamary.it"^^xsd:string	
<http://www.disit.org/km4city/resource/eb438ed6ea3fd1a74168c10b96898a75>	"VIA FIUME"^^xsd:string	"43.777560768127806"^^xsd:float	"11.249684427529154"^^xsd:float	"STEFANIA_INTERNATIONAL"^^xsd:string	"stefaniarooms@gmail.com"^^xsd:string	
<http://www.disit.org/km4city/resource/76a2a0aee0aed4731ee79039bca7be66>	"PIAZZA STAZIONE"^^xsd:string	"43.776904122144011"^^xsd:float	"11.249425264354052"^^xsd:float	"BENETTON"^^xsd:string		
<http://www.disit.org/km4city/resource/ae05adee17e6db7bd5a20a0da399f1c5>	"VIA L. ALAMANNI"^^xsd:string	"43.777098642136679"^^xsd:float	"11.245715701172763"^^xsd:float	"DELLE_NAZIONI"^^xsd:string	"hotel@dellenazioni.it"^^xsd:string	
<http://www.disit.org/km4city/resource/948840eb339c306a37d03c98003419d0>	"VIA FIUME"^^xsd:string	"43.77721418537643"^^xsd:float	"11.249910625277709"^^xsd:float	"LE_CAMERE_DEI_CONTI"^^xsd:string	"info@cameredeiconti.it"^^xsd:string	
<http://www.disit.org/km4city/resource/98af424826a4a3978af464b7119a0426>	"VIA FIUME"^^xsd:string	"43.77721418537643"^^xsd:float	"11.249910625277709"^^xsd:float	"BERKLEYS"^^xsd:string		
<http://www.disit.org/km4city/resource/b20b67c9f11d6d86e740c43bfe9a9626>	"VIA FIUME"^^xsd:string	"43.77721418537643"^^xsd:float	"11.249910625277709"^^xsd:float	"ANGELICA"^^xsd:string	"info@hotelangelicafirenze.com"^^xsd:string	
<http://www.disit.org/km4city/resource/c3fef7434a651c47476b17246051921e>	"VIA FIUME"^^xsd:string	"43.77721418537643"^^xsd:float	"11.249910625277709"^^xsd:float	"BEATRICE"^^xsd:string	"info@hotelbeatrice.it"^^xsd:string	

<http://www.disit.org/km4city/resource/22387a008b66a4f7fa7b9fa5e99e3cbd>	"PIAZZA STAZIONE"^^xsd:string	"43.77549322296133"^^xsd:float	"11.247003758283871"^^xsd:float	"Ristorante Lounge Bar Deanna"^^xsd:string	"operativo@lowcostparking.it"^^xsd:string	"Phone additional: 0552647063"^^xsd:string
<http://www.disit.org/km4city/resource/87621575f80b535591bcd74085297b93>	"VIA NAZIONALE"^^xsd:string	"43.776484340076195"^^xsd:float	"11.251082324180519"^^xsd:float	"SUORE_OBLATE_DELLO_SPIRITO_SANTO"^^xsd:string		
<http://www.disit.org/km4city/resource/fcc15183d8f456aa54a6fc669d296217>	"VIA NAZIONALE"^^xsd:string	"43.776484340076195"^^xsd:float	"11.251082324180519"^^xsd:float	"ISTITUTO_SUORE_OBLATE_SPIRITO_SANTO"^^xsd:string		

Table 35 - Results of a Validation query

This results show that the services reconciliation was successful, and the application of this method on samples taken from ontology areas, mainly affected by the reconciliation process, allows to find a confirmation that the processes involved have been carried out properly.

6.8 Volume measure

In the field of Big Data, the data volume is a key issue to be considered, so in this section, some measurements on triplestore size were performed.
Analyzing the percentage occupied by each macroclass, the results presented in Table 36 can be obtained.

Macro Class	Static Triples	Reconciliation Triples	Real Time Triples Loaded	Total on 1.5 months
Administration	2.431	0	--	2.431
Metadata of DataSets	416	0	--	416
Point of Interest (35.273 POIs in Tuscany)	471.657	34.392	--	506.049
Street-guide (in Tuscany)	68.985.026	0	--	68.985.026
Local Public Transport (<5 lines of FI)	644.405	2.385	135.952 per line per day, to be filtered, read every 30 s, they respond in minutes	(static) 646.790
Sensors (<201 road sensors, 63 scheduled every two hours)	--	4.240	102 per sensor per read, every 2 hours, they are very slow in responding	51.111.078
Parking (<44 parkings, 12 scheduled every 30min)	--	1.240	7.920 per park per day, 3 read per hour, they respond in seconds	
Meto (286	--	--	185 per location per	

municipalities, all scheduled every 6 hours)			update, 1-2 updates per day	
Temporal events, time stamp	--	--	6 for each event	
Total	**70.103.935**	**42.257**		**122.966.893**

Table 36 - Tiple counting for each macroclasses

The triplestore currently contains about 123 million triples, the larger part of which belonging to the Street Guide, which represents the backbone of the entire system.
The monthly growth of the triplestore was also estimate, taking into account the following assumptions:

- Sometimes road sensors are not working at full capacity, for counting monthly triples only 8 sensors were considered, each of which producing an average of 7 measurements each day (at full capacity each sensor can produce more than 200 measurements);
- Weather forecast sometimes have been updated only once a day, so to calculate the monthly triples amount, only 1,5 updating for a day per municipality (285 municipalities in all Tuscany Region), have been taking into account;
- The AVM systems are installed in most of the ATAF bus lines, covering the metropolitan area of Florence; considering that part of the acquired data by AVM systems are Private Data of ATAF, and that sometimes the system responds slowly, the monthly triples calculation has been made only on a single ATAF line, for which the AVM system provides an average of 15 measurements per day;
- Car Park data is the most stable among all real-time data acquired from the system, so, for this reason, 8 car parks operating almost at full capacity have been considered to calculate the monthly triples amount, each of which producing an average of 128 measurements per daily (one every 12 minutes).

The triples amount per month obtained, thanks to this assumptions, corresponds to the values reported in Table 37.

	#items	**#measurements**	**Triples per measurement**	**Daily Triples**	**Montly Triples**
Car Park	8	128	84	86.016	2.616.320
Weather Forecast	285	1,5	217	92.767,5	2821.678,125
AVM system	1	15	44.693,5	670.402,5	20.391.409,38
Road Sensor	6	7	120	5040	153.300
				854.226	25.982.707,5

Table 37 - Monthly growth for Real Time triples

So the triplestore, would have a monthly growth of at least 25 million triples, a very high value. It is therefore evident that Real Time data require a strategy to avoid that the repository may grow too quickly and will arrive in a short time to collapse the system. In fact, a time window of one month was chosen, as a period for which Real Time data are maintained within the front-end triplestore, and up to 2 times a month, the triplestore is re-generated by selecting the real-time data of the last few days, thanks to the appropriate tools presented in Section 5.4, which allows to regenerate triplestore indices. Then, this strategy allows to keep the triplestore size approximately to 96 million triples, and, if necessary, it is always possible to retrieve historical data of each Real Time measurement recorded.

