



UNIVERSITÀ DEGLI STUDI DI FIRENZE
FACOLTÀ DI INGEGNERIA - DIPARTIMENTO DI INGEGNERIA
DELL'INFORMAZIONE

Corso di Laurea Magistrale in Ingegneria Informatica

**ANALISI E SVILUPPO DI UN SISTEMA DI ACQUISIZIONE,
RAPPRESENTAZIONE ED ACCESSO DI INFORMAZIONI
GEOGRAFICHE E DI SERVIZIO, STATICHE ED IN TEMPO
REALE, PER SMART CITY**

ANALYSIS AND DEVELOPMENT OF A SYSTEM FOR
ACQUISITION, REPRESENTATION AND ACCESS OF
REAL-TIME AND STATIC SERVICES AND GEOGRAPHICAL
DATA FOR SMART CITY

Candidato

Lapo Bicchielli

Relatori: Prof. Paolo Nesi, Prof. Carlo Colombo

Correlatore: Ing. Nadia Rauch

ANNO ACCADEMICO 2012–2013

Indice

1	Introduzione	1
1.1	Smart City e Mobilità	1
1.2	Progetto Sii-Mobility	2
1.3	Scopo del lavoro	4
2	Analisi dei Dati e dell'Architettura	6
2.1	Sorgenti e tipi di dati	6
2.1.1	Grafo Stradale	7
2.1.2	Open Data Comune di Firenze e Regione Toscana	10
2.1.3	MIIC (Mobility Information Integration Center)	11
2.2	Analisi dell'Architettura	11
2.3	Dettaglio del Lavoro e Casi d'Uso	14
3	Progettazione	17
3.1	Costruzione dell'Ontologia SmartCity Ontology	18
3.1.1	Stato dell'arte	18
3.1.2	Ontologia SmartCity Ontology	22
3.1.2.1	Grafo Stradale	24
3.1.2.2	Pubblica Amministrazione	38
3.1.2.3	Trasporto Pubblico Locale	40
3.1.2.4	Servizi	44
3.1.2.5	Sensoristica	48

3.1.2.6 Sezione Temporale	58
3.2 Fase di Ingestion	60
3.2.1 Estrazione e Manipolazione Dati	63
3.2.2 Definizione modello R2RML e Generazione Triple RDF . .	97
3.2.3 Creazione Repository Semantico ed Inserimento delle Tri- ple RDF	108
4 Analisi e Riconciliazione dei dati	112
4.1 Riconciliazione Toponimi	114
4.2 Riconciliazione dei Servizi con gli Accessi	116
4.3 Riconciliazione Fermate degli Autobus	122
5 Interrogazione e visualizzazione dei dati	125
5.1 Visualizzazione tramite ServiceMap	126
5.1.1 Caso d'uso 1: ricerca Servizi appartenenti ad un Comune	134
5.1.2 Caso d'uso 2: ricerca Servizi prossimi a Fermate Autobus	142
5.1.3 Caso d'uso 3: ricerca Servizi prossimi ad un punto generico	152
5.2 Visualizzazione tramite Linked Open Graph	157
6 Conclusioni e sviluppi futuri	161
6.1 Risultati	162
6.2 Sviluppi Futuri	163
A Web Semantico	166
A.1 RDF - Resource Description Framework	167
A.2 Ontologie e OWL - Ontology Web Language	168
A.3 SPARQL - SPARQL Protocol and RDF Query Language	169
A.4 Open Data - Linked Data	170
B Software Utilizzati	171
B.1 PENTAHO Kettle	171
B.2 QGIS (Quantum GIS)	173
B.3 KARMA Data Integration Tool	174
B.4 OpenRDF Sesame	175

<i>INDICE</i>	iii
B.5 OWLIM	176
Bibliografia	177

Elenco delle figure

2.1	Rappresentazione grafica dell'intero progetto	13
2.2	Modello dei principali casi d'uso previsti per la visualizzazione dei dati	15
3.1	Grafico di rappresentazione dell'intero sistema progettato	17
3.2	Principali macroclassi dell'ontologia OTN	20
3.3	Rappresentazione grafica dell'intera ontologia SmartCity Ontology	23
3.4	Rappresentazione grafica della sezione Grafo Stradale dell'ontologia SmartCity Ontology	25
3.5	Esempio di differenti rapporti di cardinalità tra Estesa Amministrativa e Toponimo	33
3.6	Esempio di Estesa Amministrativa formata da differenti Toponimi	35
3.7	Rappresentazione grafica delle informazioni contenute all'interno degli ShapeFile Elementi Stradali	37
3.8	Macro-sezione Pubblica Amministrazione dell'ontologia SmartCity Ontology	38
3.9	Macro-sezione Trasporto Pubblico Locale dell'ontologia SmartCity Ontology	40
3.10	Esempio di Percorsi differenti su una stessa Linea - Immagini prelevate dal sito ATAF - Mappa di Google Maps	42
3.11	Macro-sezione Servizi e Attività dell'ontologia SmartCity Ontology	45

3.12 Macro-sezione Previsioni Meteorologiche dell'ontologia SmartCity Ontology	49
3.13 Macro-sezione Parcheggi dell'ontologia SmartCity Ontology	52
3.14 Macro-sezione Sensori dell'ontologia SmartCity Ontology	54
3.15 Macro-sezione AVM dell'ontologia SmartCity Ontology	56
3.16 Macro-sezione Temporale dell'ontologia SmartCity Ontology	59
3.17 Rappresentazione grafica del processo di Ingestion dei dati	61
3.18 Rappresentazione grafica del processo globale di Ingestion attualmente in funzione per il processo Sii-Mobility	62
3.19 Visualizzazione tramite Spoon del job principale della macrosezione Grafo Stradale	64
3.20 Script per la conversione di file da .shp a .kml e da Gauss-Boaga a WGS84	65
3.21 Dettaglio dello step Execute a shell script all'interno dell'ambiente Kettle	66
3.22 Dettaglio della trasformazione Get_Folders.ktr	67
3.23 Dettaglio dello step Get SubFolder Names	67
3.24 Impostazioni del job main_job.kjb	68
3.25 Dettaglio del job Main_Job.kjb	68
3.26 Dettaglio della trasformazione set_variables.ktr	69
3.27 Dettaglio della trasformazione Province.ktr	69
3.28 Dettaglio della trasformazione Road_Element.ktr	71
3.29 Procedura di mappatura degli attributi della classe RoadElement nei corrispondenti valori di dominio	72
3.30 Dettaglio della trasformazione Road.ktr	73
3.31 Dettaglio della trasformazione Administrative_Road.ktr	74
3.32 Dettaglio della trasformazione Street_Number.ktr	75
3.33 Dettaglio della trasformazione Maneuver.ktr	75
3.34 Dettaglio della trasformazione Entry_Rule.ktr	77
3.35 Dettaglio della trasformazione Entry_Rule2.ktr	79
3.36 Impostazioni dello step dom_tip_prp.dbf che mappa il contenuto dell'attributo RESTRVAL nei corrispondenti valori di dominio	79
3.37 Dettaglio della trasformazione Municipality.ktr	80

3.38 Dettaglio della trasformazione Open_KML_File.ktr	81
3.39 Impostazioni degli step Split fields per il recupero delle singole informazioni all'interno dei dati KML	82
3.40 Dettaglio delle trasformazioni Entry.ktr, Node.ktr e Milestone.ktr	83
3.41 Dettaglio della trasformazione Open_Coordinates.ktr	84
3.42 Dettaglio della trasformazione Process_Coordinates.ktr	84
3.43 Impostazioni dello step Split field to Rows grazie al quale si crea una nuova riga per ogni coordinata dell'elemento stradale	85
3.44 Esempio di elaborazione dello step Split field to rows	85
3.45 Dettaglio della trasformazione Save_Coordinates.ktr	87
3.46 Impostazioni ed esempio dello step Row flattener	88
3.47 Impostazioni dello step Delete temporary files per la cancellazione dei file temporanei	89
3.48 Dettaglio del job MySQL_To_RDF.kjb	89
3.49 Impostazioni dello step Generate RDF per la generazione delle triple RDF a partire da tabelle MySQL	90
3.50 Impostazioni dello step Truncate MySQL Tables che svuota le tabelle MySQL di appoggio generate dalle trasformazioni precedenti	91
3.51 Dettaglio del job Filter_Duplicates.kjb	91
3.52 Dettaglio del job Filter_Administrative_Road.kjb	92
3.53 Esempio di triple generate da KARMA relative a Administrative Road	93
3.54 Dettaglio del job Split_Big_Files.kjb	95
3.55 Impostazioni dello step Write Instructions per la generazione del file contenente le istruzioni per il caricamento dei file sul repository OWLIM	96
3.56 Impostazioni dello step Load Triples per il caricamento definitivo dei dati su repository RDF	97
3.57 Interfaccia web iniziale del software KARMA	98
3.58 Finestra di impostazione di parametri KARMA per l'importazione di dati da database	99
3.59 Importazione dei dati della tabella tbl_manovre all'interno dell'interfaccia web di KARMA	99

3.60 Impostazioni di mappatura della colonna ID_MAN della tabella tbl_manovre	100
3.61 Associazione colonna ID_MAN alla proprietà dc:identifier della classe Sii-Mobility:Maneuver	101
3.62 Associazione colonna VIA_GNZ alla proprietà dc:identifier della classe Sii-Mobility:Node e creazione del collegamento tra le classi	102
3.63 Definizione dell'ObjectProperty Sii-Mobility:concerning tra le clas- si Node e Maneuver	103
3.64 Esempio di mappatura completa della tabella tbl_manovre	104
3.65 Esempio di file R2RML generato da KARMA	105
3.66 Esempio di script per la generazione batch delle triple RDF per la tabella MySQL tbl_accesso	106
3.67 Esempio di triple RDF generate in modalità batch da KARMA dalla tabella tbl_el_stradale	107
3.68 Interfaccia web del software OWLIM per la generazione di un nuovo repository	109
3.69 Esempio di esecuzione della console di OWLIM	110
4.1 Grafico di rappresentazione della fase di Riconciliazione dei Dati .	113
4.2 Tabella MySQL di appoggio per la riconciliazione dei toponimi . .	115
4.3 Query SPARQL per la ricerca all'interno del repository di una corrispondenza per l'indirizzo VIA DELLA VIGNA NUOVA 40/R- 42/R, FIRENZE	118
4.4 Esempi di riconciliazione tra indirizzo e accesso	119
4.5 Query SPARQL per la ricerca all'interno del repository di tutte le vie del comune di ABBADIA SAN SALVATORE contenenti la parola TRENTO	120
4.6 Procedura di riconciliazione manuale dei Servizi attraverso ricerca dell'ultima parola	121
4.7 Esempio di query SPARQL per il recupero della Road a cui ap- partiene una BusStop	123
5.1 Interfaccia web di Sesame per l'interrogazione del repository con query SPARQL	126

5.2	Interfaccia attualmente in funzione per il servizio ServiceMap	127
5.3	Rappresentazione grafica dell'interrogazione e della visualizzazione dei dati tramite ServiceMap	128
5.4	Dettaglio del menu in alto a sinistra dell'interfaccia per ServiceMap	130
5.5	Dettaglio del menu in alto a destra dell'interfaccia per ServiceMap	131
5.6	Dettaglio del menu in basso a sinistra dell'interfaccia per ServiceMap	133
5.7	Query SPARQL per la ricerca dei Servizi di tipo FARMACIA e GUARDIA MEDICA nel comune di EMPOLI	135
5.8	Dati recuperati dalla query SPARQL di figura 5.7	136
5.9	Esempio di dati JSON corrispondenti ai risultati della query di figura 5.7	137
5.10	Risultati della ricerca di servizi nel Comune di EMPOLI	138
5.11	Diagramma di sequenza client-server per il caso d'uso numero 1	139
5.12	Query SPARQL per il recupero delle previsioni meteo più recenti per il comune di LUCCA	140
5.13	Visualizzazione nel menu contestuale di ServiceMap delle previsioni del tempo per il comune di LUCCA	141
5.14	Selezione di una fermata dell'autobus e visualizzazione dei corrispondenti dati AVM	143
5.15	Selezione delle categorie di servizi di interesse, del numero massimo di risultati e della distanza massima a cui ricercare i servizi .	144
5.16	Risultati della ricerca di Servizi prossimi ad una Fermata Autobus	145
5.17	Query SPARQL per la ricerca dei Servizi di tipo FARMACIA prossimi alla fermata STAZIONE SCALETTE	146
5.18	Dati recuperati dalla query SPARQL di figura 5.17	147
5.19	Esempio di dati JSON corrispondenti ai risultati della query di figura 5.17	147
5.20	Visualizzazione su mappa dei risultati ottenuti in risposta alla query di figura 5.17	148
5.21	Diagramma di sequenza client-server per il caso d'uso numero 2	149
5.22	Query SPARQL per la prima fase di recupero di dati AVM sulla fermata STAZIONE SCALETTE	150
5.23	Primi risultati della query di figura 5.22	151

5.24 Query SPARQL per la seconda fase di recupero di dati AVM sulla fermata STAZIONE SCALETTE	151
5.25 Visualizzazione su ServiceMap delle previsioni di transito sulla fermata autobus FRATELLI ROSSELLI	152
5.26 Dati sull'occupazione del parcheggio G.Guerra di Empoli	154
5.27 Query per il recupero dell'ultima situazione conosciuta riguardante l'occupazione del parcheggio Stazione Binario 16	155
5.28 Esempio di ricerca di un indirizzo approssimativo di un punto sulla mappa	156
5.29 Query SPARQL per la ricerca di un indirizzo approssimativo a partire da un punto sulla mappa	157
5.30 Esempio di visualizzazione tramite il software LOG dei dati RDF a partire dalla fermata dell'autobus FM0275 - ANTONIO DEL POLLAIOLI 05	159
A.1 Esempio di informazioni in formato di triple RDF	167
A.2 Query di esempio SPARQL	169

Elenco delle tabelle

2.1	Esempio di contenuto del file dom_tip_gnz.dbf	10
4.1	Elenco dei risultati ottenuti in risposta alla query di figura 4.7 . .	124
6.1	Risultati dei singoli processi di riconciliazione dei servizi e risultati totali	163

Capitolo 1

Introduzione

1.1 Smart City e Mobilità

Il termine **Smart City** è un vocabolo utilizzato, al giorno d'oggi, per descrivere tutte quelle strategie di pianificazione urbana che mirano al miglioramento della qualità della vita dei cittadini. Questo obiettivo è perseguito mediante l'utilizzo di tecnologie innovative di comunicazione, che permettono di collegare direttamente i cittadini con le infrastrutture materiali della città. Tra i vari settori che determinano se una città può essere considerata o meno una Smart City, vi è quello della **Mobilità e dei Trasporti**. Il settore della Mobilità coinvolge milioni di persone ogni giorno, e permette a lavoratori, studenti, turisti, e a molte altre tipologie di persone, di potersi spostare con facilità tramite **veicoli privati**, oppure utilizzando il **trasporto pubblico locale**. Ad oggi, i sistemi di trasporto presentano molto spesso situazioni di congestione o di scarsa interoperabilità tra agenti diversi. Questo comporta un peggioramento della qualità della vita dei cittadini che spesso si trovano a dover fare i conti con traffico, disservizi, eccetera.

Un primo passo che può essere effettuato è quello di utilizzare l'impor-

tante mole di dati che, periodicamente, viene rilasciato sotto forma di **Open Data**, dalle **Pubbliche Amministrazioni**: ad oggi, infatti, sono reperibili in rete molteplici dataset, contenenti informazioni riguardanti il settore dei trasporti ed i contesti adiacenti (ad esempio le ordinanze comunali, i dati relativi a servizi e attività, le previsioni del tempo, eccetera). Il Comune di Firenze e la regione Toscana, sono tra le realtà più attive in Italia nella pubblicazione di questi dati. L'elaborazione di questi singoli dataset in maniera distinta, tuttavia, non comporta molti vantaggi. Di contro, si ottengono risultati di interesse nel momento in cui questi dati vengono combinati tra loro, ma soprattutto quando essi vengono fusi con i dati, statici ed in real-time, provenienti da servizi privati di gestione e monitoraggio dei trasporti. Tutti questi dati, opportunamente elaborati ed immessi in una struttura semantica di memorizzazione ed interrogazione facilmente interpretabile, permettono di creare la base per lo sviluppo di applicazioni in grado di rendere una città *intelligente*.

Per realizzare un sistema di questo tipo, è necessario definire una base di conoscenza, o **ontologia** integrata, che permetta di gestire la riconciliazione dei dati provenienti da fonti diverse e la complessità e la semantica degli stessi. Ad oggi, non esistono delle ontologie standard per le Smart City, ed è quindi necessario un grande sforzo rivolto alla costruzione di una ontologia ad hoc.

1.2 Progetto Sii-Mobility

In questo contesto, si affaccia il progetto **Sii-Mobility**, promosso dal **MIUR (Ministero dell'Istruzione, dell'Università e della Ricerca)**, che prevede la realizzazione di un sistema per migliorare il trasporto e la mobilità e per fornire in tempo reale informazioni utili a cittadini, imprese e pubbliche amministrazioni. Il progetto è risultato tra i vincitori del bando **Smart Cities and Communities**, proprio per l'area *Trasporti e Mobilità Terrestre*. In questo contesto, il laboratorio **DISIT (Distributed Data Intelligence and Technologies)**, facente parte del *Dipartimento di Ingegneria*

dell'Informazione, è partner tecnico e scientifico del progetto.

Sii-Mobility è un progetto molto vasto, all'interno del quale, questa tesi copre un piccolissimo ambito. Il progetto Sii-Mobility si prefigge come obiettivo la costruzione di una piattaforma in grado di fornire informazioni e supporto a cittadini, imprese e pubbliche amministrazioni. Attraverso la soluzione Sii-Mobility, sarà possibile fruire di una vasta gamma di applicazioni tra le quali, ad esempio:

- **Soluzioni di guida personalizzata:** spostamenti intelligenti con e senza mezzi, informazioni sul traffico, eccetera.
- **Gestione personalizzata delle politiche di accesso:** utilizzo di politiche di incentivo e dissuasione del veicolo, mobilità dei crediti, flussi di monitoraggio.
- **Piattaforma di partecipazione e sensibilizzazione:** invio di informazioni personalizzate al cittadino mediante applicazioni web, mobili, totem, e ricezione da parte dello stesso di segnalazioni, informazioni, eccetera.
- **Monitoraggio in tempo reale del trasporto pubblico:** domanda e offerta di servizi TPL, integrazione ed elaborazione dei dati.
- **Gestione delle aree ZTL:** modifica dinamica dei confini in base a situazione di congestione, inquinamento, eccetera.
- **Interoperabilità ed integrazione dei sistemi di gestione:** definizione di standard, riconciliazione di dati, verifiche e validazioni, eccetera.

Una volta ottenuti questi risultati, sarà possibile raggiungere gli obiettivi prefissati di Sii-Mobility, ovvero la riduzione del costo sociale della mobilità (inversamente proporzionale al miglioramento della qualità della vita dei cittadini), la semplificazione nell'utilizzo dei sistemi di trasporto, ed infine la standardizzazione nei sistemi di gestione per Smart City e la loro interoperabilità verso altri sistemi esterni simili, italiani o europei.

1.3 Scopo del lavoro

Il lavoro principale di questa tesi si è sviluppato all'interno del progetto Sii-Mobility appena descritto e ne rappresenta una minima parte iniziale.

L'obiettivo principale di questa tesi è quello di generare un repository **semantico** in formato **RDF**, che contenga dati provenienti da molteplici fonti di tipo Public Data (in particolare **Open Data**) e Private Data. L'intero progetto può essere racchiuso in 4 macro-fasi:

- **Costruzione di una Ontologia chiamata SmartCity Ontology per modellare i dati di una Smart City**
- **Acquisizione ed Inserimento Dati o Ingestion**
- **Riconciliazione e Validazione dei Dati**
- **Interrogazione e Visualizzazione dei Dati**

Nella prima fase si è costruita un'**ontologia** ad hoc per mappare i dati in un modello adatto alla gestione del problema della Mobilità e di tutti gli aspetti contigui.

Durante la fase di **Ingestion** si sono progettate le trasformazioni di tipo ETL che i dati sorgente dovevano necessariamente subire per poter essere utilizzabili all'interno dell'ontologia. Inoltre, è stato progettato il sistema di conversione dei dati in triple RDF.

Nella terza fase, quella di **Riconciliazione e Validazione**, si sono analizzati i dati immessi nel repository e ne sono state estratte le inconsistenze. A partire da esse, si è provveduto a riconciliare i dati che presentavano dei problemi.

Infine, una volta che i dati hanno iniziato a fluire correttamente all'interno del repository, si è realizzata una interfaccia web, che tramite query **SPARQL**, restituisse dei risultati in risposta a interrogazioni geografiche basate su una **mappa interattiva**.

Il lavoro di questa tesi è suddiviso come segue: nel capitolo 2 viene riportata l'analisi dei dati e un primo schema di progettazione. Nel capitolo

Scopo del lavoro

3 viene descritta in dettaglio la progettazione dell'intero sistema, a partire dalla costruzione dell'ontologia, passando per la fase di Ingestion suddivisa in: definizione delle trasformazioni ETL, generazione delle triple RDF tramite modello R2RML e inserimento delle triple sul repository. Nel capitolo 4 vengono analizzati i dati inseriti e proposte alcune tecniche di riconciliazione dei dati inconsistenti. Nel capitolo 5 vengono descritte in dettaglio la progettazione e l'uso delle principali applicazioni di visualizzazione dei dati, infine nel capitolo 6 vengono presentate le conclusioni ed i risultati di questo progetto ed i possibili sviluppi futuri.

Capitolo 2

Analisi dei Dati e dell'Architettura

Prima di procedere con la progettazione del sistema, sono stati analizzati in dettaglio i vari dati a disposizione, ed è stata schematizzata una bozza del sistema che si sarebbe andati a costruire.

2.1 Sorgenti e tipi di dati

L'idea originale del progetto prevede che la struttura portante del repository finale sia formata dai dati facenti parte del cosiddetto **Grafo Stradale** della regione Toscana. Esso comprende le informazioni relative all'intera rete viaria toscana, comprensiva di strade, toponimi, numeri civici, eccetera.

Successivamente, alla struttura di base del Grafo Stradale, sarebbero stati associati molti altri dati di interesse legati alla mobilità ed ai trasporti, quali ad esempio dati relativi al **Trasporto Pubblico Locale**, alla situazione dei **Parcheggi**, alle **Previsioni Meteo**, eccetera.

2.1.1 Grafo Stradale

I dati strutturali di base sui quali è nato il lavoro provengono dall’Osservatorio Regionale per la Mobilità ed i Trasporti della Regione Toscana¹. Tramite questi dati è stato possibile costruire l’ossatura del progetto, ovvero il **Grafo Stradale** della regione Toscana. Il Grafo Stradale è la rappresentazione del reticolo stradale regionale ed è organizzato seguendo la struttura propria dei Grafi ovvero con una base formata da *Archi* (Elementi Stradali) e *Nodi* (Giunzioni Stradali). A queste strutture vengono associate molteplici informazioni quali Toponimi Stradali (insiemi di Elementi Stradali raggruppati secondo alcuni criteri all’interno di singoli comuni), Indirizzi (tramite Accessi e Numeri Civici), eccetera. Maggiori dettagli sono descritti più avanti in questo paragrafo.

Poichè i dati riguardanti le infrastrutture di trasporto (Strade, Ferrovie, Numerazione Civica) non cambiano frequentemente, l’osservatorio non prevede la possibilità di scaricare i dati tramite web service, ma solo tramite interfaccia web, previa autenticazione mediante un certificato rilasciato al laboratorio. Quindi, al momento, non è possibile implementare un servizio che automaticamente effettui il download di nuovi dati, ma deve essere l’utente a richiederne l’invio manualmente.

I dati contenuti all’interno degli archivi recuperati dall’osservatorio sono molteplici; in particolare, nella versione rilasciata durante lo sviluppo del progetto, sono presenti due categorie principali di dati: il Data Pack Indirizzario (DPI) ed il Data Pack Accessorio (DPA). Il Data Pack Indirizzario contiene tutte le informazioni necessarie a costruire il Grafo Stradale, tra queste troviamo:

- **Elementi Stradali:** entità lineari delimitate da due Giunzioni. Gli Elementi Stradali sono la componente di base dell’intero Grafo Stradale, e sono composti da un insieme ordinato di punti.

¹Sito di Supporto ai Lavori di Gestione e Aggiornamento delle Banche Dati dell’Osservatorio Regionale per la Mobilità ed i Trasporti - <http://www501.regione.toscana.it/osservatoriotrasporti/>

- **Giunzioni:** punti di intersezione tra gli assi di due Elementi Stradali. Dal punto di vista geometrico, una Giunzione è un'entità puntuale (detta anche Nodo) rappresentata da una coppia di coordinate.
- **Toponimi:** dalla definizione recuperata dai manuali della Regione si legge che un Toponimo Stradale *corrisponde ad una porzione della rete della mobilità, alla quale è assegnato un certo nome da un dato Comune, nome che fa parte dello stradario comunale*. In pratica, l'entità Toponimo è definita come un insieme di Elementi Stradali di uno stesso Comune aggregati secondo il nome assegnato dal Comune stesso.
- **Estese Amministrative:** anche in questo caso si tratta di insiemi di Elementi Stradali. Tuttavia, nel caso specifico dell'Estesa Amministrativa il raggruppamento viene effettuato in base a criteri amministrativi. Per maggiori dettagli sulla differenza tra Toponimo ed Estesa si rimanda alla sezione 3.1.2.1.
- **Numeri Civici:** come facilmente intuibile, l'elemento Numero Civico serve a definire uno specifico indirizzo all'interno di una strada. In realtà, a dispetto di quanto sia immaginabile, all'entità Numero Civico non è associata direttamente una componente spaziale. Questo perchè ad ogni Numero Civico è associata una entità di tipo Accesso.
- **Accessi:** elementi puntuali (quindi dotati di coordinate geografiche) che identificano un accesso ad uno specifico luogo di residenza o attività situato ad un determinato Numero Civico.
- **Cippi Chilometrici:** elemento puntuale che contiene il valore della chilometrica di una Estesa Amministrativa rispetto al punto di inizio.