6.9 Time response evaluation

The different components of the system architecture were deployed on 11 virtual machines (VM), each equipped with 12GB RAM and an Intel Xeon X5690@3.47Gz CPU, running Ubuntu 14.04 distribution or WinServer2008.
During the development of the system, particular attention was paid to identifying the right scheduling time of the different Real Time processes; in fact, occasionally some road sensors employ a very long time to provide an answer, and leave process running to wait for data, is not a good solution, because it occupies system resources unnecessarily. It was therefore necessary to insert a time-out period for such more problematic processes, beyond which the problematic process is killed by the process scheduler and rescheduled after a few minutes. Thanks to this technique, a more balanced and reliable system has been obtained, in which a part of the resources is always available (consequently avoiding deadlocks), and in which the following Real Time processes are always scheduled:

- All parking sensors are scheduled every 30 minutes (12 processes);
- Road sensors, currently limited to Florence, Empoli, Arezzo and Piombino, are scheduled every 6 hours (catalogs from 25 to 64, from 13 to 23 and catalogs 3 and 4);
- AVM sensors on all 5 lines currently available, are scheduled at intervals of 10 minutes;
- All 286 weather processes are scheduled with an operation interval of 6 hours.

In order to evaluate the performance of Ingestion and Mapping phases (Phase I and Phase III in Figure 25), in the following table, data relating to execution time of each process, were collected.

Dataset	Ingestion Execution Time	Mapping Execution Time
Previ_Firenze	4	5
Previ_Firenze	6	32

Previ_Buti	7	9
Previ_Buti	2	2
Previ_Bucine	4	4
Previ_Bucine	4	4
AVM_L4	30	4
AVM_L4	25	4
AVM_L4	19	5
AVM_L6	202	22
AVM_L6	256	25
AVM_L6	273	21
Parcheggio161	9	17
Parcheggio161	17	17
Parcheggio162	11	11
Parcheggio162	9	13
Sensori43	1124	3
Sensori43	185	2
Sensori13	25	22
Sensori13	11	9

Table 38 - Ingestion and Mapping execution time (in seconds)

As can be seen from the values in the table above, on average, the processes do not employ more than ten seconds to run, especially those relating to the Mapping phase. However, as regards the Ingestion phase, it is possible to observe how a single sensor, the number 43 for example, during two different time sampled, responded with time very different, which in the first case exceeding 18 minutes.

Finally, always regarding the Indexing phase, the time needed to create a triplestore, containing only all static data, with OWLIM-SE, is about 8 hours. These long time is mainly due to inference that the same OWLIM performs on read data, and to the indices creation. To use a different RDF database management systems, could be a solution that can help to save time during this step; but each different triplestore applies a different type of inference on data, and then a considerable variation on creating time, could mean a minimized amount of data infers, that limit benefits of data interconnection.

Chapter 7

7. Conclusions

The research and development work, that has allowed to build the entire architecture for enrichment, interconnection and exploitation of data, presented in this thesis, has involved more people of the DISIT lab, the Distribute Data Intelligence and Technologies Lab, of the University of Florence. Within the scenario presented, I have personally dealt with the Km4City Ontology study and realization, presented in Chapter 4, supported by the experience of Prof. Nesi and Eng. Bellini. I also personally developed the tool used during the Data Validation phase (Section 5.6), and all R2RML models, used instead in the Mapping phase (Section 5.3). In relation to the Ingestion phase, instead, I have personally dealt with the data ingestion of the *Street Guide and Rail Network* macroclass, and I provided support during SPARQL reconciliation (Section 5.5.1).

This dissertation formalizes an ontology which aims to semantically interconnect information from different sources within a city. The derived ontology has been obtained by means of an incremental process performed analyzing, integrating and validating each added data set. Thus the resulting ontology is a strong generalization of a large set of data modeling problems. Considering the amount of data produced each day within a city, and more specifically in the analyzed use case, we can place the research project in the field of Big Data.
In this scenario, the KM4City Ontology has proven to be a powerful modeling language that has allowed to relate classes, properties and ontological instances, thus creating a repository exploited for the construction of new advanced services, directed to the citizen.
Our model is capable of storing information about the city such as data on the population, accidents, flooding, votes, administrations, location of point of interests (including museums, tourism attractions, restaurants, shops, hotels), ambient data, weather status and forecast, data coming from mobility and transport such as those

created by ITS, for bus management, and solutions for managing parking areas, car flow, accesses on RTZ.

The research project conducted in this context, has also allowed to build a system for the ingestion of public and private data for smart city with related aspects as road graph, services available on the roads, traffic sensors etc. This architecture includes both open data from public administration and private data coming from transport systems integrated mangers, thus addressing and providing real time data of transport system, i.e., the busses, parking, traffic flows, etc. So the realized system allows managing large volumes of data coming from a variety of sources considering both static and dynamic data. This data is then mapped to the km4City Ontology and stored into an RDF-Store where this data are available for applications via SPARQL queries to provide new services to the users. In addition, a thorough verification and validation process performed, allowed us to identify the set of triples to: (i) improve and enrich the model, and (ii) perform the corrections. Thus improving and enabling the deductive capabilities of the final model. Finally, the proposed architecture also provides a visualization and exploration tool to explore the data available in the RDF-Store.

In particular, this thesis address the problem of detecting how the different data can be interconnected to maximize the value, which can then be obtained from their exploitation. A particular effort is dedicated to designing a robust and performing architecture for the automatic ingestion and mapping of data.
In addition, on Section 5.5, the comparison between two possible reconciliation approaches has been addressed in detail: the performed assessment and comparison has produced a clear results demonstrating that the best quality of results are obtained by using the approach based on SPARQL queries plus some manual actions. Also the simple usage of SPARQL queries resulted to be better ranked with respect to the SILK based link discovering. On the other hand, the writing of link discovering algorithms resulted to be much simpler and faster that performing a set of specific SPARQL queries. Moreover, the Quality Improvement process has also allowed to further improve the Silk reconciliation results, thus reducing the difference between the two approaches.

The experience gained from the study of literature and from the trial conducted in the city of Florence, has allowed to assess the actual potential of the ontology for the construction of complex architectures, highlighting their suitability in building systems with a high degree of interoperability, maintainability and evolvability.
The creation of applications that exploit the repository created, has made evident some criticalities. In particular, it highlighted the lack of maturity of the ontological repositories, that today can not compete in terms of stability with the more traditional platforms for relational databases, which are, however, of little use having to work with large volumes of data. It's perfectly reasonable to expect, however, that these problems

will be overcome in the near future by the normal evolution of technology and for the moment can be mitigated through the use of techniques such as the periodic regeneration of the indices proposed in Section 5.4.

Regarding the Completeness, it seems to be well addressed by all the analyzed datasets. Moreover we noticed that there is only one case in which this value is less than 79%. The Accuracy dimension, instead, exceeds 20% only on 2 of the 27 datasets analyzed. The Consistency values, but also other assessments made on data before and after the application of quality improvement phase, show that this process allows to obtain considerable improvements on data, both in terms of quality and quantity of information that is then possible to exploit.
Thanks to the analysis of ingested datasets, that aims to measure their quality, it was possible to provide suggestions to data providers, showing them how to intervene on their Public or Private Data, in order to improve quality; this service can be also provided by the DISIT Lab, thanks to the Quality Improvement processes realized within the architecture (Section 5.2).

Furthermore, the architecture built should enable the use of data by PA and SME, to realize new services to be provided to citizens; it is therefore important to be able to improve the architecture, where it has showed some minor problems. For this purpose, the Ingestion phase will be enriched with new datasets, that make it interesting their interconnection with data already existing inside the triplestore; the Quality improvement phase, instead , will be improved thanks to the errors that has not been able to correct, which represent the first new cases to be treated, in order to obtain a greater number of triples from the ingested data. The Link Discovery Reconciliation method has shown good results, so we will try to automate this stage, as much as possible; with regard to data Validation, this phase has demonstrated its importance as it allowed us to detect errors within the created triplestore, which were easily resolved. On the other hand, however, relating to validation there is still some work to be done, because this phase has not been carried out on all ingested datasets, in addition, according to the findings in Section 6.6, in relation to the counts, a more reliable method must be developed for the desired verification, for example more context for each province could have been included inside triplestore, respectively, related to the each individual classes of the Street Graph.

Concluding this data analysis project, the mission and objectives, as stated in the introduction chapters of this dissertation, are accomplished, and a more complex extension of the conducted research could concentrate on the development of new innovative services, as explained in the following. Furthermore, the methodology devised in this dissertation provides an adequate basis to extend the research in many possible directions regarding the various phases of the proposed architecture.

7.1 Future works

Future works comprehend the prosecution of research on the Km4City Ontology: other OpenData sets will be integrated to the current ontology, which as a result will be further expanded and enhanced to host the new dataset and the new data interconnections, that can be created.
The next step, already in development, will be to identify famous names, points of interest, locality names that can be linked to other data set as DBpedia3 or GeoNames4 according to a Linked Open Data model, thanks to NLP algorithms.

Another interesting topic that shall be addressed soon, concerns the search for a new RDF management system that allows to handle large quantities of triples effectively (Big Data oriented RDF), and to solve problems encountered, in this dissertation, working with OWLIM-SE or at least to make them less influential on the entire realized system. In fact, thanks to the research work done in these PhD, it was possible to verify that the triplestore RDF, do not scale very well if they reach an excessive size.

Furthermore, at the moment, the issue of long data loading time on OWLIM-SE has been bypassed creating a backup triplestore already containing all the static part of triples generated. If necessary, this backup is used as a base for the indies regeneration process, allowing to complete this operation in a very short time.
However, once the problems related to triplestore RDF, will be resolved, additional investments can be done in the direction of designing new innovative services to the citizens of Florence and Tuscany; in fact, the architecture has been designed to allow the exploitation of triplestore by third parties.
For example, a new service created could follow the ServiceMap style, improving its performance and increasing its available functionalities. The new service could be organized in tabs, each of which allows to identify a specific use case addressed: we imagine a tab dedicated to mobility, in which it is possible to calculate the interest route, based on the detected position and the indicated destination; the system will list the identified solutions based on an increasing travel time. Logged in users can also save their preferences in relation to the various use cases implemented by the new system.

Finally, this dissertation is intended to be a stimulus for PA and SME to contribute to the creation of more semantically interoperable data, and to create new innovative services that improve the citizen quality of life.

Appendix A

To complete the project, many software have been used, most of which are Open Source. Each software presented in the following sections, has been used in a specific phase of the work, which will be specified within each description.

A.1 Pentaho Kettle

To processing the input data from the different sources listed in Chapter 3, a software *ETL* (Extract Transform and Load) has been used, because the volume of data to be processed is very high, but the operations to be applied are, for the most part, very common and of acceptable difficulty (string manipulation, joining tables, file splitting, filtering rows, format conversions, etc.). In fact, as the name suggests, an ETL software, mainly extract data from multiple sources, process them through a chain of transformations, and finally perform the data loading within a data structure selected.
After a thorough anlysis carried out on the main ETL software that is CloverETL, Pentaho Kettle Data Integration and Talend Open Studio, the choice fell on Pentaho Kettle.
In detail, some Pentaho Kettle characteristics that have influenced the choice are:

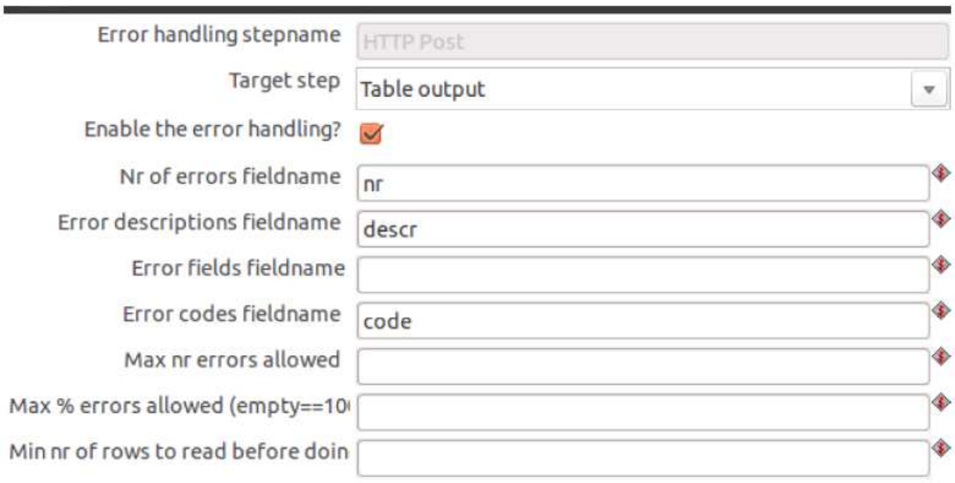
- Its wide spectrum of possible input sources and output formats (CSV, JSON, Shapefile, KML, relational databases, NoSQL Datatbase, etc.)
- Its ability to export the transformations and execute them in batch mode using a scheduler.
- Its aility of manipulating transformations injecting native Java (or Javascript) code directly to them, in order to fully customize the results.
- The high number of users that animate the Kettle's online community and its extensive documentation available on the web.

Kettle has two working modes: via its GUI (the process is called *Spoon*) and through the terminal, via an executable called *Kitchen*. Thanks to the graphical interface it is possible to define the two main Kettle components, that is, *Transformations* and *Jobs*. A transformation is the core element of the software, that allows to define the main processing operations on data stream. A Job is instead a transformations sequence, thanks to which is possible to define in detail, the operations flow to be carried out. Thanks to a *Job*, is possible to define sequence of operations, parallelism between operations, loops, exception handling, etc. while with *Transformation* this can not be done.

Through the graphical interface both *Transformations* and *Jobs* can be performed: Kettle provides also several tools for analysis and logging, to check in detail the implementation of operations and to take action in case of errors. *Transformations* and *Jobs* saved in the specific Kettle format, may be performed later also via command line: it is in fact possible to schedule various processes and to execute them automatically without a user that interacts with the program.

A.1.1 *Error Handling Kettle*

Pentaho Kettle has an interesting integrated mode for error handling, within *Transformations*. In many figures of Chapter 5.1 is indeed possible to see some red dashed lines connecting steps: these links are activated only if the step, from which the red dashed line originates, generates an error. In this case, all steps defined in error handling branch, will be performed.
It is also possible to generate information about the fields, once embarked on the branch of error handling. To activate error handling, simply set as in Figure 113, the corresponding properties.