Il Data Pack Accessorio, presente solo dall'ultima versione, contiene invece i seguenti tipi di dati:

- **Punti di Interesse**
- **Incroci Semaforizzati**

- Limitazioni di Accesso agli Elementi Stradali
- Svolte Proibite (o Consentite)

Al momento della stesura di questa tesi, in realtà, questo secondo insieme di dati è stato rilasciato soltanto in parte. Per quanto riguarda lo sviluppo del progetto sono stati utilizzati esclusivamente i dati relativi a **Manovre** consentite o proibite e a **Regole di Accesso** su determinati Elementi Stradali o Manovre. Per tutte le altre tipologie di informazioni, i dati rilasciati non sono presenti, oppure sono in quantità talmente limitate da aver fatto propendere per un loro impiego soltanto in sviluppi futuri del progetto.

Per quanto riguarda il formato dei dati, all'interno degli archivi i file contenenti informazioni geografiche sono codificati in **Shapefile ESRI**, ovvero in file con estensione *.shp*, i quali a loro volta sono necessariamente collegati a file *.shx* e *.dbf*. Gli Shapefile sono file contenenti informazioni vettoriali utilizzati prevalentemente per sistemi informativi geografici. Il file *.shp* conserva le geometrie, il file *.shx* memorizza l'indice delle geometrie, mentre all'interno del file *.dbf* è presente l'elenco degli oggetti assieme a tutti i loro attributi. I tipi di dati che possono essere salvati all'interno di uno Shapefile sono **Punti** (usato ad esempio per la classe Giunzione), **Poli-linee** (insieme di spezzate collegate e ordinate, come ad esempio gli Elementi Stradali) e **Poligoni** (usati, ad esempio, nella rappresentazione dell'estensione di un Comune o di una Provincia).

Tutti gli altri dati, quelli cioè che non posseggono informazioni geografiche, sono rilasciati con estensione *.dbf*, un formato attraverso il quale vengono salvati database di tipo **xBase**. Tali file sono facilmente leggibili da software per fogli di calcolo quali Microsoft Excel o OpenOffice Calc.

Per quanto riguarda la composizione dei dati si hanno due distinti tipi di tavole: le tavole con i dati effettivi sono contrassegnate dalle iniziali **GIA** (ad esempio **GIA_ACCESSO.dbf**, **GIA_EL_STRADALE.shp**, eccetera), mentre i file le cui iniziali sono **DOM** sono le cosiddette tavole di dominio, all'interno delle quali sono memorizzati tutti i possibili valori che può avere un particolare attributo dei dati. Ad esempio, all'interno della tabella **GIA_GIUNZIONE.dbf**, per ogni record è presente l'attributo

TIP_GNZ, che indica il tipo di giunzione in formato numerico. All'interno del file **dom_tip_gnz.dbf** sono presenti tutti i valori che può assumere tale campo, con la relativa descrizione testuale. Di seguito, nella tabella 2.1, viene mostrato un esempio del contenuto del file **dom_tip_gnz.dbf**.

Valore	Descrizione
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 / titolarità / 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

Tabella 2.1. Esempio di contenuto del file **dom_tip_gnz.dbf**

2.1.2 Open Data Comune di Firenze e Regione Toscana

Oltre ai dati provenienti dall'Osservatorio dei Trasporti sono stati utilizzati anche Open Data provenienti da altre due fonti: dal Comune di Firenze² e dalla Regione Toscana³. Gran parte di questi dati sono stati lavorati da un progetto parallelo [1]. In alcuni casi, tuttavia, si è reso necessario un lavoro aggiuntivo per adattarli al lavoro svolto in questa tesi e per farli convergere

²OpenData Comune di Firenze - <http://opendata.comune.fi.it/>

³OpenData Regione Toscana - <http://dati.toscana.it/>

in maniera corretta, all'interno del repository RDF globale. Pertanto, i dati provenienti da queste fonti alternative sono descritti in maniera sommaria.

Dalla Regione Toscana è stato prelevato un set di dati contenente informazioni riguardanti un gran numero di **Servizi** presenti sul territorio (suddivisi in categorie principali quali *Arte e Cultura*, *Banche*, *Emergenze*, *Enogastronomia*, eccetera) ed un set contenente invece informazioni dettagliate sul **Meteo** in Toscana (suddiviso in *Bollettini* e *Previsioni*) fornito dal consorzio LaMMA (Laboratorio di Monitoraggio e Modellistica Ambientale per lo sviluppo sostenibile). Nel primo caso i file forniti sono in formato CSV, mentre nel caso dei dati meteo il formato utilizzato è l'XML. I dati provenienti dal Comune di Firenze invece contengono informazioni sulla linea del Tram di Firenze (in formato KMZ) e statistiche sul territorio, anch'essi in formato CSV.

2.1.3 MIIC (Mobility Information Integration Center)

L'ultima sorgente dati utilizzata ai fini di questa tesi è il MIIC (Mobility Information Integration Center), un progetto della Regione Toscana che si occupa della raccolta in tempo reale di informazioni relative a trasporto pubblico, traffico, emergenze. Anche in questo caso, gran parte del lavoro è stato svolto da un progetto parallelo [2], quindi di seguito è fornita solo una breve descrizione dei dati di interesse. I dati provenienti dal MIIC riguardano principalmente **Sensori** (sia *Stradali* che *Meteorologici*), **Parcheggi** e **AVM** (Automatic Vehicle Monitoring), ovvero informazioni riguardanti i mezzi pubblici dell'area metropolitana di Firenze. In tutti questi ultimi casi i dati provengono da Web Service accessibili in parte liberamente ed in parte mediante registrazione e sottoscrizione. Il formato dei dati è sempre XML.

2.2 Analisi dell'Architettura

L'obiettivo di base di questa tesi è quello di produrre un sistema che permetta la memorizzazione di dati di varie tipologie e provenienti da fonti diverse.

In particolare, si vorrebbero poter memorizzare sia dati di tipo statico che dati Real-time. Questi dati dovrebbero attraversare una serie di elaborazioni atte a trasformarli in un formato adatto alla successiva trasformazione in formato semantico RDF. Per garantire uno storage permanente in grado di contenere una mole sempre crescente di dati, si è pensato di fare ricorso ad un database di tipo NoSQL intermedio. Una volta memorizzati su tale store, i dati sarebbero nuovamente caricati e mappati in RDF tramite dei software di tipo **Data Integration Tool**. L'intera procedura di richiesta, elaborazione e salvataggio dei dati dovrebbe essere regolata attraverso un processo di scheduling che, periodicamente, elabori nuovi dati e li immetta all'interno di un repository semantico. In contemporanea, si vuole definire una ontologia ad ampio raggio, che permetta di modellare tutti i dati provenienti dalle varie fonti, in un formato funzionale ai requisiti di una Smart City. Ovviamente, non è possibile evitare una analisi approfondita sui dati ed eventualmente, una loro riconciliazione, nel caso in cui vengano evidenziate delle inconsistenze. Infine, il progetto prevede di creare una o più interfacce utente, attraverso le quali si possa interrogare il repository e navigarci all'interno per ottenere le informazioni richieste.

Lo schema del progetto che si vuole andare a realizzare è mostrato in figura 2.1.

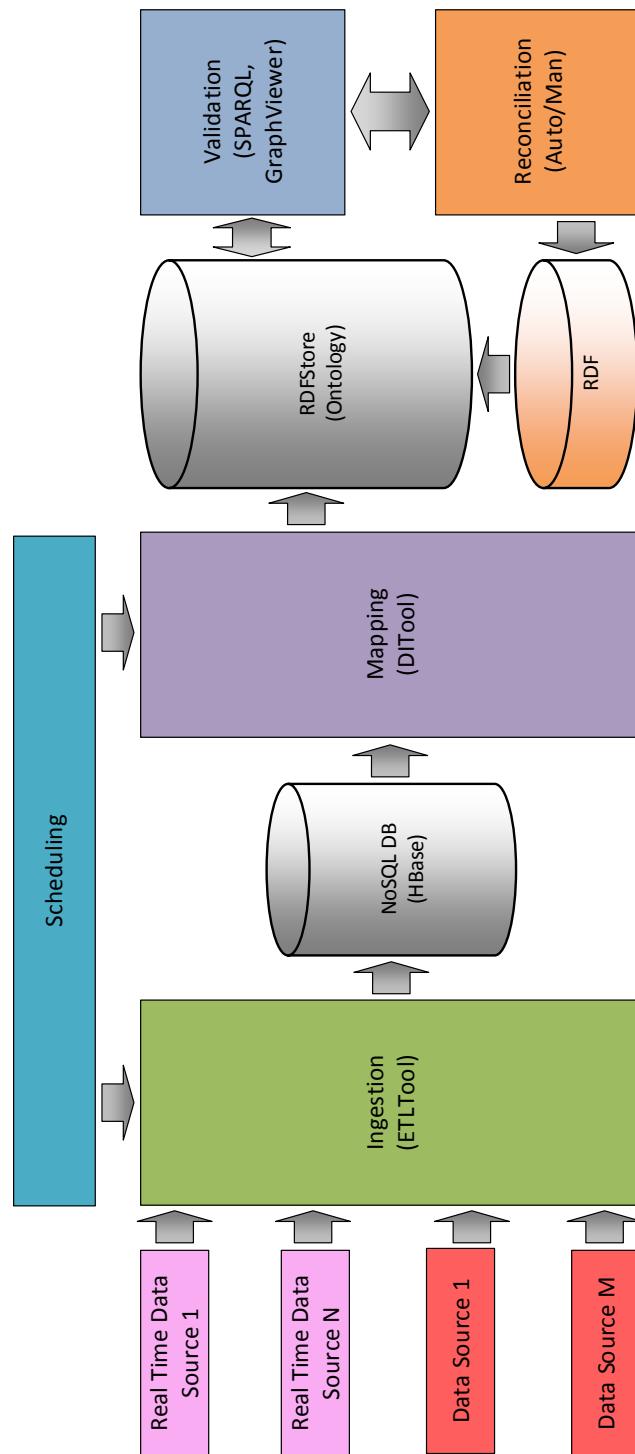


Figura 2.1. Rappresentazione grafica dell'intero progetto

2.3 Dettaglio del Lavoro e Casi d'Uso

Nel contesto appena presentato, in questa tesi ci si occupa principalmente della fonte di dati primaria che costituisce l'ossatura dell'ontologia, ovvero il **Grafo Strade**. Le fasi di Ingestion descritte nel prossimo capitolo riguardano esclusivamente i dati provenienti dall'Osservatorio Regionale per la Mobilità ed i Trasporti della Regione Toscana. I dati ottenuti dalle altre fonti (Comune di Firenze, Regione Toscana, eccetera), al momento della stesura di questa tesi, erano già stati lavorati. Per questo lavoro, ci si è occupati di una loro manipolazione marginale, al fine di renderli compatibili con i dati facenti parte del **Grafo Strade**. Di seguito, tali operazioni non sono descritte. Pertanto, in riferimento alla figura 2.1 dell'architettura generale, in questa tesi si è lavorato direttamente solo su uno dei flussi di input iniziali (**Data Source 1**) e non è stato direttamente sviluppato il processo di scheduling. Inoltre, i dati elaborati in questa tesi non passano attraverso un database NoSQL, ma sono direttamente salvati, temporaneamente, in un database MySQL.

Come detto, in alcuni casi, i processi di Ingestion relativi ai dati delle fonti alternative al grafo strade sono stati ritoccati per far confluire correttamente i dati all'interno del repository. In alcuni casi, tuttavia, non si è potuto eliminare direttamente delle inconsistenze che erano intrinseche nei dati. In tutti questi casi, si è dovuto ricorrere a dei processi di **riconciliazione** manuali o semiautomatici, descritti in dettaglio nel capitolo 4.

Per permettere di verificare in modo immediato la correttezza dei dati inseriti nell'ontologia, e per mostrare ad un utente informazioni di interesse, in questo contesto, si è sviluppata una applicazione interattiva che utilizza, oltre ai dati del grafo strade, anche i dati provenienti dalle altre fonti. In particolare, sono state create delle interrogazioni al repository che restituiscono dati relativi, ad esempio, a **Servizi** (comprensivi di servizi di **Parcheggi Auto**), **Fermate del Bus**, dati **AVM**, in modo geolocalizzato, attraverso collegamenti con il Grafo Strade.

In relazione al lavoro svolto, e non all'architettura globale presentata in precedenza, sono stati concepiti alcuni casi d'uso di interesse. un diagramma

di esempio dei principali casi d'uso previsti è mostrato in figura 2.2:



Figura 2.2. Modello dei principali casi d'uso previsti per la visualizzazione dei dati

Per l'utente, sono quindi previsti 3 casi d'uso principali:

- **Ricerca nel Comune**
- **Ricerca vicino Fermata Bus**
- **Ricerca vicino Punto su Mappa**

In tutti e tre i casi si hanno delle fasi di preselezione dell'oggetto di ricerca di base (**Selezione Provincia e Comune**, **Selezione Linea e Fermata**, **Selezione Punto su Mappa**). La funzionalità centrale della applicazione

ServiceMap è quella generica di **Ricerca Servizi**, che può essere estesa con la ricerca delle principali categorie di servizi, ad esempio **Ricerca Alloggi**, **Ricerca Educazione**, eccetera.

Nel primo caso, ovvero quello di **Ricerca nel Comune**, si intendono recuperare tutti i servizi delle categorie selezionate dall'utente, e le fermate degli autobus, presenti in un determinato comune. Nel caso d'uso **Ricerca vicino Fermata Bus**, si considerano come centro della ricerca le coordinate geografiche di una Fermata del Bus, selezionata dall'utente. A partire da tale centro, si ricercano, in un raggio massimo scelto dall'utente, tutti i servizi ed eventualmente altre fermate del bus di interesse. Infine, grazie all'ultimo caso (**Ricerca vicino Punto su Mappa**), si estende il secondo caso d'uso permettendo di impostare, come centro della ricerca, un punto qualsiasi della mappa (ad esempio, la posizione GPS attuale di un utente, oppure le coordinate di un servizio).

In aggiunta a queste ricerche principali, si sono previste delle funzionalità accessorie, quali ad esempio la **Ricerca tra quanto passa un autobus** che fornisce le previsioni per i prossimi transiti di varie linee di autobus su una determinata fermata (spiegato in dettaglio in sezione 3.1.2.5), ed il **Controllo Posti liberi in Parcheggio**, che fornisce lo stato di occupazione attuale di un determinato parcheggio.

Capitolo 3

Progettazione

La fase di progettazione del lavoro ha permesso di definire in dettaglio l'intero sistema concepito. In figura 3.1, si può visualizzare lo schema previsto per il progetto.

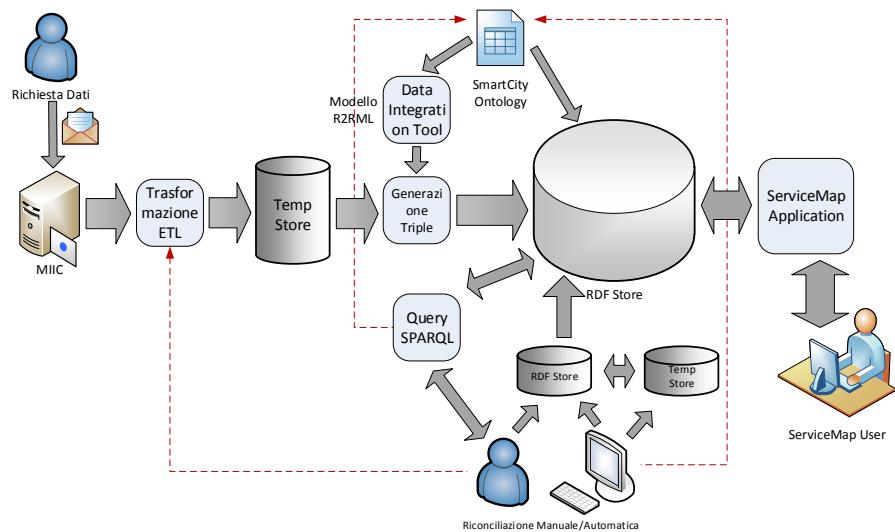


Figura 3.1. Grafico di rappresentazione dell'intero sistema progettato

Nei prossimi paragrafi, vengono descritte le principali fasi della progettazione, ad eccezione della progettazione della ServiceMap Application, ovvero della mappa interattiva di visualizzazione dei dati, che viene descritta nel capitolo 5.1.

3.1 Costruzione dell'Ontologia SmartCity Ontology

3.1.1 Stato dell'arte

L'architettura che si intende costruire richiede che le informazioni siano mappate su una ontologia. Nelle prime fasi del progetto, perciò, è stato necessario effettuare una ricerca approfondita che permetesse di capire se esistessero altre ontologie, rese disponibili pubblicamente, che fossero compatibili con i dati di interesse. Un'ontologia generale che coprisse tutte le aree di dominio del progetto non è stata trovata, pertanto sono state ricercate anche altre ontologie che potessero essere utilizzate per specifiche parti del lavoro.

Al fine di garantire l'interoperabilità dei dati inseriti in una futura ottica di Linked Data, sono state utilizzate in diverse parti dell'ontologia classi e DataProperties facenti parte di vocabolari e ontologie note. Di seguito, sono elencate le principali ontologie di uso comune che sono state utilizzate all'interno del progetto. La descrizione dettagliata dell'ontologia SmartCity è presente nel prossimo paragrafo.

Una delle ontologie più utilizzate è **FOAF** (**F**riend **o**f **a** **F**riend)¹. Tramite FOAF è possibile descrivere le persone, i legami che le collegano reciprocamente e con altri oggetti. Così, ad esempio, nell'ontologia **SmartCity Ontology**, la classe PA (Public Administration) viene definita come sottoclasse della Classe *foaf:Organization*, che nel vocabolario viene descritta nel seguente modo: “*the Organization class represents a kind of Agent corresponding to social institutions such as companies, societies etc.*”. Inoltre, all'interno dell'ontologia, spesso viene utilizzata la DataProperty *foaf:name*

¹FOAF Project Vocabulary - <http://xmlns.com/foaf/spec/>

che rappresenta, di fatto, la soluzione più utilizzata per i nomi dei vari oggetti (siano essi persone, province o fermate dell'autobus).

Un altro vocabolario di riferimento utilizzato è il **Dublin Core**², il quale è costituito da un piccolo insieme di termini generali, utili per la descrizione di risorse (sia reali che sul web). Nel progetto, questo vocabolario è stato particolarmente utilizzato quando ad esempio era necessario descrivere la fonte di un particolare dato, utilizzando la DataProperty *dc:source*, oppure le date di inserimento o di modifica (attraverso le proprietà *dc:created* e *dc:date*).

Per la gestione di informazioni georeferenziate si è scelto di usare il vocabolario **Geo WGS84**³. Questo semplice vocabolario mette a disposizione principalmente le DataProperties *geo:lat* e *geo:long* per la memorizzazione delle coordinate di un qualsiasi oggetto georeferenziato. L'utilizzo di questo vocabolario risulta particolarmente utile in concomitanza con il software OWLIM utilizzato in questo lavoro, grazie al fatto che esso mette a disposizione strumenti per la generazione di indici geospaziali proprio a partire dalle DataProperties appena citate.

Per quanto riguarda la parte di Servizi e Punti di Interesse, ai principali oggetti di questo tipo è stata associata una vCard, ovvero una serie di DataProperties proprie dall'ontologia **vCard**⁴. Essa permette la gestione di informazioni relative a persone e ad attività, con proprietà quali ad esempio i campi *vcard:street-address* e *vcard:locality* relativi all'indirizzo di esercizio di un servizio, oppure il campo *vcard:organization-name*, che ne contiene il nome o la ragione sociale.

Sempre riferita alla classe Service dell'ontologia **SmartCity Ontology**, si ha la proprietà *skos:note* dell'ontologia **SKOS (Simple Knowledge Orga-**

²Dublin Core Metadata Initiative - <http://dublincore.org/>

³Basic Geo (WGS84 lat/long) Vocabulary - <http://www.w3.org/2003/01/geo/>

⁴vCard Ontology - <http://www.w3.org/TR/vcard-rdf/>

nization System)⁵. SKOS è una famiglia di linguaggi creata per il Semantic Web al fine di rappresentare classificazioni, tassonomie. Il campo *skos:note* è stato utilizzato per memorizzare informazioni aggiuntive dei punti di interesse, quali ad esempio orari di apertura, servizi offerti, dettagli storici, eccetera.

Infine, per rendere omogeneo il salvataggio di informazioni di carattere temporale è stata utilizzata l'ontologia **Time**⁶. Tutte le entità che hanno una data associata (ad esempio i report meteorologici oppure le previsioni sui transiti degli autobus) sono stati collegati alla classe *time:Instant* per una corretta gestione degli intervalli temporali.

In aggiunta alle ontologie comuni appena citate, si è deciso di inserire nel progetto anche una parte dell'ontologia **OTN (Ontology of Transportation Networks)**[3]. Tale ontologia, utilizzabile in ambito Smart City, permette di modellare una rete di trasporti cittadina. Al suo interno infatti, sono definite varie macrosezioni contenenti, ad esempio, classi per la gestione del grafo stradale, del trasporto pubblico locale, piuttosto che risorse per la mappatura di dati geometrici generici (quali nodi, archi o superfici).

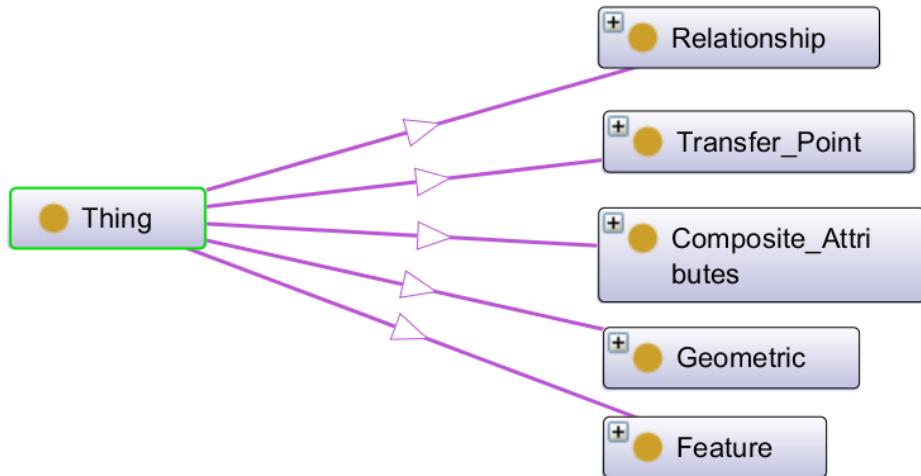


Figura 3.2. Principali macroclassi dell'ontologia OTN

⁵SKOS Simple Knowledge Organization System - <http://www.w3.org/2004/02/skos/>

⁶Time Ontology in OWL - <http://www.w3.org/TR/owl-time/>

In figura 3.2 è mostrata una schermata in cui è possibile vedere le principali macroclassi dell'ontologia OTN:

- **Feature:** comprende le principali classi di interesse quali ad esempio *Road_And_Ferry_Feature*, *Railways*, *Service*, *Public_Transport*, eccetera.
- **Transfer_Point:** permette di definire punti di trasferimento come *Junction*, *Parking*, *Bus_Station*, eccetera.
- **Relationship:** contiene classi di interesse come ad esempio *Manoeuvre*.
- **Composite_Attribute:** comprende classi per la memorizzazione di informazioni composite quali ad esempio *House_Number_Range*, *TimeTable*, eccetera.
- **Geometric:** prevede le classi generiche *Edge*, *Node* e *Face* per la definizione di entità spaziali monodimensionali, bidimensionali o di segmenti.

Facendo ricorso alle linee guida per la creazione di Linked Open Data (si veda l'Appendice A.4), all'interno dell'ontologia **SmartCity Ontology**, sono quindi state definite esplicitamente, dove semanticamente possibile, delle relazioni tra le classi create e quelle dell'ontologia OTN. Per fare un esempio, la classe *Sii-Mobility:BusStop*, che rappresenta una fermata dell'autobus, è stata definita come sottoclasse della *otn:StopPoint*.

Altre due ontologie analizzate che tuttavia, al momento della stesura di questa tesi, non sono state utilizzate, sono la **Geonames**⁷ e la **Location**⁸.

Geonames principalmente è un database geografico che contiene decine di milioni di nomi geografici recuperabili tramite web services. Queste entità contengono molteplici informazioni relative alla loro categoria, alle coordinate geografiche, eccetera. Ognuna di queste entità è rappresentata secondo

⁷Geonames Ontology - <http://www.geonames.org/ontology/documentation.html>

⁸Core Location Vocabulary - https://joinup.ec.europa.eu/asset/core_location/description

il paradigma del Semantic Web, quindi attraverso una URI univoca. L'ontologia Geonames permette di mappare tutte le informazioni del database attraverso il formato OWL. Geonames, con buona probabilità, verrà utilizzato in futuro per collegare i dati del progetto di questa tesi con risorse esterne, nella consueta ottica di Linked Data.

Tramite il vocabolario Location è possibile definire un generico luogo in diversi modi: tramite indirizzo, tramite un nome geografico oppure genericamente come una entità geometrica.

3.1.2 Ontologia SmartCity Ontology

Dopo aver analizzato lo stato dell'arte delle ontologie attualmente presenti in rete, all'interno del laboratorio DISIT è stato deciso di sviluppare una ontologia ex novo per il progetto. Grazie alla fase di analisi effettuata in precedenza, questo lavoro ha contribuito in parte alla sua progettazione, senza tuttavia prendere parte attivamente alla realizzazione vera e propria.

In figura 3.3 è mostrata la rappresentazione grafica dell'ontologia **Smart-City Ontology**, sviluppata dal laboratorio DISIT, allo stato attuale.