Error handling stepname	HTTP Post
Target step	Table output
Enable the error handling?	✓
Nr of errors fieldname	nr
Error descriptions fieldname	descr
Error fields fieldname	
Error codes fieldname	code
Max nr errors allowed	
Max % errors allowed (empty==10	
Min nr of rows to read before doin	

Figure 113 - Kettle error handling

All *Transformations* defined for this project, generate the following fields:

- **Number of errors**: that indicates the number of errors generated during the step execution;
- **Error description**: extended description of the error type;
- **Error codes**: error ID code.

In addition, each *Transformation* performs two in the branch of error handling, i.e. *output Table* and *Insert/update*. The first step adds a row to the MySQL table *errors*, which contains an errors log, which is made up with the following columns:

- **Process**: it identifies the process name; its possible values are those contained in the column *process* of the *ProcessManager* table;
- **Section**: it can be *A*, *B* or *C*, depending on which part of the process has generated the error;
- **Time**: field that has a default value, that is the timestamp generated by *MySQL*, which is added when Kettle generates the error line;
- **Code**: it is the error code;
- **Number**: the number of errors occurs;
- **Description**: a brief error description.

Moreover, the *insert/update* step updated to *yes* the error field of the *ProcessManager* table, within the row which has the process identifier equal to *processName* value.
This error handling has proven to be very useful when debugging, because it allows to keep track of errors occur, with detailed information directly provided from Kettle.

A.2 QGIS

Quantum GIS is an Open Source software Geographical Information Systems (GIS) which allows to acquire, manipulate and display information on geo-localized data. This software has a good graphical interface and a very advanced set of executables for its main functionality, recallable via command line. QGIS, within the project, has been used to unify the geographic projection system, on input data coming from different sources. In fact, most of the data belonging to the *Street Guide* is encoded according to the standard *Gauss-Boaga* (code EPSG 3003, also called Monte Mario). To date, the mainly used coordinate system, e.g. from services such as GPS satellite navigation or online maps providers such as OpenStreetMap or Google, is the *WGS84* (code EPSG 4326). In the specific QGIS has been used for the conversion of data from the system to the Gauss-Boaga WGS84, but also to convert the multiple *Shape* files available in KML, due to the fact that the version of *Pentaho Kettle* used, does not accept this data format as input.

A.3 KARMA Data Integration

Karma is a software developed by a team from the University of Southern California, which allows to map in a semi-automatic way, data from files or relational databases, in RDF. It is a recently developed software released under the Apache 2.0 license, and it can be used in two different ways: as a web application run through the platform *Apache Tomcat*, or from the command line to generate RDF triples offline.
The typical Karma workflow, includes to define, through web application, a data mapping on the respective classes and properties belonging to ontologies preloaded. After the mapping, a model must be exported, and its possible formats are the R2RML Model or the Service Model. For the project of this dissertation, the R2RML model has

been used, which is defined by the W3C as "a language for expressing customized mappings from relational databases to RDF datasets". After the R2RML model has been created, it can be used to generate triple in batch mode, starting from a relational database; so the intervention of a user becomes obsolete, and the process can be launched by other software as the just seen Pentaho Kettle.

A.4 OpenRDF Sesame

Sesame is an open source framework for creating and querying RDF repository. It allows to save persistent RDF database (also called Triplestores) locally in memory, on disk, or remotely on a server. Sesame can be used in several ways: through the web interface (implemented as an application running on Tomcat Web Server) through an application, like a console to be launched through the terminal, or through Java API. Sesame in fact makes available to developers the API that can be used by any Java application, which allows to fully customize the use of the framework. In all cases listed above, is possible to access to another very important feature of sesame: quering triplestore through SPARQL language.

A.5 OWLIM

OWLIM is a family of management systems for RDF semantic database; it is typically used as a plugin to add Sesame important characteristics about the knowledge management. Using OWLIM as semantic repository, in fact, new knowledge about the data can be generated through the inference process related to the first-order logic. In addition, OWLIM is developed to achieve maximum scalability even in the presence of significant amounts of data and therefore allows excellent performance in loading and in evaluation of the query.
The OWLIM version used in the project, has been released to DISIT Lab under license, and it presents tools for geo-spatial queries, for creating indexes for full-text research and many other features that increase the usability and speed-up the software.

A.6 Hadoop

Hadoop [Hadoop Apache Project] is a framework that allows managing distributed processing of big data across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each of them offering local computation and storage. The Hadoop library is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures. Hadoop was inspired from Google's Map-Reduce and Google File System, GFS, and in practical it has been realized to be adopted in a wide range of cases. Hadoop is designed to scan large data set to produce results through a distributed and highly scalable batch processing systems. Hadoop is composed of the Hadoop Distribute File System, HDFS, and of the programming paradigm Map-Reduce [Karloff, Suri and Vassilvitskii, 2010]; thus, it is capable to exploit the redundancy built into the environment. The programming model is capable to detect failures and solve them automatically by running specific programs on various servers in the cluster. In fact, redundancy provides fault tolerance and capability to self-healing of the Hadoop Cluster. HDFS allows applications to be run across multiple servers, which have usually a set of inexpensive internal disk drives; the possibility of the usage of common hardware is another advantage of Hadoop. A similar and interesting solution is HadoopDB, proposed by a group of researchers at Yale. HadoopDB was conceived with the idea of creating a hybrid system that combines the main features of two technological solutions: parallel databases in performance and efficiency, and Map-Reduce-based system for scalability, fault tolerance, and flexibility. The basic idea behind HadoopDB is to use Map-Reduce as the communication layer above multiple nodes running single-node DBMS instances. Queries are expressed in SQL and then translated into Map-Reduce. In particular, the solution implemented involves the use of PostgreSQL as database layer, Hadoop as communication layer, and Hive as the translation layer [Abouzeid et al., 2009].

A.7 HBASE

Hbase [Aiyer et al., 2012] is a large-scale distributed database build on top of the HDFS, mentioned above. It is a non-relational database developed by means of an open source project. Many traditional RDBMSs use a single mutating B-tree for each index stored on-disk. On the other hand, Hbase uses a Log Structured Merge Tree approach: first collects all updates into a special data structure on memory, and then, periodically, flush this memory on disk, creating a new index-organized data file, the called also Hfile. These indexes are immutable over time, while the several indices created on the disk are periodically merged. Therefore, by using this approach the writing to the disk are sequentially performed. HBase’s performance is satisfactory in most cases and may be further improved by using Bloom filters [Borthakur et al., 2011]. Both HBase and HDFS systems have been developed by considering elasticity as fundamental principle, and the use of low cost disks has been one of the main goals of HBase. Therefore, to scale the system results is easy and cheap, even if it has to maintain a certain fault tolerance capability in the individual nodes.

A.8 Silk

Silk Workbench is a web application that guides the user through the creating process of a specific interconnection link between two data sources. The workbench consists of:

- Workspace Browser: it provides an interface through which different projects can be load. Each project can have multiple data sources and multiple connection tasks.

- Linkage Rule Editor: it is a graphical editor that allows the user to easily create rules and link specifications. It provides several functions for comparison, thresholding, and text transformations. The rule is visually expressed through a tree view and it can be change through drag-and-drop.
- Evaluation: it allows the user to specify connection. Links appear at the time of their generation and a score of similarity is assigned to each one. Connections for which correctness has not specified, can be confirmed or rejected by the user. In addition, the interface provides detailed results on the scores composition given.

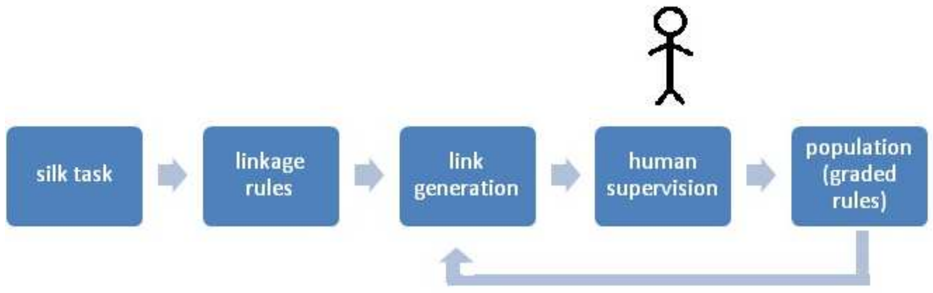


Figure 114 - Execution flow of Silk Workbench

Silk offers three different applications which address different use cases:

- *Silk single machine*: it is used to generate RDF links on a single machine. The data sets to be interconnected, can reside on the same machine or on a remote machine, which can be accessed thanks to the *SPARQL* query language.
- *Silk MapReduce* is used to generate RDF links between datasets using a cluster of multiple machines. Silk *MapReduce* allows processing large datasets by distributing the generation of links to multiple machines.
- *Silk Server* can be used as a component of identity resolution in applications that consume Linked Data from the Web.

To interact with the Silk core, a declarative language *LSL* is provided, for specifying RDF data sources, linking rules and conditions to be met for entity interconnection. The objective of this pre-processing tool is to produce a structured representation of the data to be processed. The resulting output is an RDF dump file containing the structured values extracted. By using this XML-based language, a user can specify which extraction methods will be used.
The following four sections must be defined within each Silk files:

- **Path**: every path begins with a variable that can be followed by multiple elements. If a path cannot be resolved because of a missing property, or a too restrictive filter, it returns an empty result set. To cross the graph and reach the various properties, the following operator types can be used:

  - Forward Operator "/": *<path_segment>/<property>* this operator moves forward from an entity to an object property;
  - Backward Operator "\": <path_segment>\<property> this operator moves backwards from one object property to its subject;
  - Filter operator "[]": <path_segment>[<property><compare_operator><value> <path_segment>] [@lang<comp operator><value>], this operator reduces the resources based on filter matching. Examples of the compare operators are =, <, <=,>, >=, =!.
- **Transformation**: each dataset using different formats, so Silk, to normalize and standardize the data, provides operators that transform the values of a properties set, according to a transformation function. Some of these functions include the removal of blank spaces, special characters removal, lower case, upper case, concatenation, tokenization, brackets removal.
- **Compare**: the comparison is made thanks to comparison operators, i.e. functions that assess the similarity between two results on the basis of a specific distance measure, a threshold and a weight. The weight is used to aggregate functions as a weighted average of the combination; for example, if a rule is composed of two comparisons, it is possible to assign different weights to each comparison, to give greater importance to one of each.The threshold is the maximum distance value between two comparisons; it corresponds to a value between 0 and 1. The distance measure is the metric used to compare two results; it returns 0 in case of perfect match and a higher value if a match is not perfect. Silk provides two types of measures:
  - Based on Character
  - Based on Token

  Measures based on character, compare two strings, character by character. They are generally used to detect typos. The distance measures available are the *Levenshtein*, *Jaro*, *JaroWinkler*, *equality*, *inequality*. Measures token-based, instead, work of errors at word level, for example, strings with different words order, that is "Jhon Doe" or "Doe, John". In this case the distance measures available are *Jaccard*, *Dice* and *SoftJaccard.*
- **Aggregation**: the aggregation operators have the task to combine scores resulting from multiple comparisons in a single score, according to a specific aggregation function. They can be also nested to create a nonlinear hierarchy. Among the aggregation operators there are:
  - Average: it calculates the weighted average of the comparison functions;
  - Maximum: it calculates the highest confidence score in the group;
  - Minimum: it calculates the smallest value;
  - Euclidean Distance;
  - Geometric Distance;

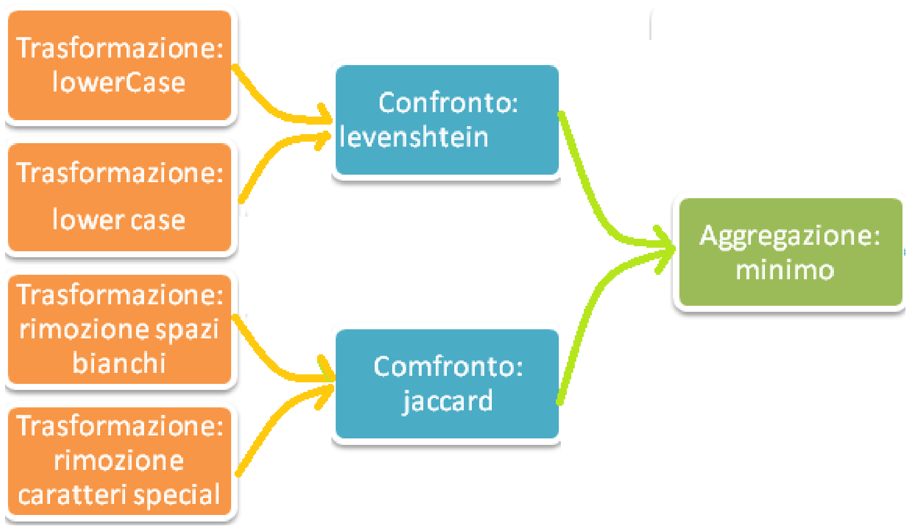

Trasformazione: lowerCase
Trasformazione: lower case
Trasformazione: rimozione spazi bianchi
Trasformazione: rimozione caratteri special
Confronto: levenshtein
Comfronto: jaccard
Aggregazione: minimo

Figure 115 - Connection rule example

Bibliography

[Abouzeid et al., 2009] Abouzeid A.; Bajda-Pawlikowski C.; Abadi D.; Silberschatz A.; Rasin A., "*HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads*", Proceedings of the VLDB Endowment, Pages 922-933, Volume 2, Number 1, August 2009.

[Aiyer et al., 2012] Aiyer A.; Bautin M.; Jerry Chen G., Damania P.; Khemani P.; Muthukkaruppan K.; Ranganathan K.; Spiegelberg N.; Tang L.; Vaidya M., "*Storage Infrastructure Behind Facebook Messages Using HBase at Scale*", Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Pages 4-13, Volume 35, Number 2, June 2012.

[Antoniu et al., 2010] Antoniu G.; Bougè L.; Thirion B.; Poline JB., "*AzureBrain: Large-scale Joint Genetic and Neuroimaging Data Analysis on Azure Clouds* ", 30 September 2010.

[BABELNET] http://babelnet.org/.

[Beckett, 2002] Beckett, D. *The design and implementation of the Redland RDF application framework*. Computer Networks, 39(5), 577-588. 2002.

[Bellandi et al., 2012] Bellandi A., Bellini P., Cappuccio A., Nesi P., Pantaleo G., Rauch N., "ASSISTED KNOWLEDGE BASE GENERATION, MANAGEMENT AND COMPETENCE RETRIEVAL", International Journal of Software Engineering and Knowledge Engineering, World Scientific Publishing Company press, vol.32, n.8, pp.1007-1038, Dec. 2012, DOI: 10.1142/S021819401240013X

[Bellini et al., 2014] Bellini, P., Benigni, M., Billero, R., Nesi, P., & Rauch, N. (2014). Km4City ontology building vs data harvesting and cleaning for smart-city services.Journal of Visual Languages & Computing.