Costruzione dell'Ontologia SmartCity Ontology

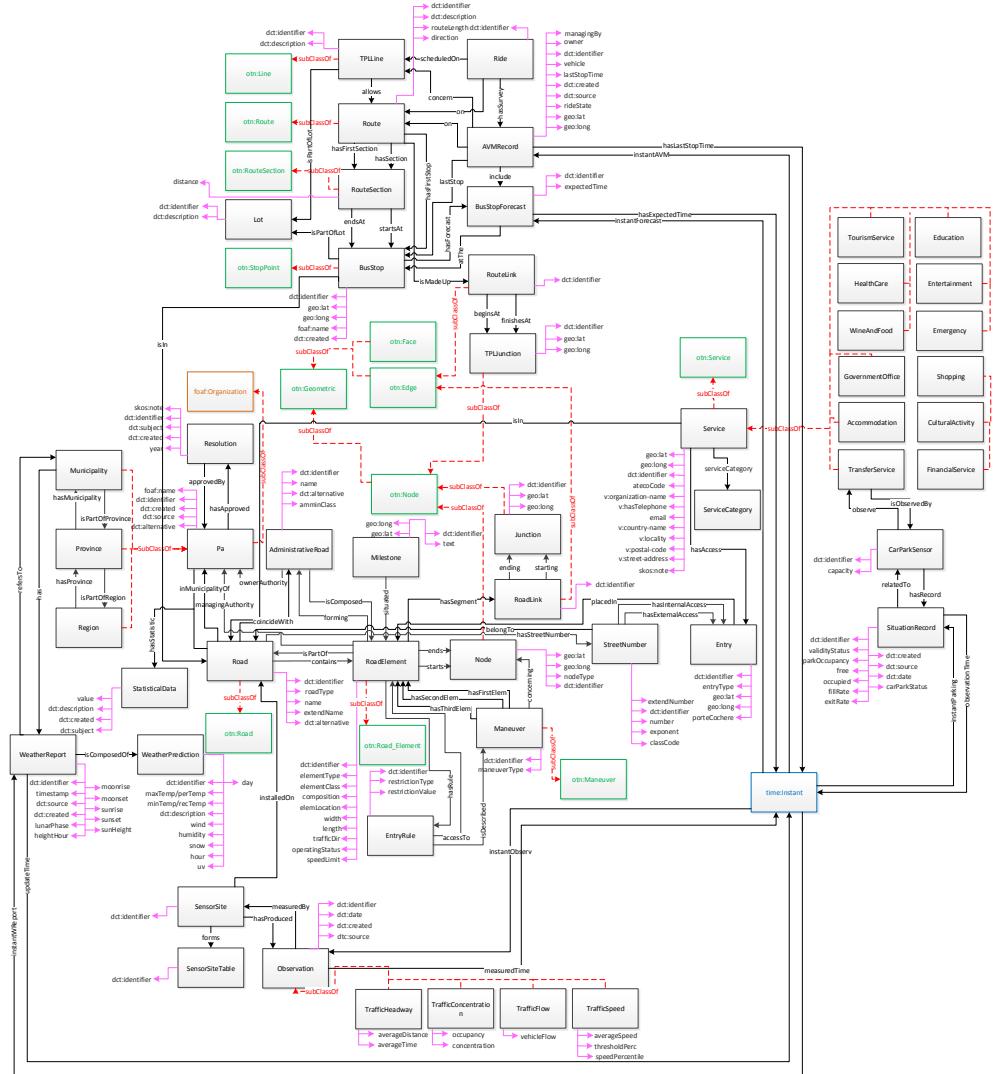


Figura 3.3. Rappresentazione grafica dell'intera ontologia SmartCity Ontology

Come si vede dalla figura, poiché il progetto è molto ampio ed i dati da trattare sono particolarmente disomogenei, l'ontologia ottenuta risulta molto vasta. Per questo motivo, di seguito l'ontologia viene suddivisa nelle principali macro-sezioni che la compongono, e per ciascuna verranno fornite informazioni dettagliate. Al momento della stesura di questa tesi, le 6 macro-sezioni contenute all'interno dell'ontologia **DISIT SmartCity Ontology** sono le seguenti:

- **Grafo Stradale**
- **Pubblica Amministrazione**
- **Trasporto Pubblico Locale**
- **Servizi e Attività**
- **Sensoristica**
- **Temporale**

3.1.2.1 Grafo Stradale

La sezione principale dell'ontologia **SmartCity Ontology** è quella del **Grafo Stradale**. Su di essa sono mappate tutte le informazioni provenienti dall'Osservatorio Regionale per la Mobilità ed i Trasporti della Regione Toscana che rappresentano la rete viaria della Toscana. In figura 3.4 è mostrata la rappresentazione grafica della sezione Grafo Stradale dell'ontologia **SmartCity Ontology**.

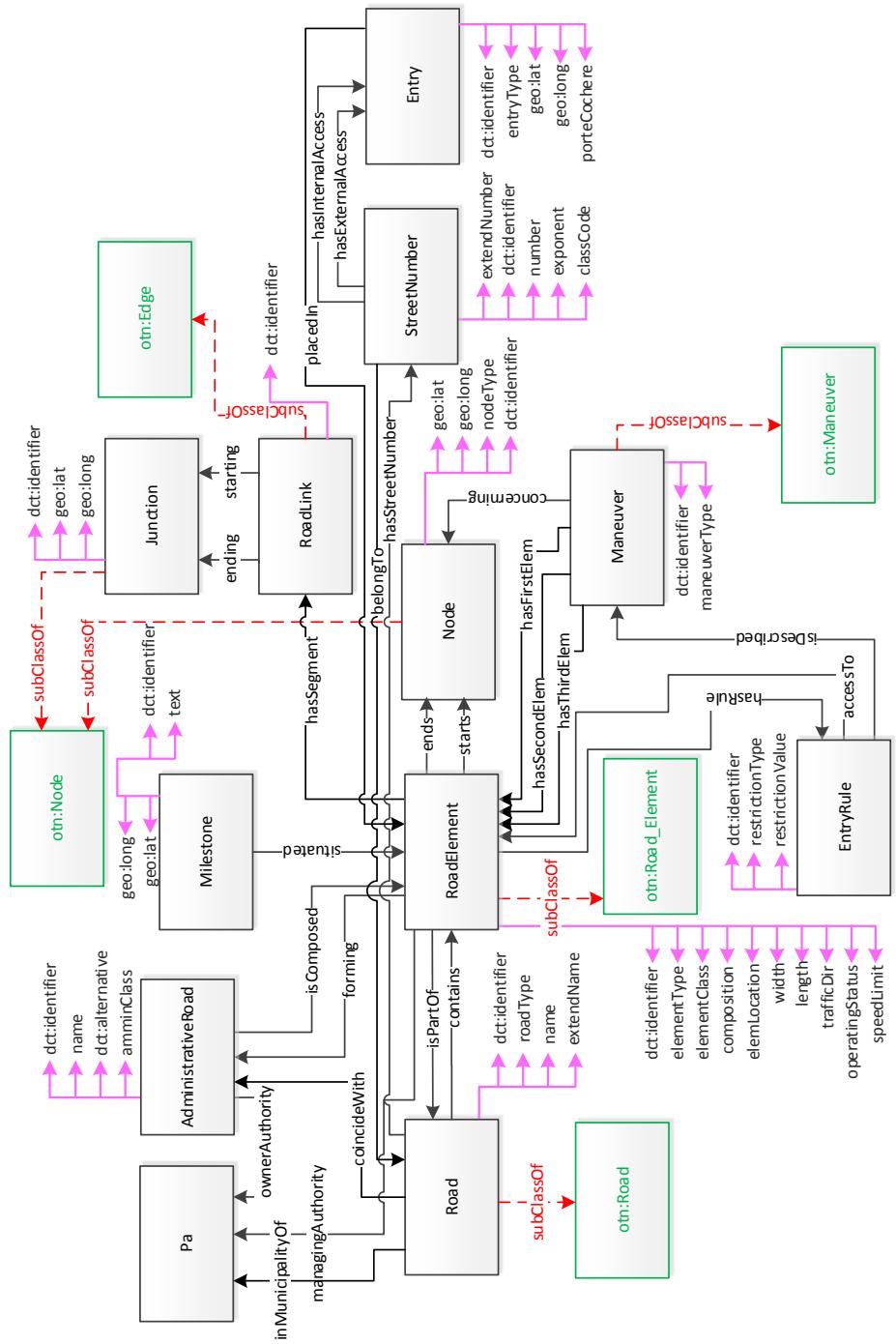


Figura 3.4. Rappresentazione grafica della sezione Grafo Stradale dell'ontologia SmartCity Ontology

Dalla figura si nota che il grado di correlazione tra le varie classi è molto elevato. Di seguito vengono riportate le principali classi, già accennate nella sezione 2.1.1.

La classe principale di questa sezione è senza dubbio la classe **Road**. Essa rappresenta, genericamente, una strada o, seguendo la terminologia dell'Osservatorio dei Trasporti, un *Toponimo*. Riprendendo la definizione fornita nel paragrafo 2.1.1, si ha che un *Toponimo* corrisponde ad *un insieme di Elementi Stradali di uno stesso Comune aggregati secondo il nome assegnato dal Comune stesso*. La classe **Road** dispone di numerose DataProperties:

- *dc:identifier*: contiene l'identificativo univoco fornito dall'Osservatorio trasporti e che viene codificato nella URI dell'oggetto. Ad esempio *RT04801700001TO*.
- *roadType*: è la tipologia del toponimo. In ambito urbanistico viene definito DUG (Denominatore Urbanistica Generica) della strada. Quindi può assumere una serie di valori quali “*VIA*”, “*PIAZZA*”, “*CORSO*”, eccetera.
- *name*: contiene il nome della strada senza il corrispondente DUG. Ad esempio “*DI SANTA MARTA*”.
- *extendName*: è il risultato della concatenazione tra DUG e name, quindi è il nome completo della strada. Ad esempio “*VIA DI SANTA MARTA*”.
- *dc:alternative*: la proprietà alternative viene utilizzata per memorizzare nomi alternativi della strada. Questa proprietà risulta utile al momento della riconciliazione dei toponimi, come spiegato nel paragrafo 4.1. Seguendo il solito esempio un possibile *dc:alternative* potrebbe essere “*VIA DI S.MARTA*” oppure “*VIA SANTA MARTA*”.

Per i motivi spiegati nel paragrafo precedente 3.1.1, la classe **Road** è definita come sottoclasse della **otn:Road**. La classe **Road** è collegata mediante ObjectProperties a molte altre classi: alla classe **RoadElement** attraverso

la coppia di proprietà inverse *contains-isPartOf*, alla classe **StreetNumber** mediante la coppia *hasStreetNumber-belongTo*, alla classe **PA** (Public Administration) dalla relazione *inMunicipalityOf* ed infine alla **AdministrativeRoad** grazie alla ObjectProperty *coincideWith*. Tramite la coppia *contains-isPartOf* si vuole indicare che una **Road** contiene molteplici **RoadElement**, mentre ogni **RoadElement** fa parte di una **Road**. Con le proprietà *hasStreetNumber-belongTo* si intende che un oggetto **StreetNumber** fa parte di una determinata **Road** e che di conseguenza una **Road** può avere molteplici **StreetNumber**. Infine, grazie alla proprietà *inMunicipalityOf*, si riesce a collegare una strada con il comune di appartenenza. Una spiegazione più approfondita della differenza tra **Road** e **AdministrativeRoad** viene fornita più avanti nel paragrafo assieme ai dettagli della proprietà *coincideWith* che le lega.

Come detto, una **Road** può contenere molteplici **RoadElement** o Elementi Stradali. Gli Elementi Stradali sono anch'essi fondamentali nella definizione del grafo stradale. Un Elemento Stradale, è per definizione dell'Osservatorio Trasporti, “*un'entità lineare delimitata da due Giunzioni, individuato da un insieme ordinato di punti. Esso rappresenta in genere, l'asse di un tratto di strada a singola carreggiata*”. Alla classe **RoadElement** sono associati molti attributi, tra questi possiamo trovare *dc:identifier* che ne rappresenta l'identificativo che va a formare l'URI univoca dell'oggetto (ad esempio `<http://www.disit.dinfo.unifi.it/SiiMobility/RT04701610085ES>`) e molti attributi specifici per la classificazione di un particolare tratto di strada. Di seguito sono elencati tali attributi con alcuni esempi di valori che possono contenere:

- *elementType*: tipologia dell'elemento stradale. I possibili valori sono estratti dalla tabella di dominio **DOM_TIP_ELE.dbf** e contengono ad esempio “*di tronco carreggiata*”, “*di rotatoria*”, “*di area a traffico non strutturato*”, “*di parcheggio*”, “*pedonale*”, eccetera.
- *elementClass*: classificazione tecnico/funzionale. I possibili valori sono estratti dalla tabella di dominio **DOM_CLS_TCN.dbf** e con-

tengono ad esempio “*autostrada*”, “*extraurbana principale*”, “*urbana di scorrimento*”, “*locale/vicinale/privata ad uso privato*”, eccetera.

- *composition*: composizione dell'elemento stradale. Gli unici due valori possibili sono “*carreggiata unica*” e “*carreggiate separate*”.
- *elemLocation*: sede dell'elemento. I possibili valori sono estratti dalla tabella di dominio **DOM_COD_SED.dbf** e contengono ad esempio “*a raso*”, “*ponte*”, “*galleria*”, eccetera.
- *width*: classe di larghezza dell'elemento stradale. Vi sono solo tre diverse classificazioni: “*minore di 3,5 mt*”, “*tra 3,5 e 7,0 mt*” e “*maggiori di 7,0 mt*”, più un valore di default indicato quando non è nota la classe di larghezza
- *length*: Lunghezza dell'elemento espressa in metri.
- *trafficDir*: Direzione del traffico. Un elemento stradale può risultare: aperto in entrambe le direzioni, chiuso in entrambe le direzioni, oppure aperto in un solo senso di marcia. L'ultimo caso prevede due ulteriori ramificazioni: un tratto stradale infatti può essere aperto in direzione positiva (cioè dalla giunzione in relazione *starts* a quella con relazione *ends*) e negativa (viceversa). I valori che indicano questi possibili scenari sono i seguenti: “*tratto stradale aperto in entrambe le direzioni (default)*”, “*tratto stradale chiuso in entrambe le direzioni*”, “*tratto stradale aperto nella direzione positiva (da giunzione NOD_INI a giunzione NOD_FIN)*” e “*tratto stradale aperto nella direzione negativa (da giunzione NOD_FIN a giunzione NOD_INI)*”.
- *operatingStatus*: indica lo stato d'esercizio dell'elemento stradale. Sono possibili tre valori: “*in esercizio*”, “*in costruzione*” e “*in disuso*”.
- *speedLimit*: definisce il limite di velocità massima su quel tratto di strada, espresso in Km/h.

La classe **RoadElement** è collegata alla classe **Node** (Giunzione) mediante le ObjectProperty *starts* ed *ends*. Ogni elemento stradale, infatti, è

delimitato da due giunzioni, che ne rappresentano l'inizio e la fine. La classe **Node** possiede numerose DataProperties, in particolare:

- *dc:identifier*: identificativo univoco regionale della giunzione che costituisce l'URI della risorsa.
- *geo:lat*: latitudine della giunzione.
- *geo:long*: longitudine della giunzione.
- *nodeType*: Tipologia della giunzione. Una giunzione viene definita nel momento in cui tra due distinti elementi stradali cambia una certa proprietà. Pertanto, questo campo può assumere molteplici valori, quali ad esempio: “*cambio sede*”, “*terminale (inizio o fine elemento stradale)*”, “*cambio toponimo / titolarità / gestore*”, “*variazione stato di esercizio*”, eccetera.

Gli oggetti di tipo **Node**, sono collegati, oltre che alla già citata classe **RoadElement**, anche alla classe **Maneuver** mediante la proprietà di collegamento *concerning*. Inoltre, la classe **Node** è stata definita come *subClassOf* della omonima **otn:Node** definita dalla Ontology of Transportation Networks.

Un'altra classe molto importante del grafo strade risulta essere **StreetNumber** (Numero Civico), la quale rappresenta un indirizzo esatto, cioè un numero civico su una determinata strada. A dispetto di quello che si può pensare, un numero civico non presenta direttamente le componenti spaziali utili alla sua georeferenziazione. Questo perché, ad ogni **StreetNumber**, viene associato almeno un oggetto di tipo **Entry** (Accesso), che rappresenta l'elemento puntuale che identifica sul territorio l'accesso ad uno specifico luogo di residenza o attività. Le due classi sono collegate mediante due ObjectProperties diverse: *hasExternalAccess* e *hasInternalAccess*, a seconda che quel numero civico abbia un accesso diretto o indiretto alla strada. La proprietà *hasExternalAccess* viene sempre valorizzata, l'altra soltanto nel caso in cui l'accesso fisico all'edificio o al terreno non sia direttamente posizionato

sulla strada principale. Come detto, **StreetNumber** è collegato a **Road** tramite la coppia di proprietà inverse *hasStreetNumber*-*belongTo*.

Le DataProperties di *StreetNumber* sono le seguenti:

- *dc:identifier*: identificativo univoco del numero civico.
- *number*: parte numerica del numero civico. Ad esempio “247”.
- *exponent*: eventuale esponente alfanumerico. Ad esempio “A”.
- *extendNumber*: numero civico esteso. Formato dalla concatenazione di *number* ed *exponent*. Ad esempio “247/A”.
- *classCode*: tipologia del numero civico. Può assumere i seguenti valori: “Nero”, “Rosso” o “Privo Colore” a seconda che l’amministrazione locale preveda o meno la differenziazione tra civici residenziali e commerciali.

Si è appena descritta parzialmente la classe **Entry**. Di seguito vengono fornite le informazioni mancanti. A livello di collegamenti, oltre alle già citate connessioni *hasExternalAccess* e *hasInternalAccess* con **StreetNumber**, la classe **Entry** prevede altre due ObjectProperties fondamentali: *placedIn* che riconnega l’accesso direttamente all’elemento stradale in cui si trova, e *hasAccess* che invece è usata per collegare la macrosezione di **Grafo Stradale** con quella di **Servizi e Attività**. Su tale proprietà si basa la riconciliazione tra i servizi ed il grafo stradale, descritto in sezione 4.2.

Le DataProperties della classe **Entry** sono:

- *dc:identifier*: id regionale dell’accesso.
- *entryType*: Tipologia dell’accesso. Può assumere i seguenti valori: “Accesso esterno diretto”, “Accesso esterno indiretto” e “Accesso interno”.
- *geo:lat*: latitudine espressa in coordinate WGS84 dell’accesso.
- *geo:long*: longitudine espressa in coordinate WGS84 dell’accesso.
- *porteCochere*: indica se si tratta di un “Accesso carrabile” o di un “Accesso non carrabile”.

La classe **Maneuver**, rappresenta le manovre consentite o vietate. Essa è collegata con **Node**, tramite la proprietà *concerning* già descritta, con **RoadElement** e con **EntryRule** (Regola di Accesso). Il collegamento con **RoadElement** è realizzato tramite tre distinte ObjectProperties: *hasFirstElem*, *hasSecondElem* e *hasThirdElem*. Queste tre proprietà rappresentano i tre diversi Elementi Stradali sui quali si può sviluppare una manovra, oltre alla giunzione sulla quale è definita la manovra stessa che è identificata dall'oggetto **Node** associato. La connessione con **EntryRule** è implementata tramite la proprietà *isDescribed*. In ottica Linked Data, la classe **Maneuver** è collegata alla paritetica **otn:Maneuver** dell'ontologia OTN, mentre per quanto riguarda gli attributi della classe, sono presenti le DataProperties *dc:identifier*, che contiene l'identificativo univoco regionale della manovra, e *maneuverType* che ne descrive il tipo. *ManeuverType* può assumere valori come: “*Manovra obbligatoria*”, “*Manovra proibita*”, “*Biforcazione*”, eccetera.

Per quanto riguarda le informazioni mancanti relative alla classe **EntryRule**, si può dire che essa rappresenta genericamente limitazioni e permessi di accesso su determinati elementi stradali o su particolari manovre. Per questo motivo, essa è collegata a **RoadElement** tramite la coppia di proprietà inverse *hasRule - accessTo* e alla classe **Maneuver** tramite la ObjectProperty *isDescribed*. Di seguito si elencano le DataProperties della classe **EntryRule**:

- *dc:identifier*: identificativo univoco della regola di accesso.
- *restrictionType*: tipo della regola di accesso. Può assumere vari valori, tra i quali *Passaggio bloccato*, *Regola su manovra*, eccetera.
- *restrictionValue*: ulteriore caratterizzazione della regola di accesso. Varia a seconda del valore assunto da *restrictionType*, come descritto in dettaglio in sezione 3.2.1. I possibili valori assumibili da questa proprietà sono, ad esempio: “*Chiusa in direzione positiva*”, “*Blocco fisico sulla giunzione finale*”, “*Valore di default per manovra*”, eccetera.

L'ultima classe appartenente al grafo stradale è **Milestone** (Cippo Chilometrico). Essa rappresenta il valore del chilometraggio di una estesa amministrativa in un particolare punto georeferenziato. Dal punto di vista dell'ontologia, essa, la classe **Milestone**, non è collegata direttamente con **AdministrativeRoad**, ma univocamente con **RoadElement**, tramite la proprietà *situated*, e presenta le seguenti DataProperties:

- *dc:identifier*: identificativo del cippo utilizzato per definire la URI della risorsa.
- *text*: Valore della chilometrica nel punto di localizzazione del cippo.
- *geo:lat*: latitudine del cippo chilometrico.
- *geo:long*: longitudine del cippo chilometrico.

La classe PA, mostrata in figura 3.4, è descritta in sezione 3.1.2.2.

Inizialmente, a tutte le classi della macro-area Grafo Stradale, si erano associate le DataProperties *dc:source* e *dc:created*, per rappresentare, rispettivamente, la fonte dei singoli dati (ad esempio “*Osservatorio Trasporti Regione Toscana*” o “*MIIC*”) e la data di creazione delle triple corrispondenti. Poichè, tuttavia, i dati relativi al grafo stradale sono aggiornati con frequenza molto bassa (in media una volta ogni tre mesi), è stato reputato inutile ridondare le informazioni di fonte e data di creazione sulle singole entità facenti parte dell'ontologia. Più semplicemente, si è pensato di ricorrere all'uso dei **Context** (o **Named Graph**). In estrema sintesi, i **Named Graph** sono degli insiemi di triple RDF ai quali viene assegnato un URI univoco. In questo modo, si riescono ad accorpare un certo numero di triple all'interno di uno stesso contesto, fornendo loro specifici metadati quali appunto la provenienza dei dati e la data di creazione delle triple, senza appesantire il repository con informazioni ridondanti di scarso valore.

In sezione 2.1.1 sono state introdotte le due entità **Toponimo** ed **Estesa Amministrativa**, che corrispondono, nell'ontologia **SmartCity Ontology**, alle classi **Road** e **AdministrativeRoad**. Entrambe queste entità sono

formate da insiemi di elementi stradali. La differenza principale tra le due classi è il criterio di raggruppamento di questi elementi. In particolare, seguendo le definizioni fornite dall'Osservatorio, un Toponimo è *un insieme di elementi stradali aggregati all'interno di un singolo comune in base al nome assegnato dal comune stesso*, mentre una Estesa Amministrativa è *un insieme di elementi stradali aggregati secondo criteri amministrativi*. Il problema principale di questa suddivisione è la corretta rappresentazione dei dati all'interno dell'ontologia. Dai dati del grafo stradale si è riusciti ad estrapolare le seguenti relazioni di cardinalità tra Elemento Stradale, Toponimo ed Estesa Amministrativa:

- Ogni Elemento Stradale è collegato ad un unico Toponimo. In alcuni rari casi è possibile che un elemento stradale faccia parte di due differenti toponimi, in particolare nel caso di strade disposte al confine tra comuni. Questa situazione, tuttavia, non comporta problemi per l'ontologia.
- Ogni Elemento Stradale è associato ad un'unica Estesa Amministrativa.
- Il rapporto di cardinalità tra Toponimo ed Estesa amministrativa è di N:M. In particolare, è possibile che differenti elementi stradali associati ad un singolo toponimo risultino far parte di diverse estese, così come è possibile che più elementi stradali, collegati alla stessa estesa, siano riconducibili a Toponimi diversi. In figura 3.5 è mostrato un esempio di queste due ultime eventualità.

COD_ELE	COD_TOP	COD_REG	DEN_UFF_TOPO	DEN_UFF_ESTESA	COD_PRP	TIP_PRP
RT04801317685ES	RT04801307003TO	RT04801307003PA	FRAZIONE CELLE	SENZA FRAZIONE CELLE	048013	STRADA COMUNALE
RT04801317684ES	RT04801307003TO	RT04801307061PA	FRAZIONE CELLE	SENZA FRAZIONE CELLE	999999	STRADA PRIVATA
RT04801317561ES	RT04801307003TO	RT04800000028PA	FRAZIONE CELLE	S.P. DI SAGGINALE (N.41)	048	STRADA PROVINCIALE

Caso 1: TOPO unico associato a 3 diverse ESTESE_AMMINISTRATIVE

COD_ELE	COD_TOP	COD_REG	DEN_UFF_TOPO	DEN_UFF_ESTESA	COD_PRP	TIP_PRP
RT04801317001ES	RT04801307048TO	RT04800000028PA	FRAZIONE VILLA	S.P. DI SAGGINALE (N.41)	048	STRADA PROVINCIALE
RT04801317366ES	RT04801307025TO	RT04800000028PA	VIA CIRO FABBRONI	S.P. DI SAGGINALE (N.41)	048	STRADA PROVINCIALE
RT04801317487ES	RT04801307014TO	RT04800000028PA	PIAZZA TRIESTE	S.P. DI SAGGINALE (N.41)	048	STRADA PROVINCIALE
RT04801317561ES	RT04801307003TO	RT04800000028PA	FRAZIONE CELLE	S.P. DI SAGGINALE (N.41)	048	STRADA PROVINCIALE
RT04804917566ES	RT04804906799TO	RT04800000028PA	LOCALITA' BOVINO	S.P. DI SAGGINALE (N.41)	048	STRADA PROVINCIALE
RT04804917585ES	RT04804906798TO	RT04800000028PA	LOCALITA' PIMAGGIORE	S.P. DI SAGGINALE (N.41)	048	STRADA PROVINCIALE

Caso 2: ESTESA_AMMINISTRATIVA unica comprendente 6 diversi TOPONIMI

Figura 3.5. Esempio di differenti rapporti di cardinalità tra Estesa Amministrativa e Toponimo

Queste eventualità sono più frequenti nei comuni di periferia (gli esempi mostrati sono presi dal comune di “*DICOMANO*”), ove si hanno con maggiore frequenza strade rurali o non referenziate tramite DUG standard (“*Via*”, “*Piazza*”, ecc.) ma solo tramite “*Località*”, “*Frazione*” o “*Case Sparse*”. Il significato di questi casi particolari è il seguente: il caso 1 si riferisce alla situazione in cui una particolare strada è suddivisa in varie zone in base all’ente *proprietario*. Come si vede dalla figura, la strada contrassegnata dalla denominazione “*FRAZIONE CELLE*” presenta alcuni tratti stradali privati, altri di proprietà comunale, altri ancora di proprietà provinciale. Il secondo caso, invece, si presenta principalmente in presenza di strade provinciali, regionali o statali. Queste strade, come intuibile dalla tipologia, non sono confinate ad un singolo comune, ma ne possono attraversare numerosi. Quindi, essendo la denominazione del toponimo fornita dal singolo comune, all’attraversamento di comuni diversi si avranno denominazioni diverse. In più, all’interno di un centro abitato, la stessa via può assumere nomi (Toponimi) diversi in base alla suddivisione scelta dal singolo comune. Nell’esempio mostrato, la stessa “*STRADA PROVINCIALE N.41 DI SAGGINALE*”, all’interno del comune di “*DICOMANO*”, assume sei differenti denominazioni. In figura 3.6 è mostrato uno screenshot prelevato da Google Maps in cui si comprende visivamente il problema.

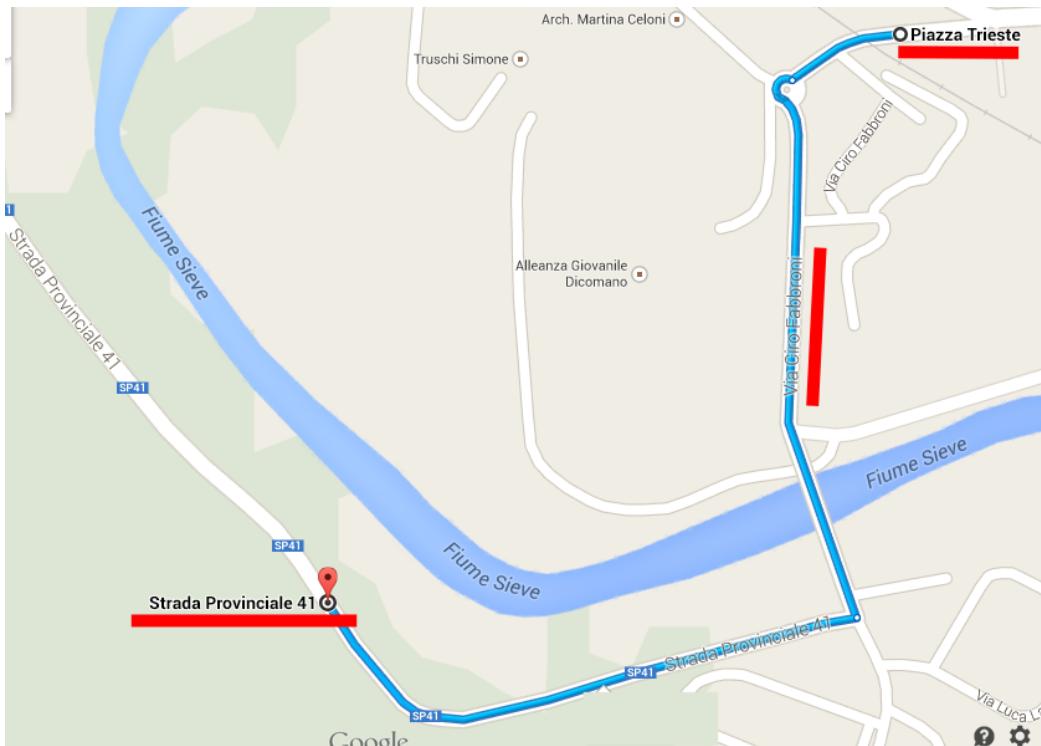


Figura 3.6. Esempio di Estesa Amministrativa formata da differenti Toponimi

In aggiunta al problema della cardinalità, è stato riscontrato anche un problema relativo al collegamento di queste entità con la classe **PA** (Pubblica Amministrazione). All'interno dei dati erano presenti molteplici collegamenti tra le classi:

- All'interno dei dati degli Elementi Stradali è stato riscontrato un campo contenente il codice della Pubblica Amministrazione che *gestisce* l'elemento Stradale.
- Nei dati relativi alle Estese Amministrative è presente un campo contenente la Pubblica Amministrazione *proprietaria* dell'estesa.
- Nel file dei Toponimi ogni record è collegato al particolare Comune di *appartenenza geografica*.

Dopo aver analizzato varie ipotesi, l'ontologia è stata delineata come mostrato in figura 3.4. Si è quindi collegato la classe **Road** alla **Administrati-**

veRoad, tramite la ObjectProperty *coincideWith*. La classe **AdministrativeRoad** e la **PA** sono state collegate mediante la proprietà *ownerAuthority*. Inoltre, tra la classe **Road** e la **PA** è stata inserita la proprietà *inMunicipalityOf*, mentre tra la **RoadElement** e la **PA** la proprietà *managingAuthority*. Il collegamento tra il singolo Elemento Stradale ed un Toponimo (ObjectProperty *isPartOf-contains*) è già stato descritto all'inizio di questa sezione. Invece, la relazione che lega **RoadElement** con **AdministrativeRoad** è definita mediante la coppia di proprietà inverse *forming-isComposed* che si interpreta nel seguente modo: una Estesa Amministrativa è formata da molti Elementi Stradali, e di conseguenza ogni Elemento Stradale forma, nel suo piccolo, l'Estesa Amministrativa.

In precedenza, è stato descritto che un elemento stradale è un'entità lineare delimitata da due Giunzioni. In questo modo si rappresenta un tratto di strada, arbitrariamente lungo, nel quale vengono mantenute le stesse proprietà descritte in sezione 3.1.2.1. Dalle coordinate delle Giunzioni, pertanto, si ottiene che un elemento stradale è una linea retta. Questa linea retta non segue necessariamente il naturale tracciato della strada, che al suo interno può avere un andamento curvilineo. Per questo motivo, sono stati analizzati più in dettaglio i dati provenienti dal grafo stradale e si è osservato che, all'interno degli Shapefile contenenti i dati degli elementi stradali, era presente un insieme ordinato di punti georeferenziati. Questi punti rappresentano in dettaglio le coordinate del tracciato che segue la strada lungo il territorio. In figura 3.7 è mostrata una rappresentazione grafica del problema.

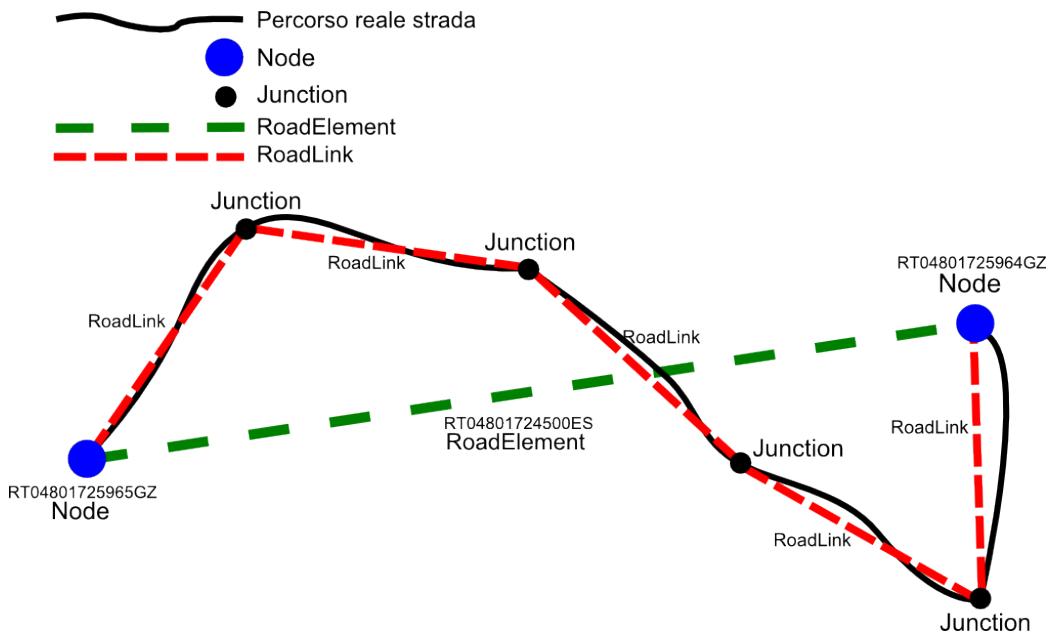


Figura 3.7. Rappresentazione grafica delle informazioni contenute all'interno degli ShapeFile Elementi Stradali

La linea nera piena rappresenta il reale percorso seguito da un elemento stradale tra due Giunzioni (indicate da cerchi blu). L'Elemento Stradale è rappresentato tramite una linea tratteggiata verde. Come si vede, l'interpolazione dell'Elemento Stradale a partire esclusivamente dalle coordinate delle Giunzioni, differisce molto dal reale andamento della strada. Per questo motivo, nell'ontologia **SmartCity Ontology** sono state inserite due classi aggiuntive: **RoadLink** e **Junction**. La classe **Junction** contiene le coordinate dell'insieme dei punti di un **RoadElement**. Il piccolo tratto di strada che collega due **Junction** successive è identificato da **RoadLink**, ed è collegato alla classe **RoadElement** grazie alla ObjectProperty *hasSegment*. Le due classi sono in relazione tramite le ObjectProperties *starting* ed *ending*. Le coordinate delle **Junction** vengono memorizzate tramite le consuete DataProperty *geo:lat* e *geo:long* dell'ontologia di uso comune Geo WGS84. L'unica altra proprietà associata a queste classi è *dc:identifier* che ne rappresenta l'identificativo univoco.

Come vedremo in dettaglio nella sezione 3.2.1, la generazione di questi dati è risultata inizialmente problematica, e successivamente, data l'enorme

mole di dati generati e la poca utilità degli stessi, si è preferito, almeno per il momento, non inserire queste informazioni all'interno del repository.

3.1.2.2 Pubblica Amministrazione

La macro-sezione Pubblica Amministrazione è composta dalle classi mostrate in figura 3.8

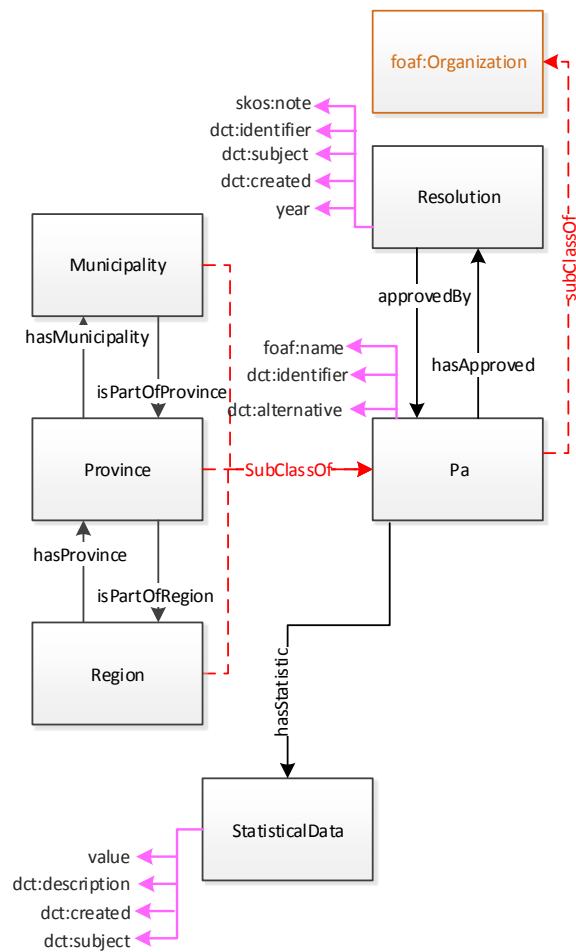


Figura 3.8. Macro-sezione Pubblica Amministrazione dell'ontologia SmartCity Ontology

La classe principale è **PA** (Pubblica Amministrazione), che è stata definita come sottoclasse della **foaf:Organization**. A sua volta, la Classe **PA** è superclasse di tre classi che rappresentano la gerarchia degli enti pubblici in Toscana: Regione (classe **Region**), Provincia (**Province**) e Comune (**Municipality**). Queste sottoclassi vengono automaticamente determinate dalle restrizioni sulle ObjectProperties che le collegano. In particolare, la classe **Region** è una sottoclasse di **PA** vincolata ad avere la proprietà *hasProvince*. In questo modo, si può definire una regione, esclusivamente come un'entità che possiede delle province. La stessa procedura si applica alla classe **Province** grazie alla proprietà *hasMunicipality*. Per ciascuna di queste ObjectProperties sono state definite le relazioni inverse: *hasProvince* è in coppia con *isPartOfRegion*, mentre *hasMunicipality* ha come proprietà inversa *isPartOfProvince*.

In aggiunta alle classi che rappresentano gli enti territoriali, sono state definite anche le classi **Resolution** e **StatisticalData**. Le istanze della prima classe rappresentano le delibere ed i provvedimenti approvati dalle varie Amministrazioni Locali e che sono reperibili, ad esempio, attraverso gli Open Data del Comune di Firenze. Questi dati sono collegati alla classe PA tramite la coppia di proprietà inverse *hasApproved* - *approvedBy*. La classe **Resolution** contiene alcune DataProperty di interesse. Tra queste, sono presenti la *dc:identifier*, la *dc:subject* e la *dc:created*, facenti parte dell'ontologia Dublin Core, che rappresentano rispettivamente, l'identificativo univoco, il titolo/argomento del provvedimento e la data di inserimento nel repository. Inoltre, tramite la proprietà *skos:note* è possibile memorizzare il contenuto della delibera e tramite la proprietà *year*, propria dell'ontologia, l'anno in cui è stata promulgata. Infine, l'ultima classe collegata alla PA è la **StatisticalData** che, come suggerisce il nome, contiene dati statistici relativi ai singoli comuni, alla regione o anche alle singole strade. Al suo interno, la classe **StatisticalData** contiene alcune DataProperties. In particolare, l'ontologia **SmartCity Ontology** definisce la proprietà *value* all'interno del quale può essere salvato l'effettivo dato statistico di interesse. Grazie alle proprietà *dc:description*, *dc:created* e *dc:subject* si riesce anche a contestualizzare tale dato. La classe **StatisticalData** è collegabile sia alla classe **PA** che alla

classe **Road** mediante la ObjectProperty *hasStatistic*.

3.1.2.3 Trasporto Pubblico Locale

La successiva macro-sezione dell'ontologia **SmartCity Ontology** contiene la struttura necessaria alla gestione dei dati del Trasporto Pubblico Locale. In figura 3.9 è mostrata la rappresentazione grafica di questa porzione di ontologia.

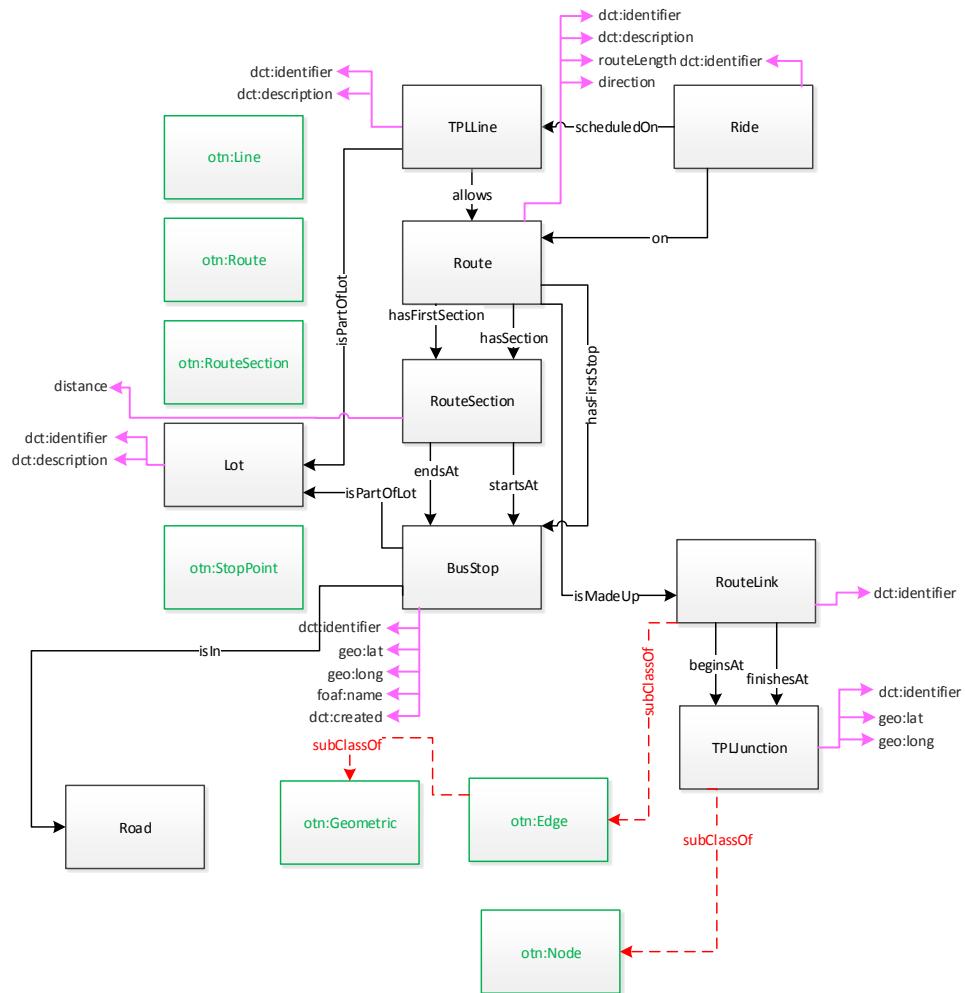
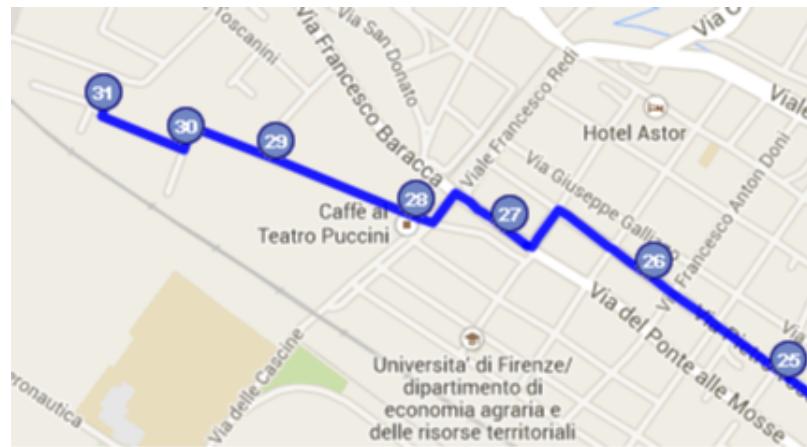


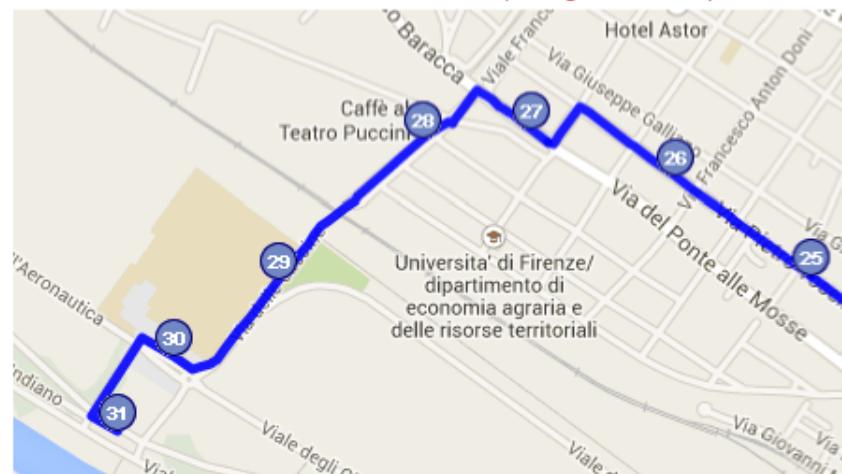
Figura 3.9. Macro-sezione Trasporto Pubblico Locale dell'ontologia SmartCity Ontology

La classe di partenza di questa sezione è la classe **Lot** (Lotto). Il servizio pubblico, in Toscana, è suddiviso in una serie di lotti in base alla zona e all'azienda fornitrice del servizio. Ciascun lotto è formato da un insieme di **TPLLine** (Linee Trasporto Pubblico). Questa relazione è espressa dalla proprietà *isPartOfLot* tra queste due classi. Per entrambe queste classi sono state impostate le DataProperties *dc:identifier* e *dc:description* grazie alle quali è possibile avere informazioni riguardanti il nome del lotto o della linea (e di conseguenza il loro identificativo univoco) ed una descrizione testuale, per una maggiore comprensione del dato. La classe **TPLLine** è stata definita come sottoclasse della **otn:Line**.

Una particolare linea può comprendere più entità di tipo **Route** (Percorso). Il caso più semplice è quello che prevede due percorsi per linea, un percorso di andata ed un percorso di ritorno. Nei dati analizzati, tuttavia, sono stati riscontrati anche casi più complessi, come ad esempio casi in cui una linea prevede più capolinea diversi, magari solo in orari particolari. In figura 3.10 è mostrato un esempio di questa situazione.



Linea ATAF: 17 - Percorso: 17B (Verga - Boito)



Linea ATAF: 17 - Percorso: 17C (Verga - Kennedy)

Figura 3.10. Esempio di Percorsi differenti su una stessa Linea - Immagini prelevate dal sito ATAF - Mappa di Google Maps

Le differenti Route sono collegate alla TPLLine tramite la DataProperty *allows*. Ogni Route è formata da un insieme di **RouteSection** (Sezioni di Percorso), ciascuna delle quali, a sua volta, è formata da una **BusStop** (Fermata) iniziale ed una finale. Tra la classe Route e le corrispondenti RouteSection sono state definite due diverse ObjectProperties, *hasFirstSection* e *hasSection*, utili ai fini di rappresentare, dal punto di vista cartografico, il percorso che segue l'autobus di una particolare linea. Tramite la proprietà *hasFirstSection* si individua il tratto di percorso iniziale. Tutte le altre sezioni sono collegate con la proprietà generica *hasSection*. L'ordine delle sezioni

del percorso è ottenibile attraverso delle interrogazioni ricorsive, che estraggono, in sequenza, le fermate dell'autobus di un determinato percorso e, di conseguenza, le relative sezioni. La **BusStop** iniziale di una **RouteSection** è reperibile grazie alla proprietà *startsAt* tra le due classi, mentre quella finale grazie alla proprietà *endsAt*. Per quanto riguarda le DataProperties, la classe **Route** prevede le consuete *dc:identifier* e *dc:description*. In aggiunta a queste, sono state definite anche le proprietà specifiche *routeLength* (lunghezza in metri del percorso) e *direction* (verso della Route: “*As*”, Ascendente o “*Di*”, Discendente). La classe **BusStop**, invece, oltre alla DataProperty *dc:identifier*, contiene la proprietà *foaf:name* che ne identifica il nome (ad esempio “*STAZIONE SCALETTE*”) e le coordinate *geo:lat* e *geo:long* per georeferenziare la fermata. Infine, in ottica Linked Data, le classi **Route**, **RouteSection** e **BusStop** sono state definite come *rdf:subClassOf* delle classi OTN **otn:Route**, **otn:RouteSection** e **otn:StopPoint**. Sempre per ottimizzare la determinazione del percorso è stata inserita nell’ontologia la relazione *hasFirstStop* tra **Route** e **BusStop** che indica direttamente la prima fermata di un percorso senza dover attraversare la classe **RouteSection**.

La logica del trasporto locale, oltre alle entità Linea, Percorso e Fermata, prevede un ulteriore concetto, quello di Corsa. Su ogni percorso, giornalmente, passano numerose corse, ciascuna delle quali è associata ad un determinato orario. Per questo motivo, l’ontologia **SmartCity Ontology** è stata corredata di una classe **Ride** collegata alla classe **TPLLine** ed alla classe **Route**. Nel primo caso il collegamento è espresso mediante la proprietà *scheduledOn*, mentre l’ObjectProperty che collega la Corsa al Percorso è denominata *on*. Ogni **Ride** è determinata univocamente dalla solita proprietà *dc:identifier*. La gestione degli orari delle corse è effettuata tramite i dati forniti dai sensori AVM, come descritto in sezione 3.1.2.5.

L’unica classe georeferenziata della macro-sezione Trasporto Pubblico Locale è la **BusStop**. I Percorsi sono stati definiti come sequenze di Fermate, perciò l’eventuale visualizzazione su una mappa di tale percorso, necessariamente, non potrà seguire il naturale andamento della strada. Si presenta quindi lo stesso problema evidenziato in sezione 3.1.2.1. Per questo motivo sono state predisposte due ulteriori classi per la gestione dettagliata e

geografica delle rotte del trasporto pubblico: **RouteLink** e **TPLJunction**. Tramite queste due classi si riesce a suddividere ogni percorso in numerosi piccoli segmenti georeferenziati, ciascuno delimitato da due giunzioni. Per farlo, si collega la classe **Route** alla **RouteLink** tramite la proprietà *isMadeUp* e ogni **RouteLink** alle corrispondenti **TPLJunction** di inizio e di fine tramite le ObjectProperties *beginsAt* e *finishesAt*. Entrambe le due ultime classi descritte hanno associata la proprietà *dc:identifier*. Inoltre, la classe **TPLJunction** prevede, al proprio interno, la memorizzazione delle coordinate geografiche tramite le DataProperties *geo:lat* e *geo:long*. Anche in questo caso, si riescono ad associare le classi create appositamente per l'ontologia, con le classi corrispondenti dell'ontologia OTN. In particolare, si associano **RouteLink** e **TPLJunction** alle classi principali della macrosezione otn:Geometric, ovvero **otn:Edge** e **otn:Node** tramite la proprietà *rdf:subClassOf*. Al momento della stesura di questa tesi, tuttavia, i dati relativi alle due classi appena descritte non sono ancora stati resi disponibili, pertanto saranno implementati con i prossimi sviluppi del progetto.

Infine, con lo scopo di collegare la macro-area TPL al Grafo Stradale, è stata aggiunta la ObjectProperty *isIn*, che lega oggetti di tipo **BusStop** con la classe **Road**. Questo legame non è stato immediatamente ottenuto dai dati provenienti dal MIIC, ma è stato necessario crearlo attraverso un processo di riconciliazione descritto in dettaglio al capitolo 4.3.

3.1.2.4 Servizi

A partire da un certo insieme di dataset provenienti dagli Open Data della Regione Toscana, è stato possibile recuperare informazioni relative a Servizi e Attività presenti sul territorio toscano. Per questa macrosezione è stata quindi predisposta la porzione di ontologia mostrata in figura 3.11.