[Bellini et al., 2014B] P Bellini, M Benigni, R Billero, P Nesi, N Rauch, "Ontology Bulding vs Data Harvesting and Cleaning for Smart-city Services", DMS 2014 Conference.

[Bellini et al., 2013A] Bellini P., Di Claudio M., Nesi P., Rauch N., "Tassonomy and Review of Big Data Solutions Navigation", Big Data Computing, Edited by Rajendra Akerkar Chapman and Hall/CRC 2013, Pages 57–101, Print ISBN: 978-1-4665-7837-1

[Bellini et al., 2013B] Bellini, P., Nesi, P., Bruno I. & Paolucci M. Institutional Services and Tools for Content,Metadata and IPR Management, IJSEKE, International Journal of Software Engineering and Knowledge Engineering, 2013.

[Bellini, Cenni and Nesi, 2012] Bellini P., Cenni D.; Nesi P., *"On the Effectiveness and Optimization of Information Retrieval for Cross Media Content"*, Proceeding of the KDIR 2012 is part of IC3K 2012, International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, Barcelona, Spain, 4-7 October 2012.

[Bellini, Nesi, 2013] Bellini, E., & Nesi, P. *Metadata Quality assessment tool for Open Access Cultural Heritage institutional repositories.* In Information Technologies for Performing Arts, Media Access, and Entertainment (pp. 90-103). Springer Berlin Heidelberg. 2013.

[Belllini, Nesi, 2014] Pierfrancesco Bellini and Paolo Nesi, Performing Arts LOD of ECLAP Content Service, LOD2014, Workshop Linked Open Data: where are we?, organized by WRC italy and CNR, Rome, 2014.

[Bellini, Nesi, Rauch, 2014] P Bellini, P Nesi, N Rauch, "Knowledge Base Contruction Process for Smart-city Services", ICECCS 2014. 19th International Conference on Engineering of Complex Computer Systems, 4-7 August,Tianjin, China.

[Bellini, Nesi, Rauch, 2014B] Pierfrancesco Bellini, Paolo Nesi, Nadia Rauch, Ontology Construction and Knowledge Base Feeding and Cleaning for Smart-city Services, a shorter version has been presented at the W3C workshop, titled Smart City data via LOD/LOG Service, LOD2014, Workshop Linked Open Data: where are we?, organized by WRC italy and CNR, Rome, 2014, http://www.disit.org/5606, http://www.disit.org/6036

[Bellini, Nesi, Serena, 2014] Bellini, P., Nesi, P., & Serena, M. (2014). MyStoryPlayer: experiencing multiple audiovisual content for education and training. Multimedia Tools and Applications, 1-41.

[Bellini, Nesi, Venturi, 2014]Bellini, P., Nesi, P., & Venturi, A. (2014). Linked Open Graph: browsing multiple SPARQL entry points to build your own LOD views. Journal of Visual Languages & Computing.

[Berners-Lee, 2006] T. Berners-Lee, "*Linked Data*", http://www.w3.org/DesignIssues/LinkedData.html, 2006.

[Bishop et al., 2011]Barry Bishop, Atanas Kiryakov, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, Ruslan Velkov, "*OWLIM: A family of scalable semantic repositories*", Semantic Web Journal, Volume 2, Number 1 / 2011.

[Bizer et al., 2009] C. Bizer, T. Heath and T. Berners-Lee Linked Data - the story so far. Int. Journal on Semantic Web and Information Systems, 5, (3), 1-22, 2009.

[Borgo, 2004] Borgo, S. "*Classifying (medical) ontologies*." Laboratory for Applied Ontology (LOA), Institute for Cognitive Sciences and Technology (ISTC-CNR), 2004.

[Borthakur et al., 2011] Borthakur D.; Muthukkaruppan K.; Ranganathan K.; Rash S.; SenSarma J.; Spielberg N.; Molkov D.; Schmidt R.; Gray J.; Kuang H.; Menon A.; Aiyer A., "*Apache Hadoop Goes Realtime at Facebook*", Proceedings of the 2011 International Conference on Management of Data, Athens, Greece, June 2011

[Bose et al., 2001] Bose I.; Mahapatra R.K., *"Business Data Mining - a Machine Learning Prespective"*, Information & Management, Pages 211-225, Volume 39, Number 3, December 2001.

[Broekstra et al., 2002] Broekstra, J., Kampman, A., & Van Harmelen, F. *Sesame: A generic architecture for storing and querying rdf and rdf schema.* In The Semantic Web—ISWC 2002 (pp. 54-68). Springer Berlin Heidelberg. 2002.

[Bruce and Hillmann, 2004] Bruce, T.R., Hillmann, D. *Metadata in Practice, chap. The continuum of metadata quality: defining, expressing, exploiting,* pp. 238–256. ALA Editions, Chicago, IL (2004)

[Bryant et al., 2010] Bryant R.E.; Carbonell J.G.; Mitchell T., *"From Data to Knowledge to Action: Enabling Advanced Intelligence and Decision-Making for America's Security"*, Computing Community Consortium, Version 6, July 2010.

[Calvanese et al., 1998]Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D. and Rosati, R. Description Logic Framework for Information Integration. In Proceedings of the 6th International Conference on the Principles of Knowledge Representation and Reasoning (KR-98). Italy. 1998.

[CAP] CAP - https://www.oasis-open.org/committees/download.php/6334/oasis-200402-cap-core-1.0.pdf.

[Caragliu et al, 2009] Caragliu, A; Del Bo, C. & Nijkamp, P. *"Smart cities in Europe"*. Serie Research Memoranda 0048 (VU University Amsterdam, Faculty of Economics, Business Administration and Econometrics), 2009.

[CC0] Creative Commons 0: http://creativecommons.org/publicdomain/zero/1.0/.

[CCA] Creative Commons Attribution: http://creativecommons.org/licenses/by/2.0/.

[CCSA] Creative Commons by Share Alike: http://creativecommons.org/licenses/by-sa/4.0/.

[Cimiano et al., 2013] Lopez, V., Unger, C., Cimiano, P., & Motta, E. *Evaluating question answering over linked data.* Web Semantics: Science, Services and Agents on the World Wide Web, 21, 3-13, 2013.

[Cranefield, 2001] Cranefield, S. *UML and the Semantic Web*, 2001.

[D'Antonio and Tanskanen, 2014] S. d'Antonio and E. Tanskanen. "*Le app e le tecnologie che rendono le città più inclusive e sostenibili: Italia e Finlandia a confronto*" - http://goo.gl/HB3jx7, 2014.

[DataHub] State of the LOD Cloud 2014: http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/.

[DateTime] W3C - Date and Time Formats - http://www.w3.org/TR/NOTE-datetime.

[DateX] DATEX Background http://www.datex2.eu/content/datex-background.

[De Witt et al., 2012] DeWitt S.; Sinclair R.; Sansum A.; Wilson M., *"Managing Large Data Volumes from Scientific Facilities"*, ERCIM News, Page 15, Number 89, April 2012.

[Dean & Ghemawat, 2008] Dean, J., & Ghemawat, S. *MapReduce: simplified data processing on large clusters.* Communications of the ACM, 51(1), 107-113, 2008.