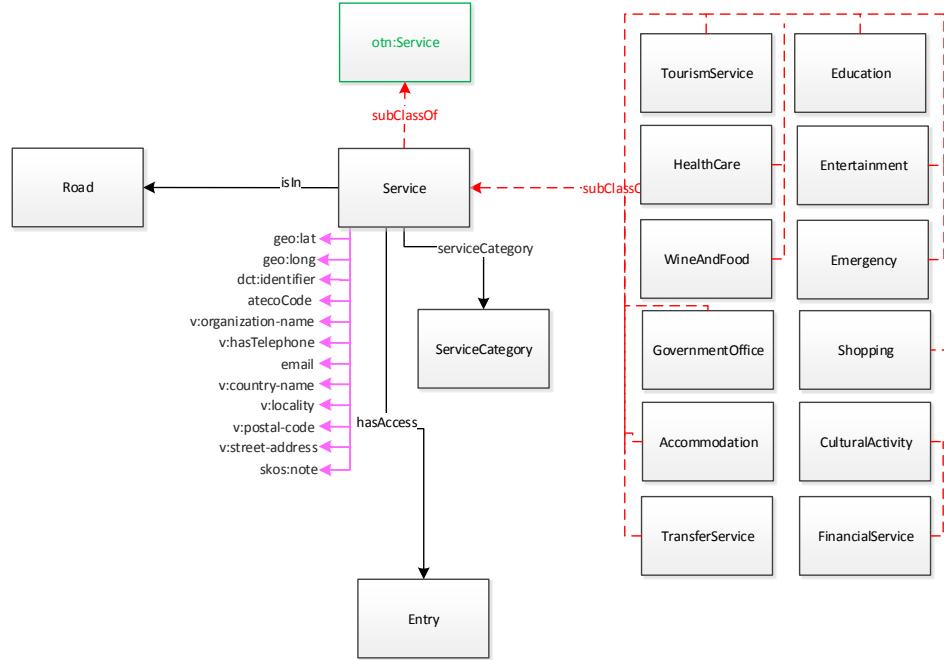


Figura 3.11. Macro-sezione Servizi e Attività dell'ontologia SmartCity Ontology

La classe principale di questa sezione è **Service**, definita come *subClassOf* della classe **otn:Service**. Alla classe generale Service sono state inoltre associate 12 sottoclassi, ciascuna rappresentante una certa tipologia di servizi. Questa suddivisione ricalca quella effettuata dalla Regione Toscana, che fornisce i dati di ciascuna sottoclasse in un file distinto. Di seguito vengono elencate tali sottoclassi:

- **Accommodation:** contiene i principali servizi di alloggio, quali ad esempio “*alberghi*”, “*ostelli*”, “*campeggi*”, eccetera.
- **CulturalActivity:** contiene le principali attività culturali, quali ad esempio “*musei*”, “*biblioteche*”, eccetera.
- **Education:** è composta da tutti quei servizi che fanno parte della sfera educativa-scolastica, come per esempio “*scuole elementari*”, “*licei*”, “*università*”, eccetera.

- **Emergency:** racchiude tutte le strutture delegate alla gestione delle emergenze pubbliche, quindi vi si possono trovare servizi come “*pronto soccorso*”, “*carabinieri*”, “*vigili del fuoco*”, eccetera.
- **Entertainment:** comprende tutte le attività di svago e divertimento quali, ad esempio, “*cinema*”, “*discoteche*”, “*piscine*”, eccetera.
- **FinancialService:** contiene tutti quei servizi riconducibili alla sfera economico-finanziaria, come per esempio “*banche*”, “*ATM*”, eccetera.
- **GovernmentOffice:** tutti gli uffici governativi, ad esempio “*agenzia delle entrate*”, “*motorizzazione civile*”, “*questura*”, eccetera.
- **Healthcare:** include tutte le strutture di assistenza sanitaria, quali ad esempio “*ambulatori*”, “*ospedali*”, “*cliniche private*”, eccetera.
- **Shopping:** racchiude tutti gli esercizi commerciali di vendita al dettaglio, come “*centri commerciali*”, “*outlet*”, “*negozi*”, eccetera.
- **TourismService:** è composta da tutte quelle attività che forniscono servizi turistici, “*visite guidate*”, “*tour operator*”, “*agenzie di viaggi*”, eccetera.
- **TransferService:** include tutti i servizi di trasferimento, quali ad esempio “*parcheggi auto*”, “*stazioni ferroviarie*”, “*aeroporti*”, eccetera.
- **WineAndFood:** comprende tutti le attività di ristorazione, come per esempio “*ristoranti*”, “*bar*”, “*gelaterie*”, eccetera.

Queste classi rappresentano le macroaree di interesse a cui può appartenere un servizio. In realtà, l'elenco di tutte le possibili tipologie di servizio è descritto come collezione di *Individuals* della classe **ServiceCategory**, collegata alla classe principale **Service** tramite omonima ObjectProperty *serviceCategory*. Questi *Individuals*, raggruppati a seconda della loro tipologia, formano le 12 principali categorie.

A livello di DataProperties, l'unica che ne prevede è la classe principale **Service**. In particolare, esse sono elencate di seguito, e sono rappresentate in gran parte da proprietà dell'ontologia vCard, spiegata in sezione 3.1.1:

- *dc:identifier*: identificativo univoco del servizio.
- *geo:lat*: latitudine in coordinate WGS84 del servizio.
- *geo:long*: longitudine in coordinate WGS84 del servizio.
- *atecoCode*: codice ATECO del servizio. Rappresentano la tipologia del servizio secondo la classificazione imposta dall'Agenzia delle Entrate italiana.
- *vcard:organization-name*: ragione sociale del servizio.
- *vcard:hasTelephone*: numero di telefono del servizio.
- *email*: indirizzo email dell'attività. Si è preferito definire una proprietà aggiuntiva piuttosto che utilizzare la proprietà di default dell'ontologia vCard per motivi di univocità. La proprietà di vCard richiedeva che l'indirizzo email fosse univoco. Se ciò non era verificato (come ad esempio è possibile in presenza di catene di negozi dello stesso marchio presenti in più località della Toscana), il sistema collegava erroneamente tutte le entità che avevano lo stesso indirizzo email, facendole risultare come un'unica entità.
- *vcard:country-name*: provincia di riferimento dell'attività.
- *vcard:locality*: comune di appartenenza del servizio. Questo campo, assieme al campo *street-address*, descritto poco più avanti, sarà fondamentale per il processo di riconciliazione dei servizi descritto in sezione 4.2.
- *vcard:postal-code*: CAP del servizio.
- *vcard:street-address*: indirizzo per esteso del servizio.
- *skos:note*: dettagli e note sull'attività.

La macroarea dei Servizi si collega al resto dell'ontologia attraverso alcune ObjectProperties molto importanti. Innanzitutto, si collega all'area della

sensoristica, in particolare dei Sensori di Parcheggio attraverso la coppia di proprietà inverse *observe* e *isObservedBy*, con la classe **CarParkSensor** (spiegata in dettaglio in sezione 3.1.2.5). Inoltre, ciascun servizio è collegato al grafo strade attraverso due importanti proprietà: **hasAccess** e **isIn**. La prima collega **Service** alla classe **Entry**, la seconda alla classe **Road**. Entrambe queste proprietà servono per collegare i servizi al grafo stradale. Nel primo caso, si ottiene anche una georeferenziazione molto fine, in quanto si associa il servizio ad un accesso, il quale rappresenta la posizione geospaziale di un indirizzo specifico (cioè composto di via e numero civico). Nel secondo caso, non si ottiene georeferenziazione, ma si riesce a collegare il servizio alla via di appartenenza. La procedura per generare queste associazioni è descritta in dettaglio in sezione 4.2.

3.1.2.5 Sensoristica

La macro-sezione relativa alla sensoristica real-time, in realtà, si compone di varie sezioni differenti. Le sezioni racchiuse all'interno dell'area Sensoristica sono le seguenti:

- **Previsioni Meteorologiche**
- **Parcheggi**
- **Sensori del Traffico**
- **AVM**

Di seguito sono descritte in dettaglio le singole sezioni.

Nella prima area, ovvero in quella delle **Previsioni Meteorologiche**, sono inseriti i dati forniti dal consorzio LAMMA attraverso web services, messi a disposizione dalla Regione Toscana. In figura 3.12, è mostrata la porzione di ontologia corrispondente.

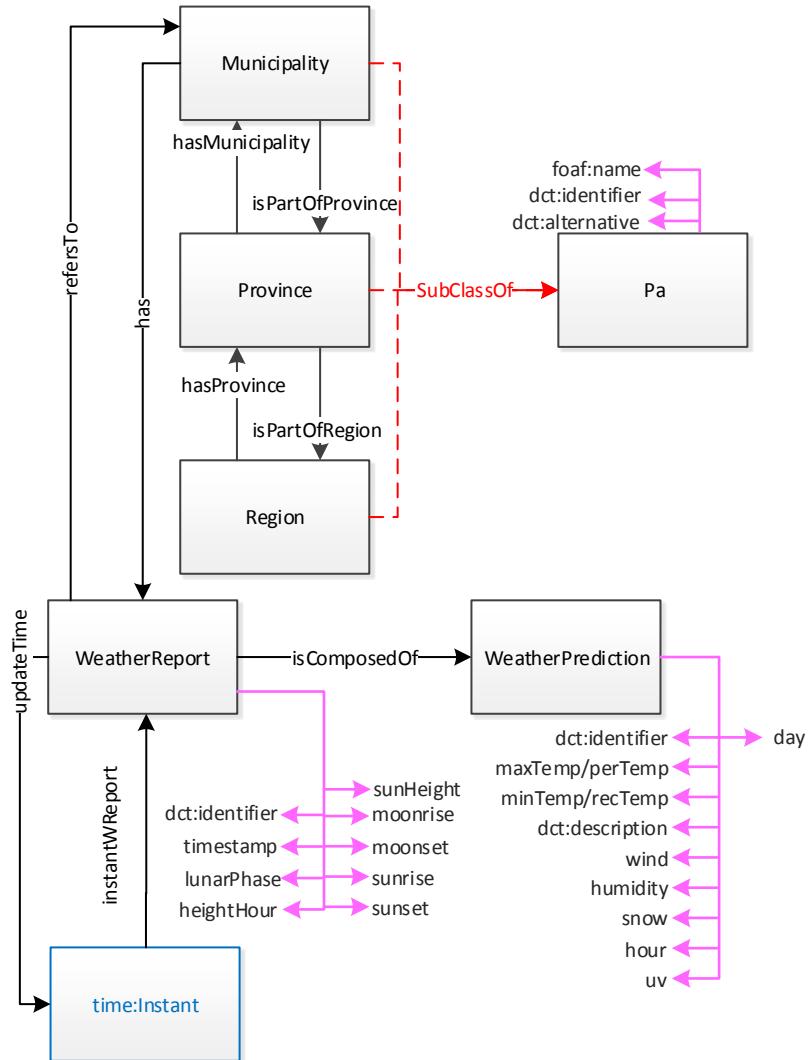


Figura 3.12. Macro-sezione Previsioni Meteorologiche dell'ontologia SmartCity Ontology

Le classi necessarie che sono state definite sono soltanto due, **WeatherReport** (Report Meteorologico) e **WeatherPrediction** (Previsione Meteorologica). L'unico collegamento con il resto dell'ontologia è fornito dalla coppia di ObjectProperties inverse *refersTo* - *has* che collega **WeatherReport** con la classe **Municipality** (già descritta nel paragrafo 3.1.2.2). In pratica, ogni oggetto di classe **WeatherReport** rappresenta un bollettino

meteo giornaliero, relativo ad un determinato comune in Toscana. Questi report vengono aggiornati con cadenza giornaliera, e contengono informazioni meteo relative al giorno corrente (quali ad esempio gli orari di alba e tramonto), oltre ad un insieme di entità di tipo Previsione Meteo per i giorni successivi. L'oggetto **WeatherReport** è collegato ai vari WeatherPrediction grazie alla ObjectProperty *isComposedOf*. Entrambe le classi contengono al proprio interno molteplici DataProperties, che sono elencate di seguito.

Proprietà della classe **WeatherReport**:

- *dc:identifier*: identificativo univoco del report.
- *dc:created*: data di generazione del file originario.
- *timestamp*: data di generazione del file in formato timestamp (numero di millisecondi a partire dal 1 Gennaio 1970).
- *lunarPhase*: attuale fase lunare (luna calante, crescente, nuova o piena).
- *sunrise*: orario di alba del sole.
- *sunset*: orario di tramonto del sole.
- *moonrise*: orario del sorgere della luna.
- *moonset*: orario di tramonto della luna.
- *sunHeight*: massima altezza del sole indicata in gradi.
- *heightHour*: orario in cui il sole si trova alla massima altezza.

Proprietà della classe **WeatherPrediction**:

- *dc:identifier*: identificativo univoco della previsione.
- *dc:description*: previsione letterale (può assumere svariati valori, tra i quali “sereno”, “coperto”, “pioggia e schiarite”, eccetera).
- *day*: giorno della settimana al quale si riferisce la previsione.

- *minTemp*: temperatura minima prevista espressa in gradi centigradi.
- *maxTemp*: temperatura massima prevista espressa in gradi centigradi.
- *perTemp*: temperatura percepita prevista espressa in gradi centigradi.
- *wind*: previsione del vento.
- *humidity*: umidità nell'aria prevista espressa in percentuale.
- *snow*: quota sul livello del mare alla quale può manifestarsi la neve, espressa in metri.
- *hour*: indica a quale fascia del giorno si riferisce la previsione (i valori possibili sono “*giorno*”/“*notte*”, “*mattina*”/“*pomeriggio*”/“*sera*”).
- *uv*: indice UV, ovvero il grado di intensità delle radiazioni ultraviolette previste sulla superficie del comune.

Anche la porzione di ontologia dedicata ai **Parcheggi** ed ai relativi sensori, è composta di due classi principali, collegate mediante una coppia di ObjectProperties inverse ad una classe di raccordo con il resto dell'ontologia. Le classi sono **CarParkSensor** e **SituationRecord**. In figura 3.13, è mostrata la porzione di ontologia corrispondente.

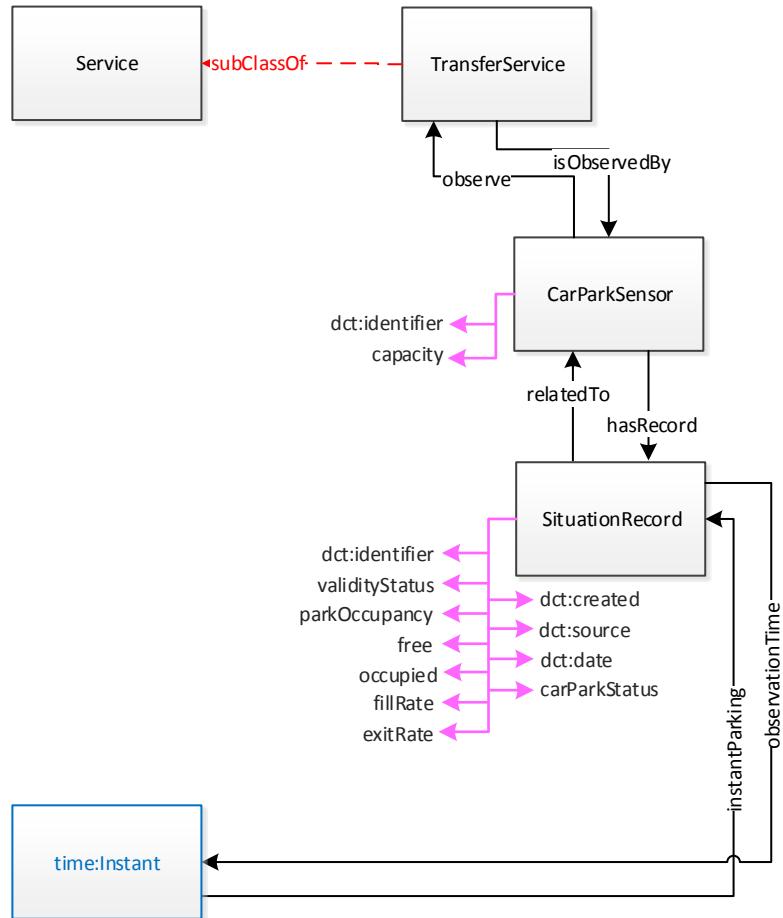


Figura 3.13. Macro-sezione Parcheggi dell'ontologia SmartCity Ontology

La classe di raccordo è la **TransferService**, che è una delle sottoclassi della sezione **Service**, descritta in sezione 3.1.2.4. Le ObjectProperties di collegamento sono *observe* e *isObservedBy*, mentre tra le due classi principali di questa sezione si ha la coppia di proprietà inverse *relatedTo* - *hasRecord*. Dal punto di vista semantico, grazie ai dati recuperati dal MIIC, si hanno a disposizione le informazioni relative ad un certo numero di parcheggi sparsi per la Toscana. Questi parcheggi, al momento della trasformazione in RDF, sono definiti come oggetti di classe *Service*. Più precisamente di classe **TransferService**, definita come *subClassOf* di **Service** (maggiori dettagli

in sezione 3.1.2.4). A questi oggetti vengono associati degli oggetti di tipo Sensore (**CarParkSensor**), i quali, periodicamente, producono dei report sullo stato del parcheggio, tramite entità di classe **SituationRecord**. Infatti, se all'interno della classe **CarParkSensor** sono previste soltanto le DataProperties *dc:identifier* e *capacity* (che indica la capacità massima di quel particolare parcheggio), la classe **SituationRecord** è corredata di numerosi attributi:

- *dc:identifier*: identificativo univoco della observation.
- *dc:created*: data di creazione del file messo a disposizione tramite web service.
- *dc:date*: data di osservazione, ovvero l'istante a cui si riferisce il report.
- *validityStatus*: indica la validità del report. Di default è impostato il valore “active”.
- *parkOccupancy*: valore in percentuale di posti occupati.
- *free*: numero di posti liberi all'interno del parcheggio.
- *occupied*: numero di posti occupati.
- *fillRate*: numero di veicoli in entrata per ora
- *exitRate*: numero di veicoli in uscita per ora.
- *carParkStatus*: valore letterale che descrive lo stato del parcheggio. Possibili valori sono “*carParkClosed*”, “*noParkingInformationAvailable*”, eccetera.

La terza sezione specifica di sensori Real-Time è quella relativa ai **Sensori del Traffico**. In figura 3.14, è mostrata la porzione di ontologia corrispondente.

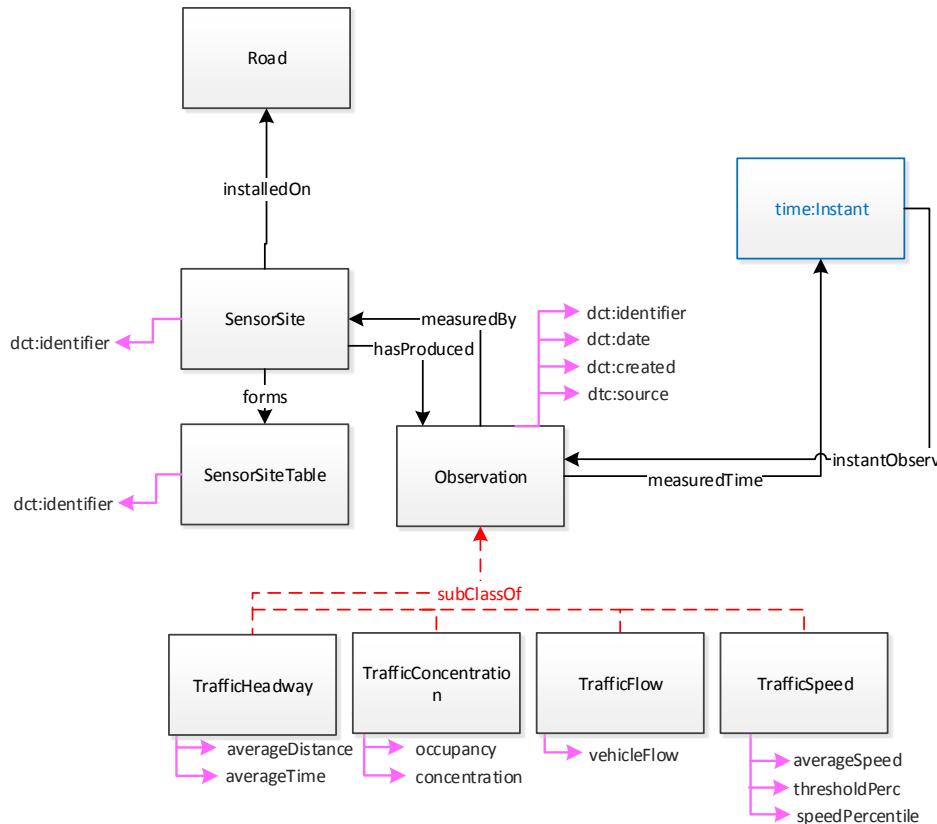


Figura 3.14. Macro-sezione Sensori dell'ontologia SmartCity Ontology

Anche in questo caso, si hanno a disposizione dati provenienti dal MIIC, recuperabili grazie all'uso di web services. La classe principale di questa sezione è la **SensorSite**, che rappresenta genericamente un sensore installato su una particolare strada. È quindi comprensibile la proprietà che collega tale classe al resto dell'ontologia, ovvero la ObjectProperty *installedOn* che collega **SensorSite** alla classe **Road**, già descritta nel paragrafo 3.1.2.1. Allo stato attuale, i fornitori dei dati relativi ai sensori non forniscono le coordinate di ubicazione dei singoli sensori, pertanto risulta, al momento, impossibile georeferenziare in modo più specifico le posizioni dei singoli sensori. Per questo motivo, il collegamento con una classe così ampia come **Road**, e non con classi più specifiche dal punto di vista geografico, non permette

una precisa collocazione del sensore sul territorio, ma indica esclusivamente il toponimo sul quale è installato. Poichè una larga parte di questi sensori sono installati su strade a lunga percorrenza (ad esempio la Strada di Grande Comunicazione Firenze-Pisa-Livorno o FI-PI-LI), molteplici oggetti di tipo **SensorSite**, disposti sulla stessa **Road**, vengono raggruppati in entità di tipo **SensorSiteTable**, mediante la proprietà *forms*. Sia **SensorSite** che **SensorSiteTable** sono dotati della consueta DataProperty *dc:identifier* per la loro identificazione univoca. I sensori, periodicamente, producono delle rilevazioni, ovvero delle **Observation**.

Le rilevazioni compiute dai sensori possono essere di più tipologie. In particolare, si possono identificare quattro diverse classi: **TrafficHeadway**, **TrafficConcentration**, **TrafficFlow** e **TrafficSpeed**. Ciascuna di esse è definita come *subClassOf* della classe **Observation** e produce un report con dati diversi. La classe **TrafficHeadway** monitora l'andamento del traffico e produce informazioni relative a *averageDistance* e *averageTime*, ovvero alla distanza media tra veicoli (espressa in metri) e all'intervallo di tempo medio tra un transito e il successivo (in secondi). La seconda classe (**TrafficConcentration**), invece, misura la concentrazione del traffico esprimendo DataProperties quali *occupancy* e *concentration* che indicano, rispettivamente, la percentuale di occupazione della strada ed il numero di veicoli per chilometro. **TrafficFlow** contiene informazioni sul flusso di veicoli, ovvero sul numero di transiti per ora, all'interno della proprietà *vehicleFlow*. Infine, **TrafficSpeed** prevede le DataProperties *averageSpeed*, *thresholdPerc* e *speedPercentile*. La prima proprietà indica la velocità media dei veicoli in transito (espressa in km/h), mentre le altre due proprietà non sono definite dal documento di riferimento [4], e non vengono attualmente valorizzate nei dati provenienti dal MIIC.

L'ultima sezione relativa alla sensoristica Real-Time è molto importante e comprende i dati forniti dal trasporto pubblico locale sotto forma di **AVM** o **Automated Vehicle Monitoring**. In figura 3.15, è mostrata la porzione di ontologia corrispondente.

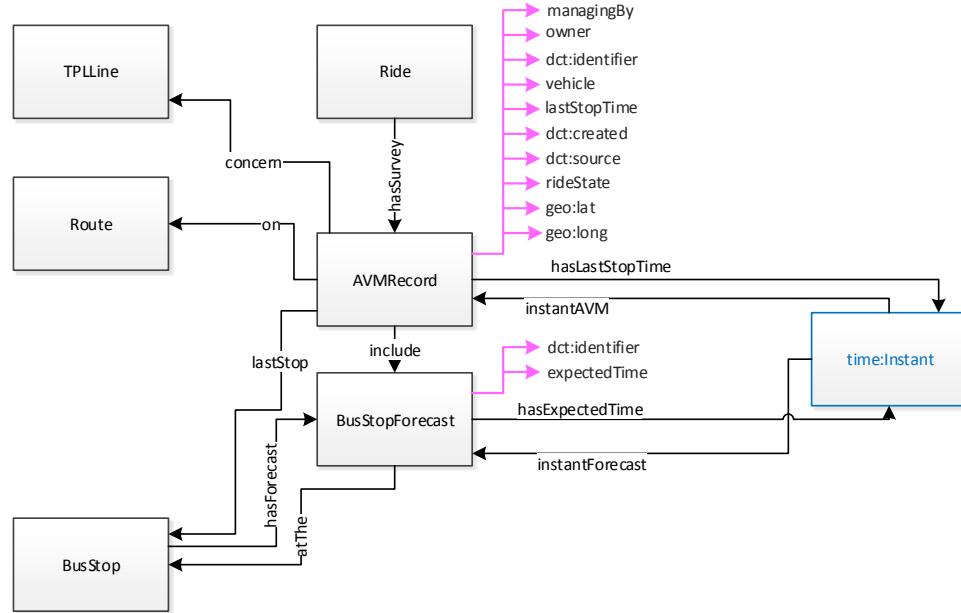


Figura 3.15. Macro-sezione AVM dell'ontologia SmartCity Ontology

L'AVM è un sistema telemetrico che serve per monitorare molteplici aspetti di un veicolo in movimento, quali ad esempio la posizione, la velocità, ed anche aspetti tecnici e di manutenzione, quali lo stato del motore ed il consumo di carburante. Il principale utilizzo di questa tecnologia risiede proprio nella gestione delle flotte di automezzi appartenenti a fornitori di trasporto pubblico locale. Grazie a dei dispositivi elettronici installati a bordo di autobus e pullman, è possibile avere, in tempo reale, tutte le informazioni di interesse relative ai mezzi di trasporto urbano ed extraurbano.

Le classi dell'ontologia relative agli AVM sono in realtà soltanto due: **AVMRecord** e **BusStopForecast**. Tuttavia, queste classi sono strettamente legate alle varie classi relative al Trasporto Pubblico Locale descritte nel paragrafo 3.1.2.3. Un oggetto di tipo AVMRecord contiene, al proprio interno, tutti i dati di interesse inviati dalla trasmittente AVM di una determinata corsa. In particolare, i dati utilizzati in questo progetto sono i seguenti:

- **dc:identifier**: identificativo univoco del report.

- *managingBy*: identificativo dell'affidatario del servizio di trasporto pubblico.
- *owner*: identificativo dell'azienda esercente il servizio di trasporto pubblico.
- *vehicle*: codice identificativo del mezzo da cui è stato emesso il report AVM.
- *dc:created*: data di creazione del report.
- *dc:source*: fonte del dato.
- *rideState*: indica lo stato della corsa. Può assumere tre diversi valori: “*In Anticipo*”, “*In Ritardo*” o “*In Orario*”.
- *geo:lat*: latitudine attuale del mezzo.
- *geo:long*: longitudine attuale del mezzo.

Ogni AVMRecord è sempre corredata di un insieme di oggetti di tipo **BusStopForecast** che rappresentano le previsioni sulle prossime fermate, in programma su quella corsa. Ciascuna **BusStopForecast** è collegata al proprio **AVMRecord** tramite la ObjectProperty *include*. La DataProperty principale di **BusStopForecast** è banalmente *dc:identifier*, che ne contiene l'identificativo univoco usato per generare l'URI dell'entità corrispondente, all'interno del triplestore.

Come detto, i collegamenti tra le classi AVM e quelle relative al trasporto pubblico locale sono, ovviamente, molto numerosi. In particolare sono presenti le seguenti ObjectProperties:

- *hasSurvey*: collega **Ride** a **AVMRecord**. Specifica a quale corsa si riferisce il Record.
- *concern*: collega **AVMRecord** a **TPLLine**. Indica a quale linea appartiene il mezzo che ha generato il report.

- *on*: collega **AVMRecord** a **Route**. Determina il percorso che sta seguendo l'automezzo.
- *lastStop*: collega **AVMRecord** a **BusStop**. Indica l'ultima fermata della corsa, attualmente non è usato perchè tale dato non è fornito dal MIIC.
- *atThe - hasForecast*: coppia di proprietà inverse che collegano **BusStop** a **BusStopForecast**. Specifica qual è la fermata alla quale si riferisce la previsione di transito.

In molti punti di questa macroarea, si hanno dei collegamenti con la sezione Temporale dell'ontologia. Tutti questi collegamenti sono descritti nel prossimo paragrafo.

3.1.2.6 Sezione Temporale

L'ultima macroarea definita dall'ontologia **SmartCity Ontology** è la sezione **Temporale**. Questa sezione, in realtà formata dalla singola classe **Instant**, prelevata dall'ontologia Time (già descritta nel paragrafo 3.1.1), punta all'inserimento di concetti legati al tempo (istanti di tempo e intervalli di tempo) all'interno dell'ontologia. In figura 3.16 è mostrata la classe **time:Instant** ed i suoi molteplici collegamenti con le classi dell'area Sensoristica.

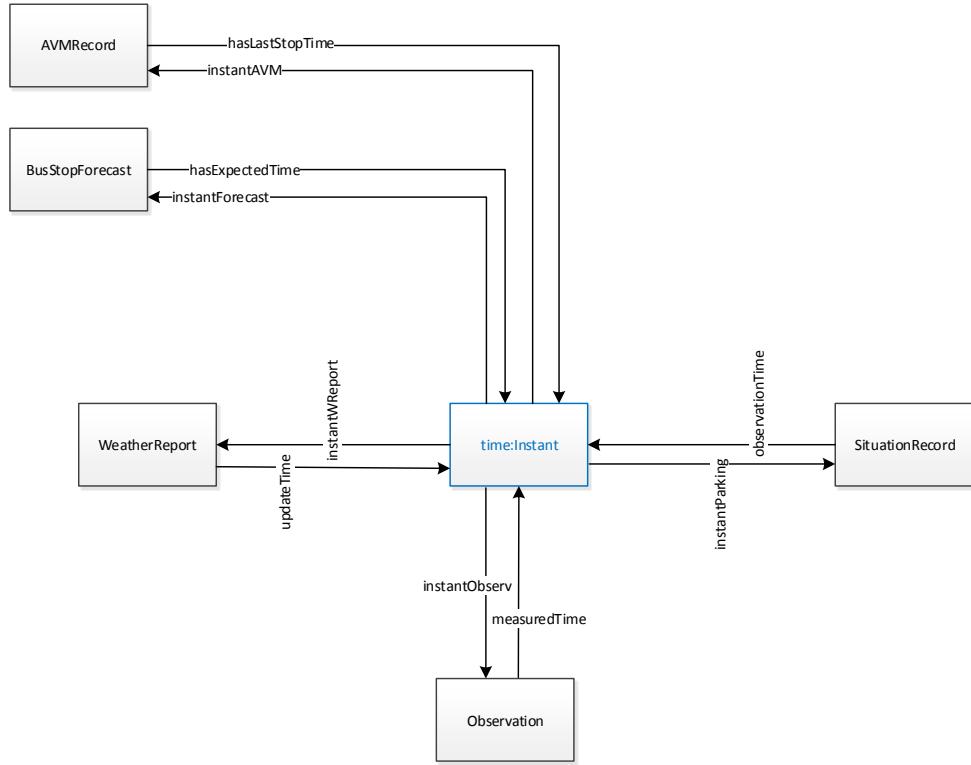


Figura 3.16. Macro-sezione Temporale dell'ontologia SmartCity Ontology

Attualmente questa sezione è stata utilizzata solo parzialmente nel progetto, ma in futuro sarà di fondamentale importanza per la gestione di istanti e intervalli di tempo in modo standard.

Come si vede dalla figura, la classe **time:Instant** è stata collegata a tutte quelle classi sulle quali è presente una informazione temporale rilevante. Ad esempio, ogni **BusStopForecast** (descritto nel paragrafo 3.1.2.5), avrà un istante di tempo associato alla previsione di transito di un autobus, su una particolare fermata. Nelle prime versioni dell'ontologia, tale dato veniva memorizzato direttamente attraverso una DataProperty dell'oggetto utilizzando il formato *xsd:date*⁹ suggerito dal consorzio W3C. Allo stato attuale, invece, ogni oggetto **BusStopForecast** è collegato ad una entità di tipo **time:Instant** contenente al proprio interno la proprietà *inXSDDate*.

⁹W3C Date and Time Formats - <http://www.w3.org/TR/NOTE-datetime>

teTime, concatenando la quale si riesce a formare l'URI della risorsa RDF corrispondente. I collegamenti della classe **time:Instant** sono i seguenti:

- **AVMRecord** (tramite la coppia di proprietà inverse *instantAVM-hasLastStopTime*): esplicita l'orario di ultima fermata relativo all'AVM Record.
- **BusStopForecast** (tramite la coppia di proprietà inverse *instantForecast-hasExpectedTime*): indica l'orario di arrivo previsto di un autobus su una determinata fermata.
- **SituationRecord** (tramite la coppia di proprietà inverse *instantParking-observationTime*): specifica l'orario di produzione del report del parcheggio.
- **Observation** (tramite la coppia di proprietà inverse *instantObservation-measuredTime*): esprime l'orario di registrazione del report del sensore.
- **WeatherReport** (tramite la coppia di proprietà inverse *instantWReport-updateTime*): definisce l'orario di ultimo aggiornamento del report meteo.

3.2 Fase di Ingestion

La progettazione della fase di Ingestion, ovvero l'intero processo di acquisizione, manipolazione e salvataggio dei dati, ha richiesto un lavoro molto accurato. Una rappresentazione grafica dell'intero processo di Ingestion è mostrato in figura 3.17.

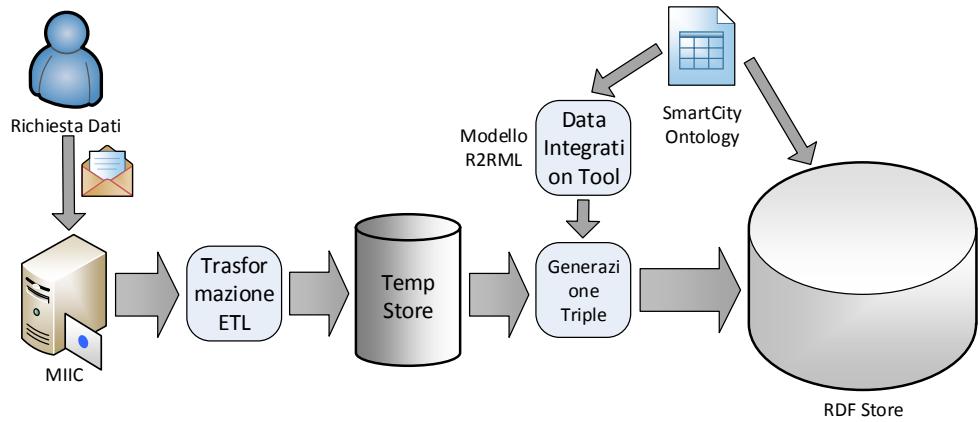


Figura 3.17. Rappresentazione grafica del processo di Ingestion dei dati

Il processo prevede, in input, i dati provenienti dall’Osservatorio Regionale per la Mobilità ed i Trasporti della Regione Toscana (**MIIC**). La **Richiesta Dati**, al momento, viene effettuata manualmente tramite un servizio web autenticato. I dati, suddivisi in vari archivi relativi alle differenti province, sono successivamente recuperati attraverso dei link inviati per email. Ogni archivio contiene 78 files contenenti tutte le informazioni relative al grafo stradale di quella provincia. Su questi dati, viene eseguita una **Trasformazione ETL (Extract, Transform, Load)**. Il risultato delle elaborazioni ETL viene inserito in un database MySQL temporaneo (**Temp Store**).

Nel paragrafo 3.1.2 è descritta in dettaglio la generazione della **Smart-City Ontology**. Grazie all’utilizzo di un software di tipo **Data Integration Tool**, si riesce, a partire dall’ontologia appena generata, a definire un **Modello R2RML** per i dati memorizzati sul MySQL Store. R2RML è un linguaggio che permette di mappare dati memorizzati su database relazionali, in triple RDF. Tramite questo modello si riesce ad effettuare la **Generazione delle triple RDF**, che vengono quindi inserite nell’**RDF Store** finale, assieme all’ontologia stessa.

Questo processo di Ingestion, una volta validato, è stato unito all’architettura presente allo stato attuale, ovvero quella mostrata in figura 3.18, la quale fa riferimento allo schema generale presentato in figura 2.1.

Fase di Ingestion

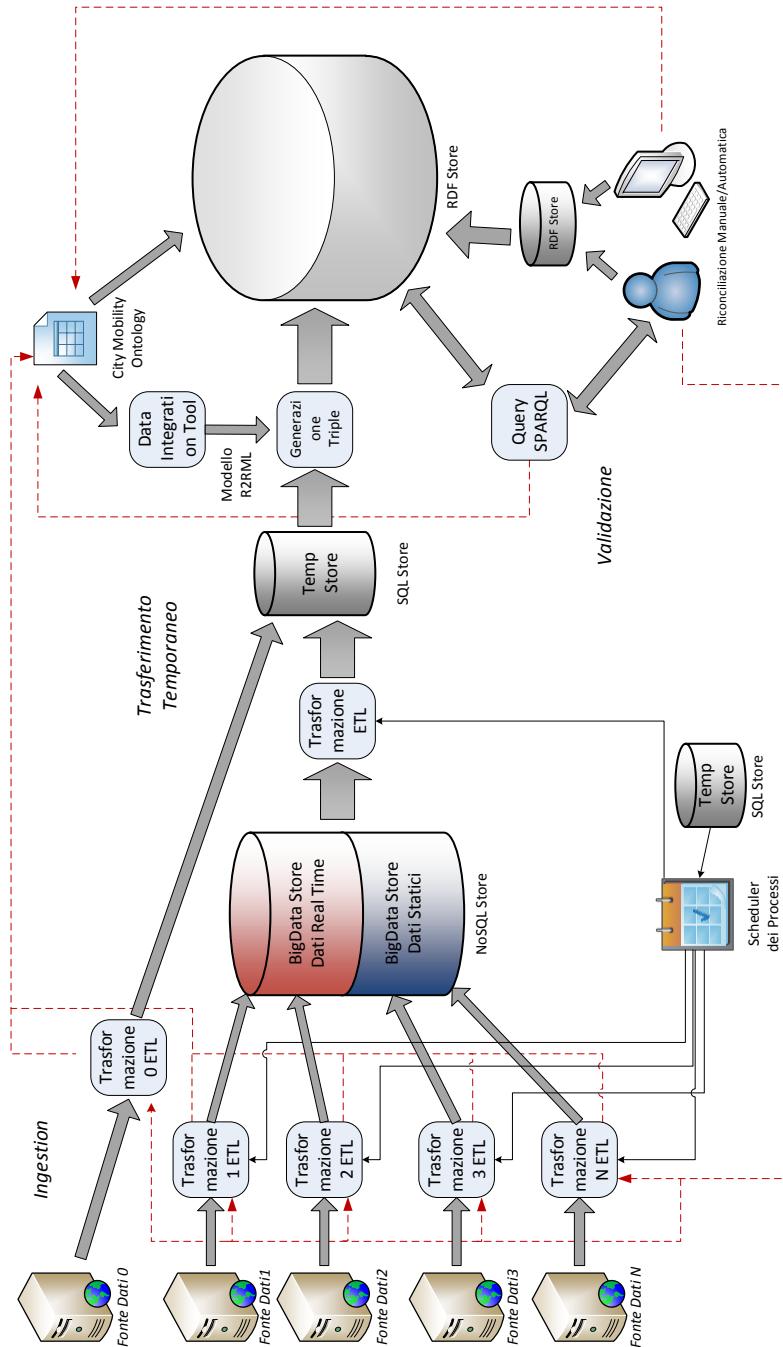


Figura 3.18. Rappresentazione grafica del processo globale di Ingestion attualmente in funzione per il processo Sii-Mobility

All'interno della figura è chiaramente visibile la fase di Ingestion dei dati statici e real-time provenienti da fonti diverse rispetto a quella del grafo stradale (**Fonte Dati 1**, **Fonte Dati 2**, eccetera). A causa della mole consistente di dati real-time da elaborare ogni giorno, per gli altri tipi di dati è stata prevista una fase intermedia di memorizzazione su un database di tipo **NoSQL Store** di tipo **Big Data**, ovvero HBase [5]. Inoltre, sulle trasformazioni ETL (etichettate come **Trasformazione 1 ETL**, **Trasformazione 2 ETL**, eccetera), è attualmente attivo uno **Scheduler dei Processi** che segue le istruzioni fornitegli da una tabella dei processi salvata in un **SQL Store** per determinare l'avvio delle trasformazioni. Nel prossimo futuro, si è già previsto di agganciare la fase di Ingestion relativa al grafo stradale verrà agganciata allo scheduler, anche se con politiche leggermente differenti.

3.2.1 Estrazione e Manipolazione Dati

In questa fase, si è provveduto alla definizione delle principali trasformazioni da applicare sui dati provenienti dall'Osservatorio. Tali dati grezzi, infatti, necessitano di subire molteplici manipolazioni al fine di renderli compatibili con l'ontologia creata.

Come spiegato dettagliatamente in appendice B.1, le manipolazioni sui dati sono state effettuate con delle trasformazioni ETL (Extract, Transform, Load) ed in particolare con il software Pentaho Kettle. In questa tesi, si è provveduto principalmente a lavorare su un'unica vasta trasformazione che gestisse tutti i differenti file del grafo stradale. Le trasformazioni relative alle altre parti dell'ontologia (TPL, Sensoristica, Servizi, eccetera) sono descritte in dettaglio nei lavori [1] e [2].

Il job principale per la trasformazione del grafo stradale è **Wrapper.kjb**. In figura 3.19 è mostrato il dettaglio del job all'interno dell'IDE di Pentaho Kettle.

Fase di Ingestion



Figura 3.19. Visualizzazione tramite Spoon del job principale della macrosezione Grafo Stradale

La prima trasformazione che viene richiamata è il lo script **coordinate conversion**. In questa fase vengono convertiti tutti i file di tipo ShapeFile, presenti all'interno delle cartelle ottenute dall'Osservatorio Trasporti, in formato KML. Questa conversione si è resa necessaria per due motivi: principalmente perchè il software Pentaho Kettle presenta un bug nell'importazione di file di tipo .shp, in secondo luogo perchè i dati provenienti dall'Osservatorio, sono georeferenziati utilizzando una proiezione cartografica particolare.

Il bug del software Pentaho si presenta ogni volta che si tenta di importare un file .shp contenente informazioni non puntuali ma di tipo polilinea come, ad esempio, i dati relativi alle singole coordinate degli elementi stradali (si veda il paragrafo 3.1.2.1), oppure di tipo poligonale (ad esempio i file dove sono memorizzate le estensioni dei singoli comuni o delle province). Durante l'importazione del file, per un comportamento anomalo del software, questi singoli punti distinti vengono collassati ad un unico punto, perdendo quindi tutte le informazioni di interesse.

In secondo luogo, le coordinate di questi punti sono memorizzate secondo la proiezione cartografica Gauss-Boaga (uno standard adottato in Italia negli anni passati). Ad oggi, la maggior parte delle informazioni geografiche viene memorizzato utilizzando la proiezione standard WGS84.

Per risolvere questo doppio problema, è stato quindi richiamato un comando del software QGIS (appendice B.2) dal nome **ogr2ogr**. Questo script è in grado di convertire informazioni geografiche tra diversi tipi di formati e attraverso molteplici proiezioni cartografiche. Per richiamarlo, è stato sufficiente creare un semplice file batch che lo invochi ed utilizzare lo step **Execute a shell script** di Pentaho Kettle. In particolare, lo script batch è il seguente:

```

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

```

Figura 3.20. Script per la conversione di file da .shp a .kml e da Gauss-Boaga a WGS84

Il ciclo **for** estrae tutti i file con estensione .shp dalla cartella in cui sono memorizzati i dati dell’Osservatorio, e dalle sue sottocartelle. Su ognuno di questi file viene eseguita la conversione tramite il comando **ogr2ogr**. Il parametro *-f* indica il formato di destinazione del file, quindi “*KML*”, *-overwrite* permette la sovrascrittura del file generato, nel caso in cui venga eseguito nuovamente lo script, “*%f.kml*” impone che il file venga salvato con lo stesso nome ma con estensione .kml, infine i parametri *-s_srs* e *-t_srs* mappano i codici di riferimento geodetici univoci relativi, rispettivamente, a Gauss-Boaga (Monte Mario) e WGS84.

Questo script viene salvato in un file chiamato **conversion.bat** e richiamato tramite Kettle, come mostrato nella seguente figura 3.21:

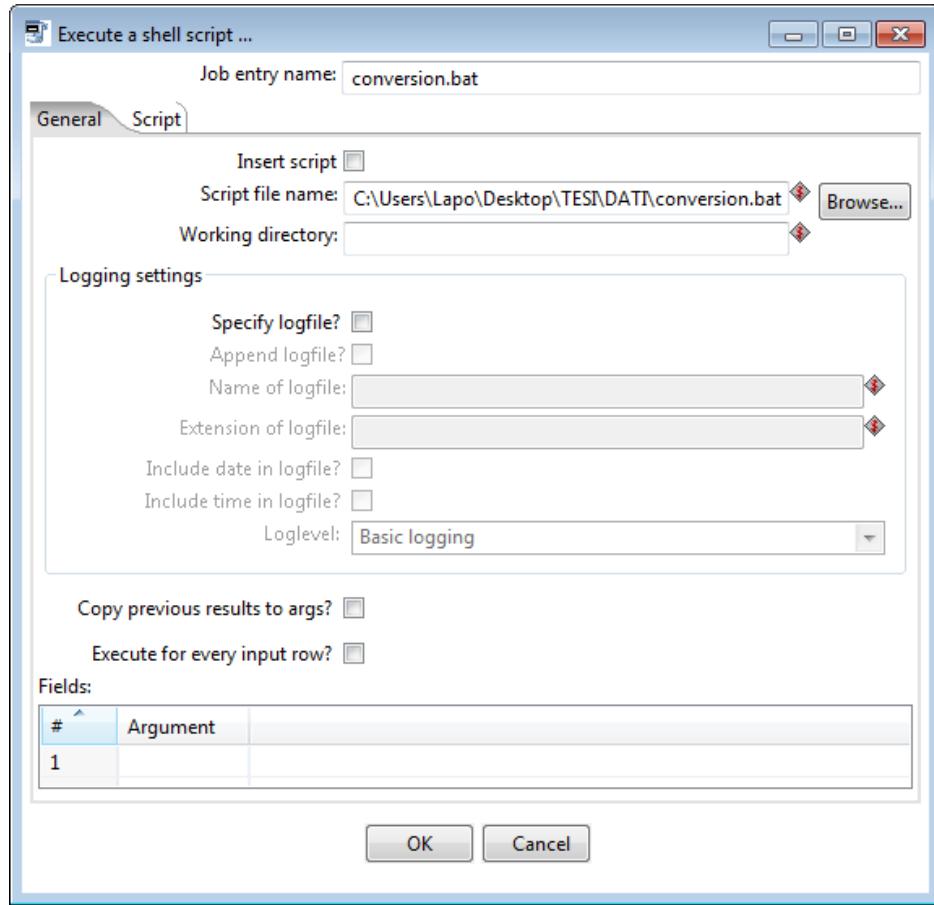


Figura 3.21. Dettaglio dello step Execute a shell script all'interno dell'ambiente Kettle

Dopo aver convertito tutti gli ShapeFile in KML, si passa all'elaborazione vera e propria dei file. Poiché solitamente i dati sono ottenuti dall'Osservatorio Trasporti suddivisi per provincia, è necessario indicare a Kettle che le stesse operazioni devono essere compiute su più sottocartelle all'interno della directory principale dei dati. Pertanto, il job principale, denominato **Main_Job.kjb** viene preceduto dalla trasformazione **Get_Folders.ktr**, attraverso la quale si estraggono tutte le sottocartelle presenti nella cartella dei dati, e si passano al job principale attraverso una struttura interna a Kettle, tramite lo step **Copy rows to result**, come mostrato in figura 3.22

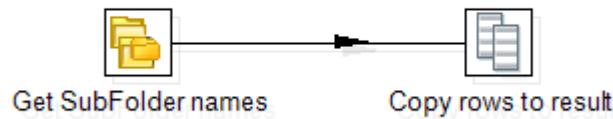


Figura 3.22. Dettaglio della trasformazione Get_Folders.ktr

Ovviamente, nel primo step di questa trasformazione, ovvero in **Get SubFolder Names**, deve essere impostato il percorso della cartella ove sono stati salvati i dati ricevuti dall’Osservatorio (dettaglio in figura 3.23).

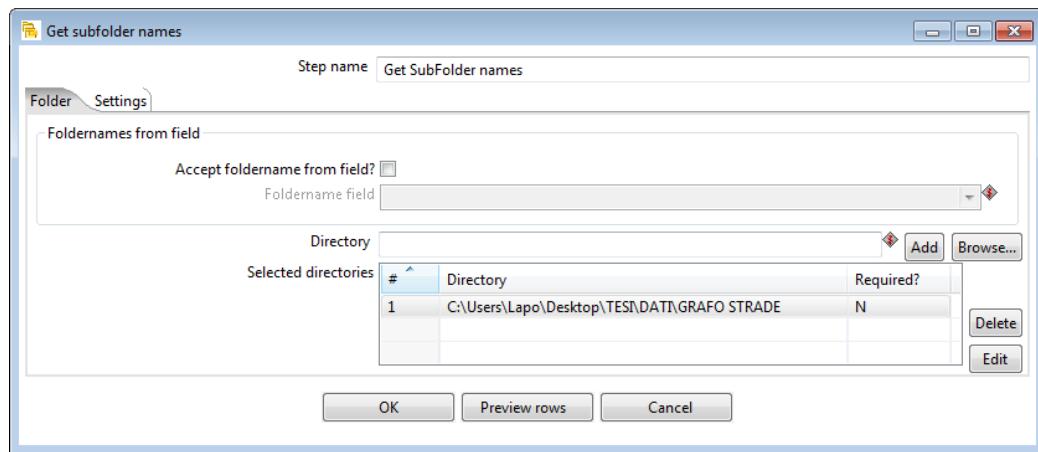


Figura 3.23. Dettaglio dello step Get SubFolder Names

Una volta salvato l’elenco delle sottocartelle, è necessario imporre a **Main_Job.kjb** di ripetere tutti gli step interni su ogni cartella. Per farlo, entrando nelle impostazioni del job, è sufficiente spuntare la casella **Execute for every input row?** all’interno del secondo tab denominato *Advanced*. In figura 3.24 è mostrata tale procedura.

Fase di Ingestion

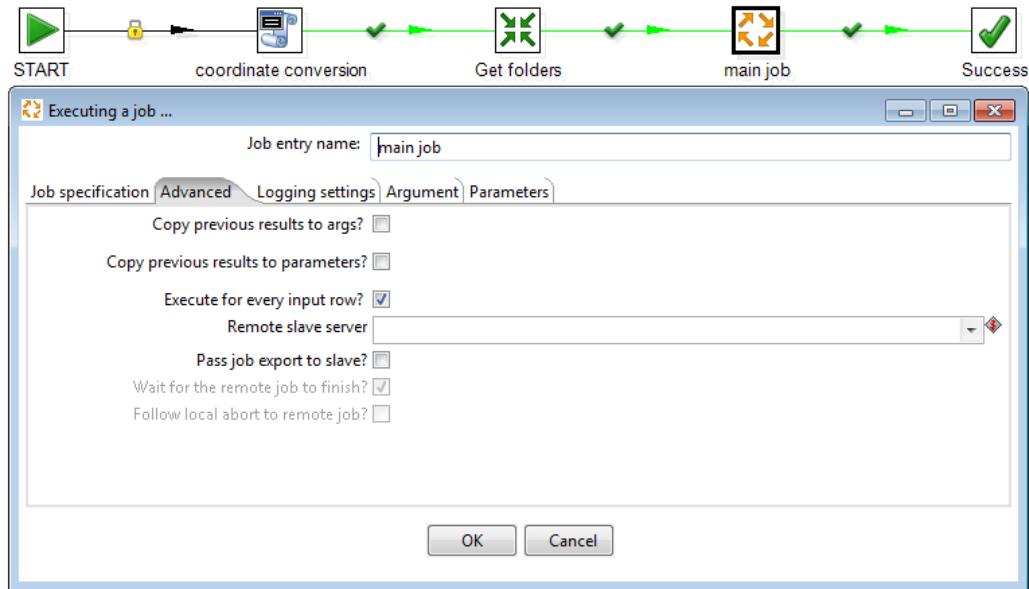


Figura 3.24. Impostazioni del job main _job.kjb

Nella prossima figura (3.25) è mostrato il job principale:

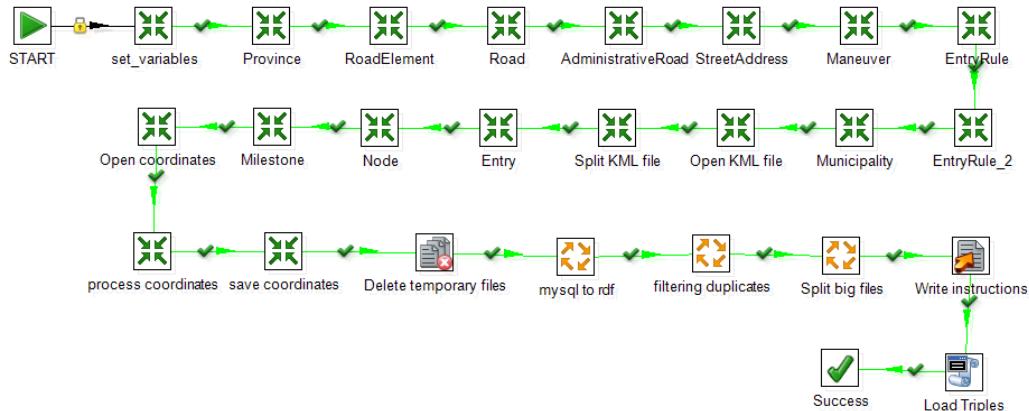


Figura 3.25. Dettaglio del job Main _ Job.kjb

All'interno del job principale, il procedimento seguito per la maggior parte dei dati del grafo stradale è il seguente: si prelevano i dati dai file corrispondenti, si lavorano tramite Kettle ed infine si salvano su tabelle MySQL temporanee strutturate ad hoc. Una volta che tutti i dati di interesse sono

stati salvati su MySQL, si provvede alla loro trasformazione in formato RDF ed alla loro memorizzazione su repository semantico.