[descLogic] Baader, F. (Ed.). *The description logic handbook: theory, implementation, and applications*. Cambridge university press, 2003.

[DICCOF] DICCOF - http://www.disit.org/5531.

[Domingos, 2005] Domingos P.; *"Mining Social Networks for Viral Marketing"*, IEEE Intelligent Systems, Pages 80-82, Volume 20, Number 1, 2005.

[Domingos, 2005] Domingos P.; *"Mining Social Networks for Viral Marketing"*, IEEE Intelligent Systems, Pages 80-82, Volume 20, Number 1, 2005.

[Drummond, 1995] Drummond, W. J. *Address matching: GIS technology for mapping human activity patterns*. Journal of the American Planning Association, 61(2), 240-251. 1995.

[Eaton et al., 2012] Eaton C.; Deroos D.; Deutsch T.; Lapis G., *"Understanding Big Data: Analytics for enterprise class Hadoop and streaming Data"*, Mcgraw-HillPubl.Comp., ISBN: 978-0071790536, March 2012.

[ECLAP] ECLAP: http://www.eclap.eu.

[Europeana] Europeana Portal - http://www.europeana.eu/portal/.

[EUROVOC] http://eurovoc.europa.eu/.

[Ferrucci et al., 2010] Ferrucci D., Brown E., Chu-Carroll J., Fan J., Gondek D., Kalyanpur A. A., Lally A., Murdock J. W., Nyberg E., Prager J., Schlaefer N., and Welty C. *Building Watson: An overview of the DeepQA project.* AI Magazine, 31(3), 2010.

[Figueireido, Rodrigues and Vale, 2005] Figueireido V.; Rodrigues F.; Vale Z., *"An Electric Energy Consumer Characterization Framework Based on Data Mining Techniques"*, IEEE Transactions on Power Systems, Pages 596-602, Volume 20, Number 2, May 2005.

[Gangemi et al., 2006] Gangemi, A., Catenacci, C., Ciaramita, M., & Lehmann, J. *Modelling ontology evaluation and validation* (pp. 140-154). Springer Berlin Heidelberg. (2006).

[Gangemi, 2005] A. Gangemi, *Ontology design patterns for semantic web content*, in: Y. Gil, E. Motta, R. Benjamins, M. Musen (Eds.), 4th International Semantic Web Conference, ISWC 2005, in: Lecture Notes in Computer Science, vol. 3729, Springer, 2005, pp. 262–276.

[GDF] Graphic Data Files - http://www.iso.org/iso/catalogue_detail.htm?csnumber=54610.

[GoodRelation] Good Relation Ontology http://www.heppnetz.de/projects/goodrelations/.

[Gruber, 1993] T. R. Gruber, *A Translation Approach to Portable Ontology Specification, Knowledge Acquisition*, Volume 5 Issue 2, p.199-220, 1993.

[Gupta et al., 2012] S.Gupta, P.Szekely, C.Knoblock, A.Goel, M.Taheriyan, M.Muslea, "*Karma: A System for Mapping Structured Sources into the Semantic Web*", 9th Extended Semantic Web Conference (ESWC2012).

[Hadoop Apache Project] Hadoop Apache Project - http://hadoop.apache.org/.

[Hanna, 2004] Hanna M., *"Data Mining in the e-learning domain"*, Campus-Wide Information Systems, Pages 29-34, Volume 21, Number 1, 2004.

[HSY] Helsinki Region Environmental Services Authority HSY: http://www.hsy.fi/en/Pages/Default.aspx.

[Iaconesi and Persico, 2012] Iaconesi S.; Persico O., *"The Co-Creation of the City, re-programming cities using real-time user generated content"*, 1st Conference on Information Technologies for Performing Arts, Media Access and Entertainment, May 2012.

[Isele and Bizer, 2013] R. Isele, C. Bizer. *Active learning of expressive linkage rules using genetic programming*. Web Semantics: Science, Services and Agents on the World Wide Web 23 (2013): pp.2-15.

[Jenny, 2014] Thomas Jenny, Smart Cities: Mission Control, http://xlgroup.com/fast-fast-forward/articles/smart-cities-solve-urban-challenges, Oct. 2014.

[Karloff, Suri and Vassilvitskii, 2010] Karloff H.; Suri S.; Vassilvitskii S., "*Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*", Pages 938-948, 2010.

[Kauppinen et al., 2011]Kauppinen, T.; Espindola, G. M. D. "*Linked Open Science-Communicating, Sharing and Evaluating Data, Methods and Results for Executable Papers*". Procedia Computer Science 4: 726. doi:10.1016/j.procs.2011.04.076. edit, 2011.

[Klyne and Carrol, 2006] Klyne, G., & Carroll, J. J. *Resource description framework (RDF): Concepts and abstract syntax*, 2006.

[Komninos, 2002]Komninos N. *Intelligent Cities: Innovation, knowledge systems and digital spaces, London and New York, Taylor and Francis,* Spon Press, 2002.

[Liguria, 2014] Liguria, Cti. *La città digitale. Sistema nervoso della smart city: Sistema nervoso della smart city*. Vol. 44. FrancoAngeli, 2014.

[Linked Data Stars] Linked Open Data 5 Star: http://www.w3.org/DesignIssues/LinkedData.html.

[Liu, Biderman and Ratti, 2009] Liu L.; Biderman A.; Ratti C., "*Urban mobility landscape: Real time monitoring of urban mobility patterns*", Proceedings of the 11th International Conference on Computers in Urban Planning and Urban Management, 2009.

[LODClaud] the LOD cloud: http://lod-cloud.net/state, 2014.

[Mans et al., 2009] Mans R. S.; Schonenberg M. H.; Song M.; Van der Aalst W.M.P.; Bakker P.J.M., *"Application of Process Mining in Healthcare - A Case Study in a Dutch Hospital"*, Biomedical Engineering Systems and Technologies, Communications in Computer and Information Science, page 425-438, Volume 25, Part 4, 2009.

[Manyika et al, 2011] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. *Big data: The next frontier for innovation, competition, and productivity*. Technical report, McKinsey Global Institute, 2011.

[MIIC-AVM] MIIC - AVM Real-time Services - http://web.rete.toscana.it/eCompliance/portale/mostraRFC?idRev=758&idRfc=227.

[MIIC-DateX] MIIC - DATEX II Client Pull Service http://web.rete.toscana.it/eCompliance/portale/mostraRFC?idRev=757&idRfc=226.

[Mislove, Gummandi and Druschel, 2006] Mislove A.; Gummandi K.P.; Druschel P., *"Exploiting Social Networks for Internet Search"*, Record of the Fifth Workshop on Hot Topics in Networks: HotNets V, Pages 79-84, August 2006.

[Ngomo and Auer, 2011] A. Ngomo, S. Auer. *LIMES: a time-efficient approach for large-scale link discovery on the web of data*. Proc. of the 22nd int. joint conf. on Artificial Intelligence, Vol.3. AAAI Press, 2011.

[NIEM] NIEM - https://www.niem.gov/Pages/default.aspx.

[Obenshain, 2004] Obenshain M.K., *"Application of Data Mining Techniques to Healthcare Data"*, Infection Control and Hospital Epidemiology, Pages 690-695,Volume 25, Number 8, 2004.