La prima trasformazione, **set_variables**, legge il nome della cartella attualmente in lavorazione dal resultset nativo di Kettle e la memorizza in una variabile denominata **FOLDER**, come mostrato in figura 3.26. Le variabili sono una delle caratteristiche più interessanti di Pentaho Kettle e servono a memorizzare informazioni richiamabili in seguito, grazie alla referenziazione col carattere \$.

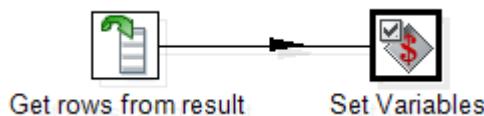


Figura 3.26. Dettaglio della trasformazione set_variables.ktr

La trasformazione successiva è denominata **Province.ktr** e prevede la memorizzazione su una specifica tabella di appoggio MySQL, dei dati facenti parte della Provincia. La trasformazione è composta dagli step mostrati in figura 3.27.

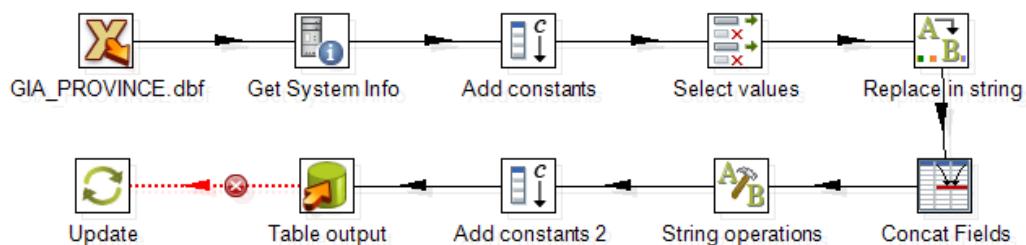


Figura 3.27. Dettaglio della trasformazione Province.ktr

Il primo step di tipo **GIA_PROVINCE.dbf** legge i dati dall'omonimo file **GIA_PROVINCE.dbf** presente all'interno della cartella del grafo stradale. Gli step successivi sono necessari per la generazione della data di creazione dei dati nel formato *xsd:dateTime* suggerito dal W3C, come descritto nel paragrafo 3.1.2.6. In particolare, lo step **Get System Info** preleva dal sistema operativo la data corrente e la memorizza nel campo *update_date*, lo

step **Add constants** crea un nuovo campo contenente la stringa “*+01:00*” che rappresenta la *timezone* in vigore in Italia, mentre lo step **Select Values**, all’interno del tab *Meta-data* converte esplicitamente alcuni campi in stringa, per una corretta lavorazione successiva. Lo step **Replace in String**, nuovamente, manipola la stringa contenente la data *update_date* per adeguarla al formato corretto, mentre la procedura di **Concat Fields**, concatena il campo *update_date*, appena modificato, con il campo *timezone* creando il campo *ISO_DATE*, che contiene la data completa di ultimo aggiornamento dei dati, nel formato corretto. Infine, dato che, per un comportamento anomalo del software, in taluni casi, durante la fase di *Concat Fields*, vengono erroneamente inseriti degli spazi in coda o in testa alla stringa, si è forzato il trim della stringa grazie allo step **String operations**. L’ultimo passo prima del salvataggio dei dati su MySQL, è rappresentato da un nuovo step di **Add constants**, in cui viene definito un nuovo campo *ORG*, all’interno del quale viene inserito il valore costante “*Osservatorio Trasporti Regione Toscana*” che rappresenta la fonte del dato e che verrà memorizzato nel campo *dc:source* dell’ontologia. I campi *dc:created* e *dc:source* vengono generati, nelle prossime trasformazioni, per tutti gli oggetti, ma non vengono caricati direttamente nel repository, per il motivo spiegato in 3.1.2.1. L’ultima fase di questa trasformazione, prevede il salvataggio dei dati modificati nella tabella temporanea MySQL, chiamata **tbl_province** creata ad hoc. Il doppio step **Table output - Update** è stato preferito al singolo step **Insert/Update** per questioni di ottimizzazione e per un migliore approccio nell’eventuale aggiornamento dei dati. In pratica, il primo step tenta di inserire un nuovo record all’interno della tabella MySQL, in caso di fallimento (che avviene nel caso in cui si tenti di inserire un record con una chiave **COD_ELE** già presente), Pentaho Kettle applica la sua politica di gestione delle eccezioni e, seguendo quella che in figura è mostrata come una linea tratteggiata rossa, esegue un update del record avente quella particolare chiave.

La trasformazione successiva, denominata **Road_Element.ktr**, serve per la memorizzazione dei dati relativi agli elementi stradali. In figura 3.28 è mostrata la sequenza di step che compone la trasformazione.

Fase di Ingestion

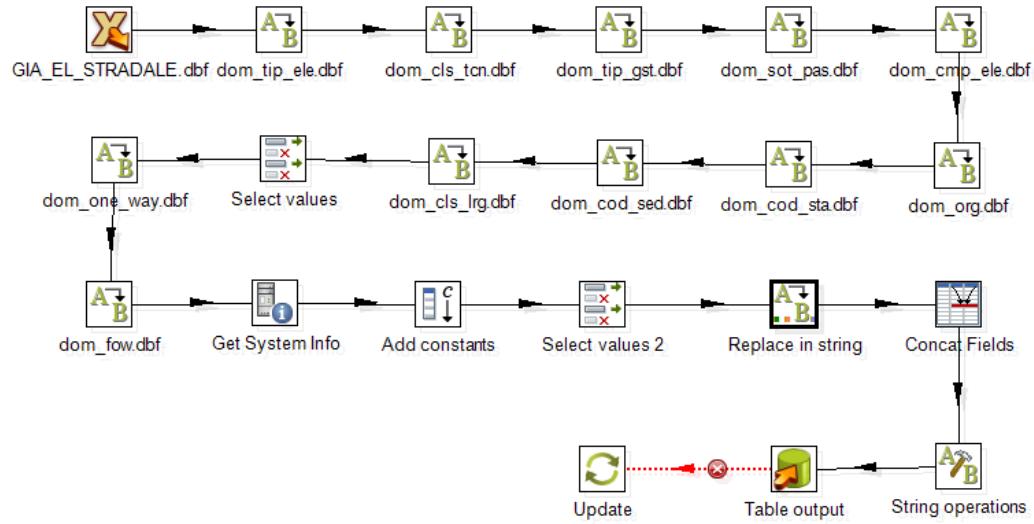


Figura 3.28. Dettaglio della trasformazione Road_Element.ktr

In questa trasformazione e nelle successive, spesso si ripetono dei pattern già descritti nella precedente trasformazione, quindi di seguito verranno descritte in dettaglio esclusivamente le nuove elaborazioni.

Il primo step **GIA_EL_STRADALE.dbf** carica nel programma i dati provenienti dall'omonimo file presente all'interno dell'archivio del grafo stradale. Gli step successivi, la cui etichetta inizia con “*dom_*” (come ad esempio **dom_tip_ele.dbf**, **dom_cls_tcn.dbf**, eccetera) sono tutti di tipo **Replace in string**, e servono per mappare i differenti attributi dell'elemento stradale nei corrispondenti valori di dominio. In figura 3.29 è mostrato un esempio di questo procedimento.

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	VALORE,C,4	DESCRIZION,C,250
0100	autostrada	0100	di tronco carreggiata
0200	extraurbana principale	0200	di area a traffico strutturato
0300	extraurbana secondaria	0201	di casello/barriera autostradale
0400	urbana di scorrimento	0202	di passaggio a livello

[dom_cls_tcn.dbf](#) [dom_tip_ele.dbf](#)

Figura 3.29. Procedura di mappatura degli attributi della classe RoadElement nei corrispondenti valori di dominio

Tra i vari step appena citati compare uno step differente (**Select values**). Anche in questo caso si utilizza il tab *Meta-data* per la conversione esplicita di un particolare campo in stringa, al fine di elaborarlo correttamente negli step successivi. I successivi step (**Get System Info**, **Add constants**, **Select values 2**, **Replace in string**, **Concat Fields** e **String operations**) servono nuovamente per la generazione della data di creazione delle triple, come nella trasformazione precedente. Infine, anche in questo caso, si procede al salvataggio dei dati nella consueta tabella MySQL, chiamata in questo caso **tbl_el_stradale**.

Appena terminata la trasformazione **Road_Element.ktr**, si avvia la trasformazione **Road.ktr**. Essa risulta essere differente rispetto alle altre viste fino ad ora, a causa delle relazioni complesse che intercorrono tra gli elementi di tipo RoadElement, Road e AdministrativeRoad, come descritto nel paragrafo 3.1.2.1. Di seguito, in figura 3.30 è mostrato il dettaglio della trasformazione.

Fase di Ingestion

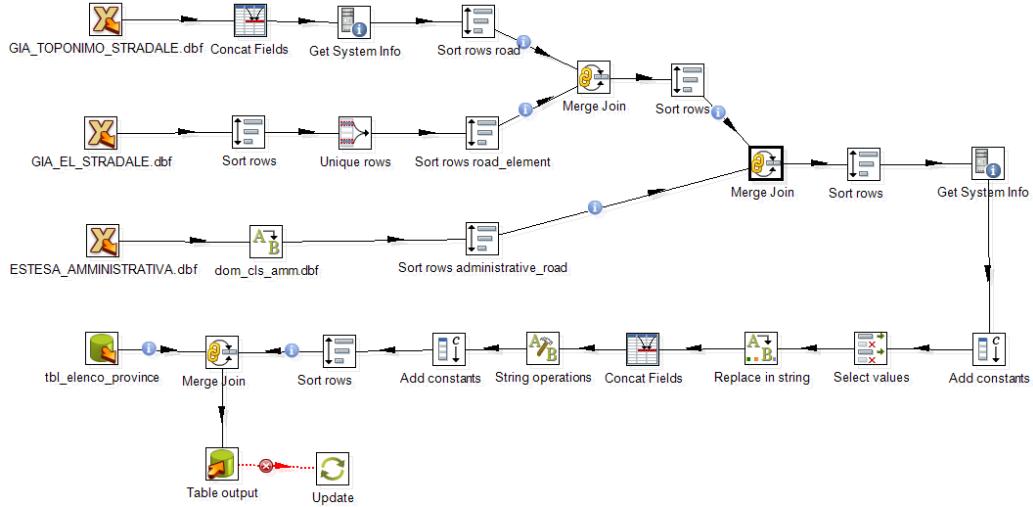


Figura 3.30. Dettaglio della trasformazione Road.ktr

In input alla trasformazione vengono letti i file

GIA_TOPONIMO_STRADALE.dbf, **GIA_EL_STRADALE.dbf** e **ESTESA_AMMINISTRATIVA.dbf**, oltre ad una tabella di supporto MySQL denominata **tbl_elenco_comuni** che sarà descritta in seguito. Su questi tre file vengono definite alcune operazioni: sui dati di Toponimo si crea inizialmente un nuovo campo chiamato *EXT_NAME* che contiene al suo interno la concatenazione tra il DUG del toponimo ed il nome della via (si vedano i dettagli in sezione 3.1.2.1), viene aggiunto il campo *update_date* tramite la solita procedura **Get System Info** ed infine, si ordinano i record in base alla chiave *COD_TOP*, grazie allo step **Sort rows road**. Sui dati provenienti dal file **GIA_EL_STRADALE.dbf**, invece, viene eseguita una operazione leggermente diversa: all'inizio le righe vengono ordinate secondo due chiavi, il codice del toponimo al quale l'elemento stradale appartiene ed il codice dell'estesa amministrativa. A questo punto, i record che hanno entrambe queste chiavi uguali vengono filtrati tramite lo step **Unique rows**, quindi rimangono esclusivamente le righe con codice toponimo e codice estesa univoci. Si ordinano nuovamente i record ottenuti in base al codice toponimo e, grazie alla procedura **Merge join**, si fondono i dati ottenuti con quelli provenienti dal file dei toponimi. Nel frattempo, i dati del file **ESTESA_AMMINISTRATIVA.dbf**, sono a loro volta ordinati

sulla chiave *COD_REG*, ed aggiornati con i dati della tabella di dominio **dom_cls_amm.dbf**. Si esegue un nuovo **Merge Join** tra i dati dell'estesa ed i dati lavorati in precedenza, e si ottiene un unico flusso in cui all'interno dei singoli record sono concatenati i dati di interesse del toponimo e quelli di raccordo con Elementi Stradali ed Estese Amministrative. Al termine di queste due fusioni si esegue nuovamente la procedura per la formattazione della data di generazione dei dati, e si aggiunge la consueta colonna con la stringa “*Osservatorio Trasporti Regione Toscana*”, per indicare la sorgente dei dati. L'ultimo step precedente al salvataggio su MySQL, è l'ennesimo **Merge join** tra i dati e la tabella di supporto **tbl_elenco_comuni**, che permette di trasformare il dato identificativo del comune di appartenenza della strada con il corretto codice ISTAT corrispondente. Dopo questa fusione, i dati vengono salvati nella tabella **tbl_toponimo**.

Dopo aver elaborato elementi stradali e toponimi, si passa alla generazione dei dati relativi alle estese amministrative, grazie alla trasformazione **Administrative_Road.ktr**. In figura 3.31 è mostrato il procedimento all'interno dell'IDE di Pentaho Kettle.

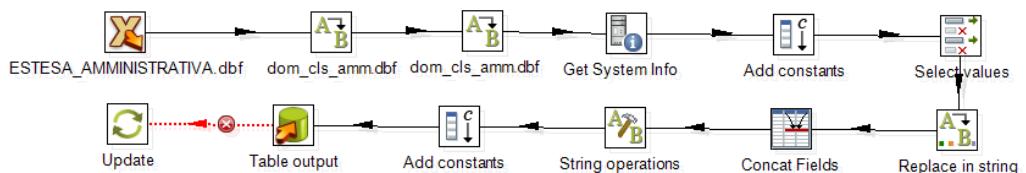


Figura 3.31. Dettaglio della trasformazione *Administrative_Road.ktr*

In questa trasformazione non vi sono sostanziali novità rispetto alle precedenti, l'elaborazione procede nel modo già descritto: i dati vengono prelevati dal file **ESTESA_AMMINISTRATIVA.dbf**, vengono mappate le tabelle di dominio, vengono gestite la data di ultimo aggiornamento e la fonte del dato, ed infine i dati sono salvati nella tabella MySQL **tbl_estesa**.

Anche nella trasformazione successiva, ovvero nella **Street_Number.ktr**, non vi sono elaborazioni diverse da quelle già descritte, pertanto viene mo-

strata esclusivamente la rappresentazione grafica del procedimento in figura 3.32.

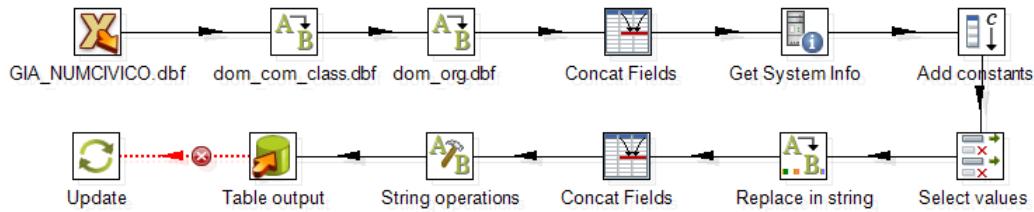


Figura 3.32. Dettaglio della trasformazione Street_Number.ktr

Il passo successivo del job principale è costituito dalla trasformazione **Maneuver.ktr**, mostrata in figura 3.33

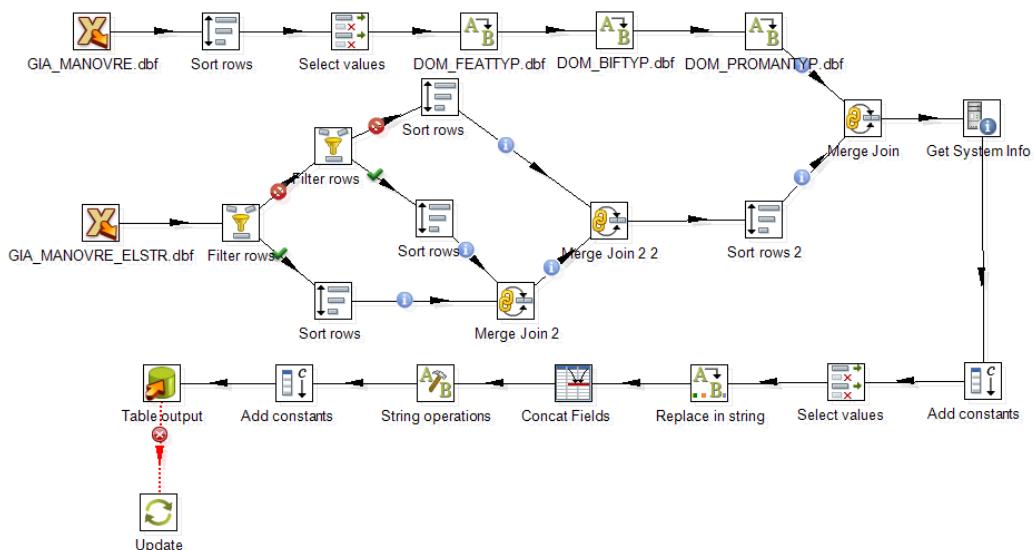


Figura 3.33. Dettaglio della trasformazione Maneuver.ktr

Questa trasformazione risulta notevolmente più complessa, a causa della semantica dei dati relativi alle manovre. In particolare, i dati rilasciati dall'Osservatorio si basano sulla definizione di due file: **GIA_MANOVRE.dbf** e **GIA_MANOVRE_ELSTR.dbf**. Nel primo file vengono elencate le singole manovre (consentite o proibite che siano) e la loro tipologia, mentre nel secondo si descrive quali elementi stradali vengono coinvolti da queste manovre. In particolare, si ha che una manovra è sempre definita su di una **Giun-**

zione e che coinvolge un numero di elementi stradali pari a 2 o 3, a seconda del tipo di manovra. Quindi, all'interno del file **GIA_MANOVRE.dbf**, sarà presente l'identificativo univoco della giunzione (*COD_GNZ*) sulla quale è definita la manovra, mentre nel file **GIA_MANOVRE_ELSTR.dbf** sono elencati gli elementi stradali coinvolti. Per come è stata definita l'ontologia (si veda il paragrafo 3.1.2.1), e per come vengono generati i file RDF corrispondenti (sezione 3.2.2), si è resa necessaria questa trasformazione complessa.

All'avvio della trasformazione vengono aperti nel software entrambi i file appena descritti. Il file primario (**GIA_MANOVRE.dbf**) viene riordinato secondo l'identificativo della manovra, ne vengono convertite alcune colonne in stringa, mentre altre vengono mappate in base ai valori di dominio. Il file secondario, invece, deve subire una mutazione più profonda: poichè i dati di una singola manovra sono ripetuti su più righe, è necessario effettuare una sorta di trasposizione tra righe e colonne dei dati. Non esistendo uno step predefinito per farlo, è risultato necessario utilizzare in cascata due filtri di tipo **Filter rows**, che suddividono il flusso di informazioni in base al valore dell'attributo *SEQNR*, ovvero il numero sequenziale dell'elemento stradale su quella particolare manovra. Questi tre distinti flussi vengono poi fusi nuovamente tramite lo step **Merge join** con chiave l'identificativo della manovra. In questo modo ogni manovra avrà un singolo record con più colonne riservate agli elementi stradali. Una volta terminata questa operazione, i dati provenienti dai due file vengono fusi, previo ordinamento sulla colonna contenente l'identificativo della manovra (Kettle impone che lo step di **Sort rows** sia eseguito esplicitamente prima di **Merge join** affinchè la fusione sia corretta). Ai dati collegati, si aggiunge l'informazione relativa alla data di aggiornamento ed alla fonte; infine, tutti i dati vengono riversati nella tabella MySQL **tbl_manovre**.

La trasformazione successiva riguarda le *regole di accesso*. In figura 3.34 è mostrata la trasformazione **Entry_Rule.ktr**.

Fase di Ingestion

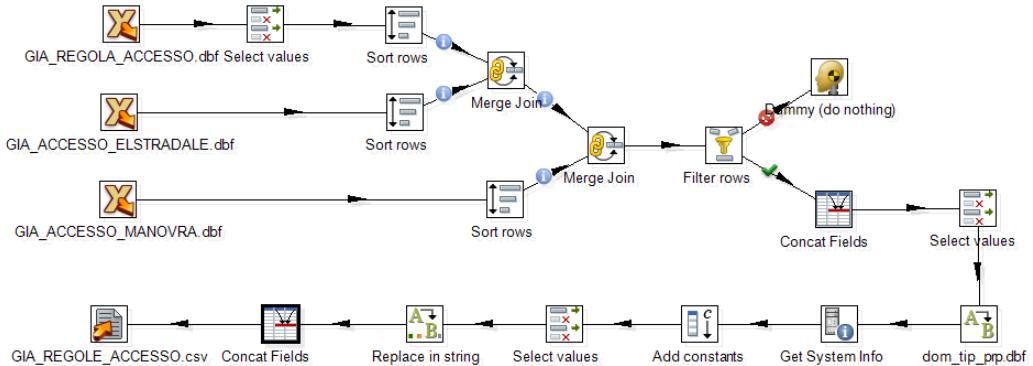


Figura 3.34. Dettaglio della trasformazione Entry_Rule.ktr

Le regole di accesso rappresentano limitazioni o permessi di accesso a determinati elementi stradali o su determinate manovre. Il loro ambito di utilizzo è molto vasto, infatti possono permettere di memorizzare informazioni inerenti, ad esempio, a zone a traffico limitato (ZTL), possono validare piani di impianti semaforici, eccetera. Allo stato attuale tuttavia, sono presenti informazioni esclusivamente all'interno di tre file:

- **GIA_REGOLA_ACCESSO.dbf**
- **GIA_ACCESSO_ELSTRADALE.dbf**
- **GIA_ACCESSO_MANOVRE.dbf**

Il file principale è **GIA_REGOLA_ACCESSO.dbf**, il quale contiene i singoli record assieme alle proprietà della regola (quali ad esempio il tipo di restrizione, o la descrizione della stessa). Gli altri due file fungono semplicemente da tabelle di raccordo in quanto, al loro interno, si trovano esclusivamente l'identificativo della regola di accesso e dell'elemento stradale (o della manovra). Poiché per la generazione delle triple RDF corrispondenti è necessario riportare tutti i dati di interesse su di una singola riga, questa trasformazione prevede un utilizzo massivo dello step di **Merge join**, al fine di concatenare in un unico record i dati provenienti dai tre distinti file sorgente. A valle dell'ultimo *Merge join* si trova un **Filter rows** che scarta quei record che sono incompleti, quelli cioè in cui non è presente né la corrispondenza

tra l'ID della regola e dell'elemento stradale, né quella con l'ID della manovra. Le regole di accesso possono essere di molti tipi differenti, ogni regola di accesso ha infatti due attributi che ne descrivono la tipologia: *RSTTYP* e *RESTRVAL*. Il primo attributo può assumere alcuni dei seguenti valori: *DF* (“*Direzione Flusso di Traffico*”), *BP* (“*Passaggio Bloccato*”), *SR* (“*Restrizioni Speciali*”), eccetera, mentre *RESTRVAL* tipicamente assume dei valori numerici che però variano di significato in base al valore contenuto in *RSTTYP*. Ad esempio, nel caso in cui il primo attributo sia *DF*, il valore *2* contenuto in *RESTRVAL* indica “*Chiusa in direzione positiva*”; se invece *RSTTYP* contiene il valore *BP*, lo stesso valore *2* contenuto in *RESTRVAL* stabilisce che la regola di accesso è di tipo “*Blocco fisico sulla giunzione finale*”. Per questo motivo, dopo lo step **Filter rows**, viene inserita una procedura di **Concat fields** sui campi *RSTTYP* e *RESTRVAL* che va a sovrascrivere il valore stesso di *RESTRVAL* con la concatenazione dei due valori. In questo modo, *RESTRVAL* contiene i valori distinti “*DF2*” e “*BP2*”. Il valore appena prodotto viene forzato a stringa tramite **Select values**, mentre sulla colonna *RSTTYP* si esegue la mappatura con i valori possibili di dominio. Purtroppo, a causa di un piccolo bug del software Pentaho Kettle, non è possibile applicare la mappatura dei valori direttamente sul campo *RESTRVAL* appena aggiornato. Per questo motivo, tale operazione viene delegata ad una trasformazione successiva, chiamata **Entry_Rule2.ktr** che sarà descritta in seguito. Prima di terminare la trasformazione, viene generata la data di creazione secondo il formato *xsd:dateTime* e, a differenza di tutte le trasformazioni precedenti, i dati vengono salvati su un file di appoggio dal nome **GIA_REGOLE_ACCESSO.csv**, piuttosto che su una tabella MySQL, proprio in virtù del problema appena descritto.