[Ocha and Duval, 2006] Ochoa, X., & Duval, E. *Quality Metrics for learning object Metadata*. In Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2006 (pp. 1004-1011). 2006.

[OSIM] http://www.disit.org/drupal/?q=node/5519.

[Osservatorio Trasporti] http://www501.regione.toscana.it/osservatoriotrasporti.

[OWL] OWL - http://www.w3.org/TR/owl2-overview/.

[Papailiou et al., 2013] Papailiou, N., Konstantinou, I., Tsoumakos, D., Karras, P., & Koziris, N. *H2RDF+: High-performance distributed joins over large-scale RDF graphs*. In Big Data, 2013 IEEE International Conference on (pp. 255-263). IEEE. 2013, October.

[Piprani and Ernst, 2008] Piprani B. and Ernst D. *A Model for Data Quality Assessment*. On the Move to Meaningful Internet Systems: OTM 2008 Workshops. Lecture Notes in Computer Science Volume 5333, 2008, pp 750-759.

[Powers, 2011] Powers, D.M.W. *Evaluation from Precision, Recall and F-Measure to roc informedness, markedness and correlation*. Journal of Machine Learning Technologies 2 (1): 37–63. (February 27, 2011)

[Reed et al, 2012] D.A. Reed, D.B. Gannon, and J.R. Larus, *"Imagining the Future: Thoughts on Computing"*, Computer, vol. 45, no. 1, pp. 25-30, jan. 2012.

[RFC2616] RFC 2616 - HTTP/1.1 http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html.

[Rivals et al., 2012] Rivals E.; Philippe N.; Salson M.; Léonard M.; Commes T.; Lecroq T., *"A Scalable Indexing Solution to Mine Huge Genomic Sequence Collections"*, ERCIM News, Pages 20-21, Number 89, April 2012.

[Roussey et al., 2011] Roussey, C., Pinet, F., Kang, M. A., & Corcho, O. *An Introduction to Ontologies and Ontology Engineering. In Ontologies in Urban Development Projects* (pp. 9-38). Springer London, 2011.

[Rusitschka, Eger and Gerdes, 2010] Rusitschka S.; Eger K.; Gerdes C., *"Smart Grid Data Cloud: A Model for Utilizing Cloud Computing in the Smart Grid Domain"*, 1st IEEE International Conference of Smart Grid Communications, October 2010.

[SCAPE Project] SCAPE Project - http://scape-project.eu/.

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

[Semanco] Nemirovski, G., Nolle, A., Sicilia, Á., Ballarini, I., & Corado, V. *Data integration driven ontology design, case study smart city*. In Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics(p. 43). ACM. 2013, June.

[Sesame] Sesame - http://www.openrdf.org/.

[Setnes et al., 2001] Setnes, M., Roubos, H. *Compact and transparent fuzzy models and classifiers through iterative complexity reduction. Fuzzy Systems*, IEEE Transactions on, 9(4), 516-524, 2001.

[ShareAlike] Attribution-ShareAlike 3.0: https://creativecommons.org/licenses/by-sa/3.0/us/.

[Silvestri et al., 2012] Silvestri L., A. Bria, L. Sacconi, A. L. A. Mascaro, M. C. Pettenati, S. Bassini, C. Cavazzoni, G. Erbacci, R. Turra, G. Fiameni, V. Ruggiero, P. Frasconi, S. Marinai, M. Gori, P. Nesi, Re. Corradetti, G. Iannello, F. S. Pavone, *"Projectome: Set up and testing of a High Performance ComputationalInfrastructure for processing and visualizing neuro-anatomical information obtainedusingconfocal ultra-microscopytechniques"*, Neuroinformatics 2012 5th INCF Congress, September 2012.

[SKOS] SKOS - http://www.w3.org/2004/02/skos/.

[Smullyan, 1968] Smullyan, R. M. *First-order logic (Vol. 21968)*. Heidelberg: Springer, 1968.

[Sonntag, 2009] Sonntag Daniel. *Introspection and adaptable model integration for dialogue-based question answering*. In Proceedings of the 21st international jont conference on Artifical intelligence, pages 1549–1554, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc., 2009.

[SSN] Semantic Sensor Network Ontology - http://www.w3.org/2005/Incubator/ssn/ssnx/ssn.

[Starcity] Lécué, F., Tallevi-Diotallevi, S., Hayes, J., Tucker, R., Bicer, V., Sbodio, M. L., & Tommasi, P. *STAR-CITY: semantic traffic analytics and reasoning for CITY*. In Proceedings of the 19th international conference on Intelligent User Interfaces (pp. 179-188). ACM. 2014, February.

[TimeBase] Time Base Ontology https://www.niem.gov/Pages/default.aspx.

[Valduriez, Pacitti, 2005] Valduriez P.; Pacitti E., *"Data Management in Large-scale P2P Systems"*, High Performance Computing for Computational Science Vecpar 2004 - Lecture Notes in Computer Science, Volume 3402, 2005.

[W3C geo] Basic Geo (WGS84 lat/long) Vocabulary http://www.w3.org/2003/01/geo/.

[Waltinger et al., 2011] Waltinger U., Breuing A. and Wachsmuth I. *Interfacing virtual agents with collaborative knowledge: Open domain question answering using Wikipedia-based topic models.* In Proceedings of the 22nd International Joint Conference on Artificial Intelligence - IJCAI 2011, Barce, 2011.

[Waltinger et al., 2013] Waltinger U., Tecuci D., Olteanu M., Mocanu V. and Sullivan S. *Usi answers: Natural language question answering over (semi-) structured industry data.* In Muoz-Avila, Hector and Stracuzzi, David J., editors, IAAI. AAAI, 2013.

[Wang et al.,2009] Wang, J., Lu, J., Zhang, Y., Miao, Z. and Zhou, B. *Integrating Heterogeneous Data Source Using Ontology*. JOURNAL OF SOFTWARE, VOL. 4, NO. 8. 2009.

[Weiss et al., 2008] Weiss, C., Karras, P., & Bernstein, A. *Hexastore: sextuple indexing for semantic web data management*. Proceedings of the VLDB Endowment, 1(1), 1008-1019. 2008.

[Wilkinson et al, 2003] Wilkinson, K., Sayers, C., Kuno, H. A., & Reynolds, D. *Efficient RDF Storage and Retrieval in Jena2*. In SWDB (Vol. 3, pp. 131-150). 2003, September.

[Woolf, Baker and Gianchandani, 2010] Woolf B.P.; Baker R.; Gianchandani E.P., *"Enabling Personalized Education"*, Computing Community Consortium, Version 9, September 2010.

[XPATH] W3C - XML Path Language (XPath) http://www.w3.org/TR/xpath.

[Zaslavsky et al, 2013] Zaslavsky, A., Perera, C., & Georgakopoulos, D. *Sensing as a service and big data.* arXiv preprint arXiv:1301.0159, 2013.

[Zenith] Zenith - http://www-sop.inria.fr/teams/zenith/.

[Zikopoulos, 2012] Paul Zikopoulos. (2012, March) *IBM Big Data: What is Big Data Part 1 and 2.* [Online]. http://www.youtube.com/watch?v=B27SpLOOhWw [Accessed on: 2012-06-08].

[Zinterhof, 2012] Zinterhof P., *"Computer-Aided Diagnostics"*, ERCIM News, Page 46, Number 89, April 2012.