Come appena accennato, per terminare correttamente la generazione dei dati relativi alle regole di accesso, si è ricorsi ad una seconda trasformazione, **Entry_Rule2.ktr**, di cui è mostrato lo schema in figura 3.35

Fase di Ingestion

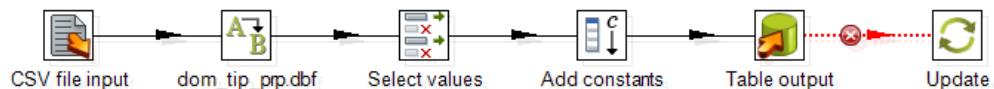


Figura 3.35. Dettaglio della trasformazione Entry_Rule2.ktr

In questa seconda parte di trasformazione, si leggono i dati dal file **GIA_REGOLE_ACCESSO.csv** appena salvato, si esegue la mappatura sui dati della colonna *RESTRVAL*, come mostrato in figura 3.36, si forzano i campi contenenti la data e l'identificativo (eventuale) della manovra a stringa, ed infine si inserisce il consueto valore “*Osservatorio Trasporti Regione Toscana*” all’interno del campo che contiene la fonte dei dati. Al solito, le informazioni elaborate vengono salvate nella tabella MySQL **tbl_regoles_accesso**.

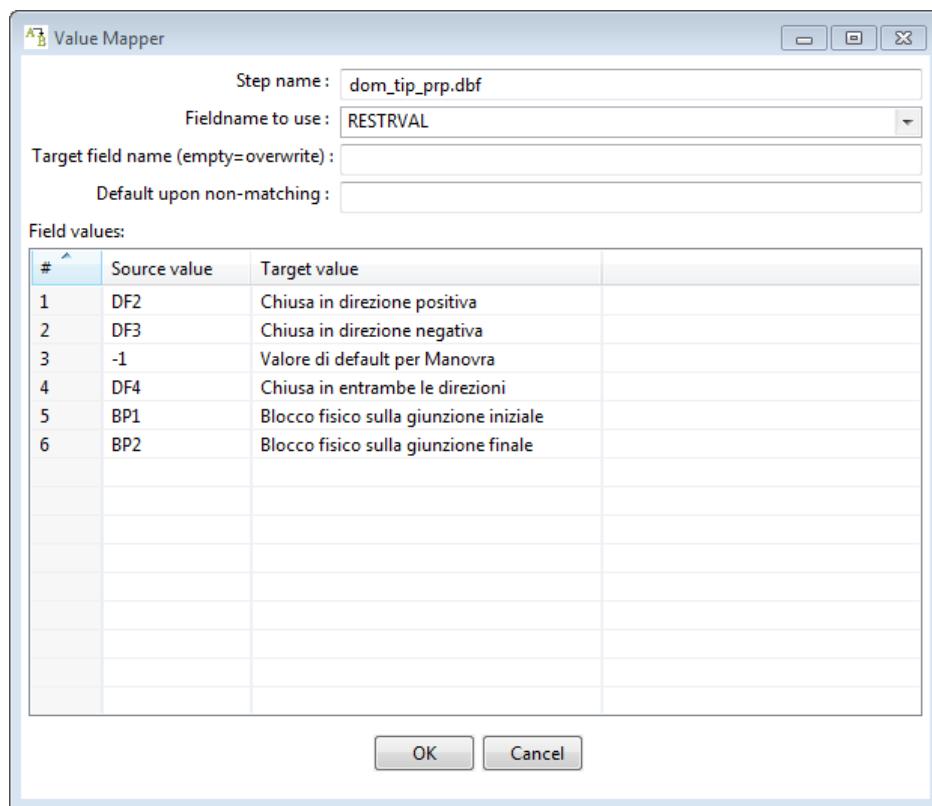


Figura 3.36. Impostazioni dello step dom_tip_prp.dbf che mappa il contenuto dell’attributo RESTRVAL nei corrispondenti valori di dominio

Municipality.ktr è la trasformazione successiva. Essa prevede la memorizzazione su MySQL dei dati relativi ai comuni. In figura 3.37 è visibile la sua rappresentazione grafica tramite interfaccia Kettle.

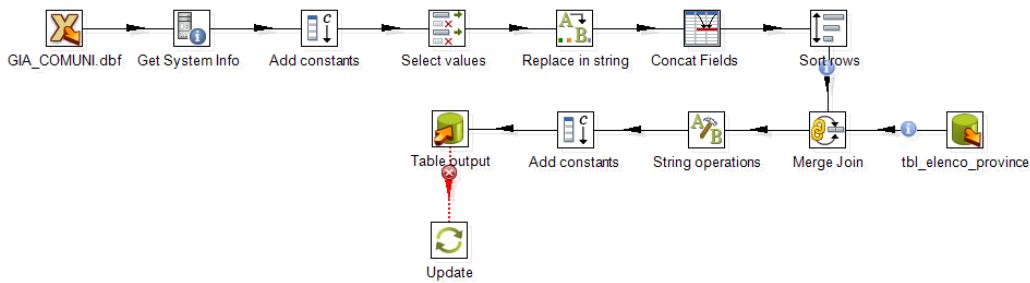


Figura 3.37. Dettaglio della trasformazione **Municipality.ktr**

La trasformazione è molto simile a quella relativa alle province, ovvero a **Province.ktr** descritta in precedenza. L'unica differenza sostanziale è la lettura della tabella MySQL di appoggio **tbl_elenco_province**, all'interno della quale sono memorizzati i codici ISTAT di tutte le province toscane. All'interno del repository viene utilizzato il codice ISTAT al posto del codice catastale (o *Codice Belfiore*) proveniente da alcuni dati dell'osservatorio. Tramite lo step **Merge join** si riesce ad appendere al flusso di informazioni, anche il codice ISTAT della provincia della quale fa parte ogni comune. La tabella MySQL di destinazione dei dati di questa trasformazione si chiama **tbl_comune**.

Fino all'ultima trasformazione descritta sono stati manipolati esclusivamente dati provenienti da file di tipo *XBase*. Le successive cinque trasformazioni (**Open_KML_File.ktr**, **Split_KML_File.ktr**, **Entry.ktr**, **Node.ktr** e **Milestone.ktr**), invece, hanno in input file di tipo *KML*, i quali sono stati convertiti come descritto all'inizio di questo paragrafo. Nelle prossime trasformazioni si elaborano dati appartenenti ai concetti di **Acesso**, **Giunzione** e **Cippo Chilometrico**, che hanno associate delle informazioni di tipo geospaziale e che quindi non possono essere elaborate direttamente in formato ShapeFile, a causa del bug già descritto in precedenza che affligge Pentaho Kettle. La conversione in KML implica un

considerevole aumento nelle dimensioni del file (mediamente il file convertito risulta essere dalle 3 alle 5 volte più grande del file originale) e ciò provoca un notevole rallentamento nella processazione del file. Inoltre, essendo il KML un derivato del linguaggio XML, il file risulta notevolmente più complicato da gestire, rispetto ad un file tabulare come *CSV* o *DBF*, anche usando un ETL come Pentaho Kettle. Per questo motivo, i file KML relativi ad Accessi, Giunzioni e Cippi chilometrici vengono aperti in parallelo da Kettle attraverso le procedure omonime **GIA_GIUNZIONE.kml**, **GIA_ACCESSO.kml** e **GIA_CIPPO.kml**, facenti parte della trasformazione **Open_KML_File.ktr**, e subito dopo riscritti in formato CSV nei file temporanei **GIA_GIUNZIONE.csv**, **GIA_ACCESSO.csv** e **GIA_CIPPO.csv**, poichè più facili da manipolare in seguito. In figura 3.38 è mostrata tale sequenza di istruzioni.

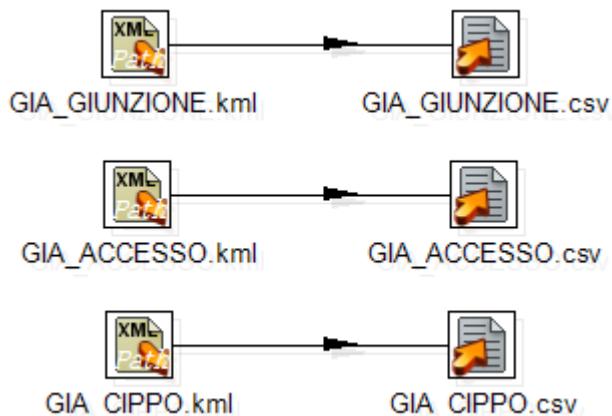


Figura 3.38. Dettaglio della trasformazione **Open_KML_File.ktr**

La trasformazione successiva (**Split_KML_File.ktr**), carica in input i file temporanei appena salvati. A causa del passaggio attraverso il formato KML, gli unici due campi prelevabili in questo momento sono: *ExtendedData*, che contiene tutti gli attributi, e *Point*, che invece contiene le informazioni georeferenziate. Tramite lo step **String operations** si esegue il trim del campo *ExtendedData*, che al proprio interno contiene degli spazi e dei ritorni a capo inutili ai fini dell'informazione, ma soprattutto dannosi per Kettle. I due step **Split fields** successivi servono proprio per ricostruire le singole

colonne degli attributi e per suddividere le coordinate dei punti tra longitudine e latitudine. In figura 3.39 sono mostrate le impostazioni per i due step consecutivi.

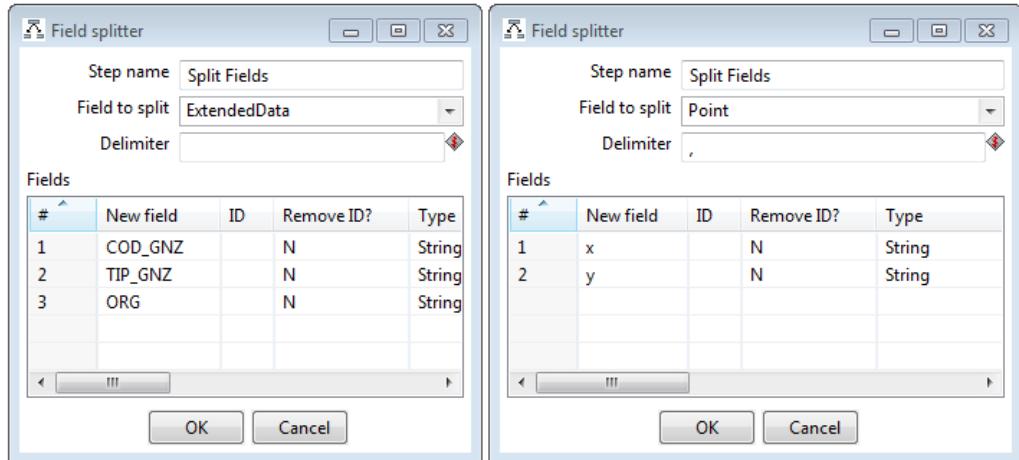
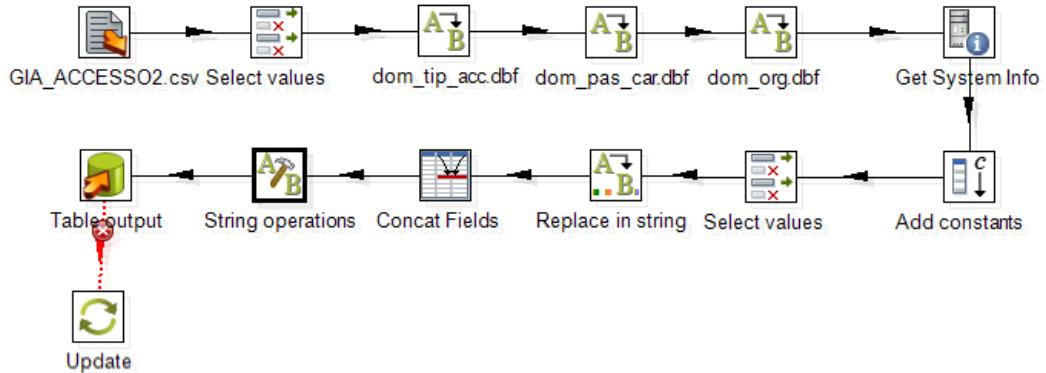


Figura 3.39. Impostazioni degli step Split fields per il recupero delle singole informazioni all'interno dei dati KML

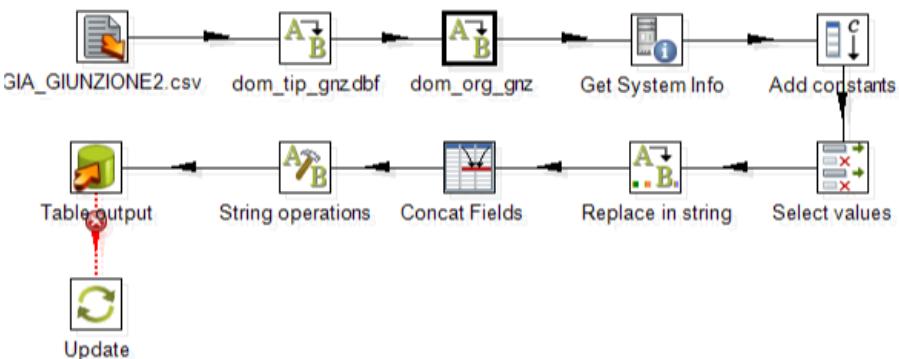
Il contenuto dei dati viene salvato nuovamente in un altro file temporaneo (rispettivamente **GIA_GIUNZIONE2.csv**, **GIA_ACCESSO2.csv** e **GIA_CIPPO2.csv**), sempre a causa del piccolo bug di Kettle, che impedisce di effettuare mappature di dominio su colonne generate all'interno della stessa trasformazione.

Le trasformazioni successive non differiscono molto tra loro, in quanto le operazioni da effettuare sono all'incirca le medesime (mappatura degli attributi nei valori di dominio, generazione della data di aggiornamento e del campo fonte dei dati). Per questo motivo viene mostrata esclusivamente una figura in cui sono presenti tutte e tre le trasformazioni (figura 3.40).

Trasformazione Entry.ktr



Trasformazione Node.ktr



Trasformazione Milestone.ktr

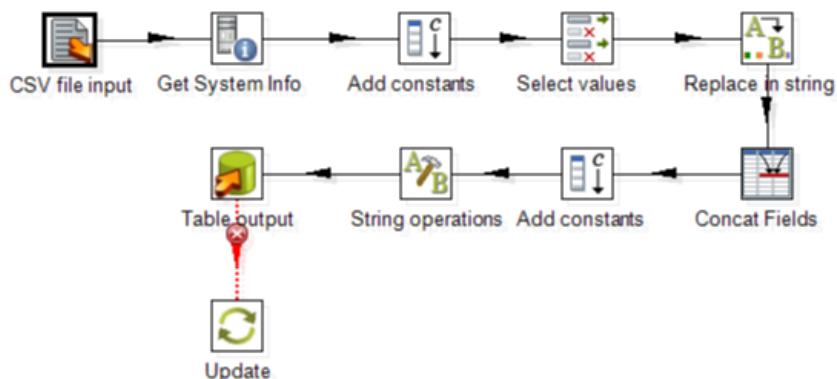


Figura 3.40. Dettaglio delle trasformazioni Entry.ktr, Node.ktr e Milestone.ktr

Le tabelle MySQL in cui vengono salvati i dati sono: **tbl_accesso**, **tbl_giunzione** e **tbl_cippo**.

L'ultima tipologia da trattare prima di passare agli step di generazione delle triple RDF e di inserimento sul repository semantico, è rappresentata dalle cosiddette *coordinate degli elementi stradali*. Si è già descritta la questione in sezione 3.1.2.1. Il trattamento di questi dati richiede tre diverse trasformazioni, simili a quelle appena viste per la generazione dei dati relativi a Accessi, Giunzioni e Cippi chilometrici. La prima trasformazione (**Open_Coordinates.ktr**), visibile in figura 3.41, semplicemente apre il file KML **GIA_EL_STRADALE.kml** e lo salva in formato CSV nel file **GIA_EL_STRADALE.csv**.



Figura 3.41. Dettaglio della trasformazione Open_Coordinates.ktr

La seconda trasformazione (**Process_Coordinates.ktr**), inizia a manipolare i dati per renderli compatibili con l'ontologia. Lo schema del procedimento è mostrato in figura 3.42, mentre di seguito viene fornita la descrizione dettagliata dei singoli passi.

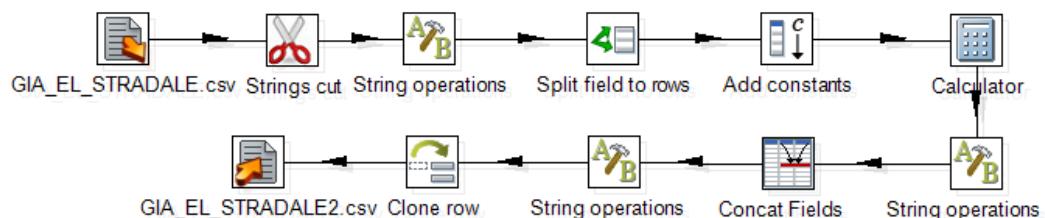


Figura 3.42. Dettaglio della trasformazione Process_Coordinates.ktr

Il primo passo è costituito dall'apertura del file temporaneo **GIA_EL_STRADALE.csv**. Subito dopo, tramite lo step **String Cut**, si estraе dal campo *ExtendedData* l'identificativo dell'elemento stradale. Come già visto nei casi precedenti, si effettua il trim del campo e poi si suddivide l'altro campo estratto dal file, ovvero *LineString* (diverso da *Point* visto in precedenza, dato che in questo caso si stanno lavorando file KML contenenti

dati non puntuali, bensì lineari). Il campo *LineString* contiene, al proprio interno, tutte le singole coordinate delle **Junction** che compongono l'elemento stradale (per informazioni si veda sempre il paragrafo 3.1.2.1). Tramite il comando **Split field to rows**, per ogni coppia di coordinate presenti all'interno del campo *LineString*, si crea una nuova riga contenente una copia delle informazioni dell'elemento stradale, con in più un nuovo campo, che contiene il numero progressivo della coordinata. In figura 3.43 sono mostrate le impostazioni dello step *Split field to rows*, nella quale si vede che le coordinate vengono salvate nel campo *COORD*, mentre il progressivo in *COORD_N*.

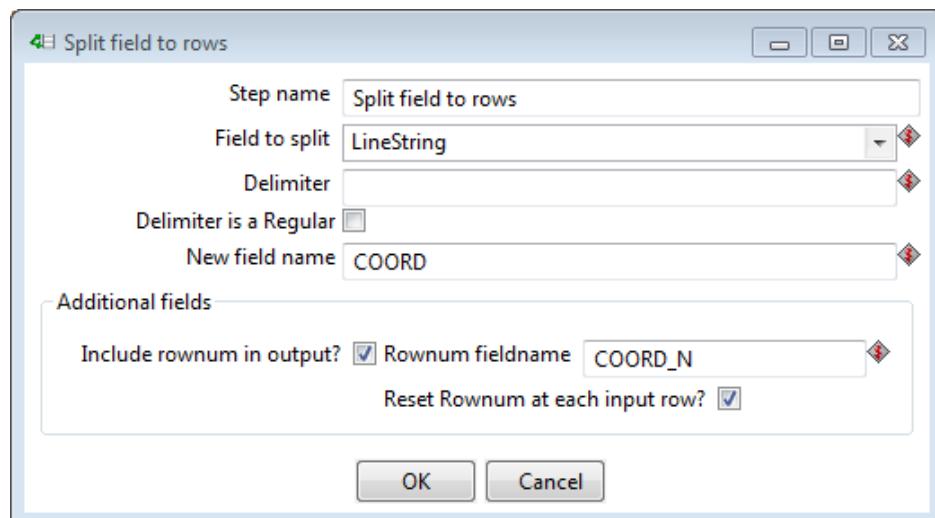


Figura 3.43. Impostazioni dello step Split field to Rows grazie al quale si crea una nuova riga per ogni coordinata dell'elemento stradale

Di seguito, in figura 3.44, invece, è mostrato un esempio dell'elaborazione.

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

Figura 3.44. Esempio di elaborazione dello step Split field to rows

I due step successivi (**Add constants** e **Calculator**) servono per creare un ulteriore colonna nella tabella dei dati, che contenga il numero progressivo della coordinata sottratto uno. Questo procedimento, come sarà maggiormente chiaro più avanti, serve per generare gli identificativi corretti da associare alle **Junction** di inizio e di fine **RoadLink**. Tramite **Add constants** si crea una colonna di supporto in cui è memorizzato costantemente il valore “1”, mentre tramite **Calculator**, si impone che un nuovo campo *COORD_N2* sia valorizzato come *COORD_N* - 1. Sul nuovo campo *COORD_N2* viene eseguito il trim, e successivamente tale valore viene concatenato con il campo *COD_ELE* (contenente l’identificativo dell’elemento stradale), tramite lo step **Concat Fields**, per generare l’identificativo univoco di quel particolare **RoadLink** nel campo aggiuntivo *COORD_ID*. Si esegue poi il trim su questo campo ed infine, prima di salvare i dati nel file temporaneo **GIA_EL_STRADALE2.csv**, si esegue lo step **Clone row**. Questa procedura provvede a replicare ogni riga dei dati. Questo risulta necessario per le elaborazioni della prossima trasformazione.

La terza ed ultima parte di elaborazione dei dati relativi alle coordinate degli elementi stradali è racchiusa nella trasformazione **Save_Coordinates.ktr**, mostrata in figura 3.45.

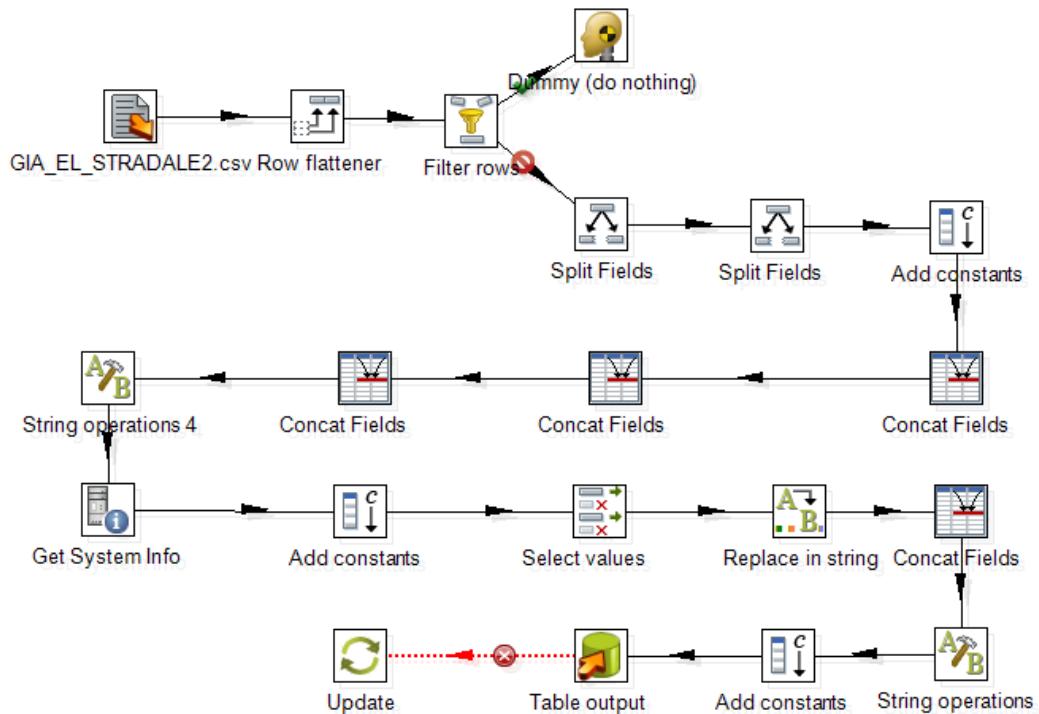


Figura 3.45. Dettaglio della trasformazione Save_Coordinates.ktr

All'inizio i dati vengono letti dal file temporaneo

GIA_EL_STRADALE2.csv. Subito dopo interviene la procedura di **Row flattener**. Tale procedura, sequenzialmente, preleva i dati di una particolare colonna e li traspone in altre colonne specificate dall'utente. In figura 3.46 è mostrata la finestra di impostazioni dello step ed un esempio di esecuzione. È qui che si nota l'utilità del precedente step **Clone row**, senza di esso, verrebbero perse la metà delle informazioni, visto che il passo di **Row flattener** ha come effetto collaterale la creazione di una sola riga, ogni due righe processate.

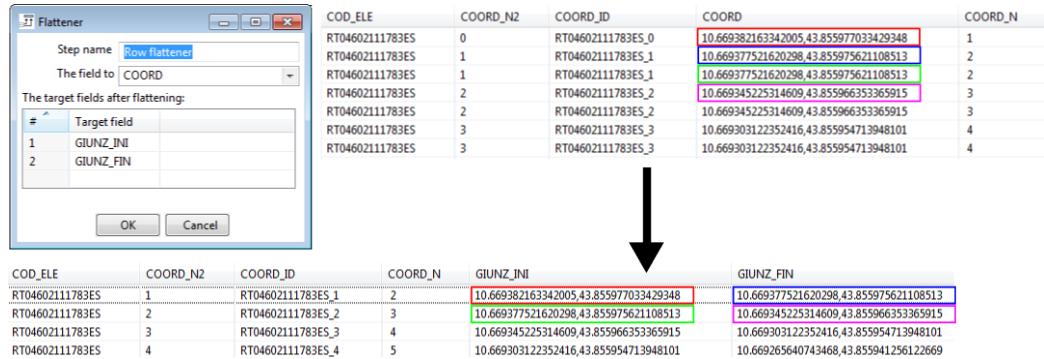


Figura 3.46. Impostazioni ed esempio dello step Row flattener

Utilizzando il filtro **Filter rows** si eliminano dal flusso tutte quelle righe al cui interno sono presenti coordinate facenti parte di due elementi stradali diversi. Per farlo, è sufficiente impostare come filtro la condizione *COORD_N2 = 0*. Tutte le righe per le quali la condizione è vera, finiscono nello step **Dummy (do nothing)**, che sostanzialmente le elimina dai dati. Se la condizione è falsa, invece, i record subiscono nuovamente due **Split fields**, che dividono le coordinate in latitudine e longitudine, per entrambe le colonne *GIUNZ_INI* e *GIUNZ_FIN*. A questo punto, si devono generare gli identificativi delle **Junction** evitando di creare duplicati inutili. Tramite lo step **Add constants** si crea una colonna di supporto che contiene la stringa *J*, e si concatenano i campi *COD_ELE* e *COORD_N* in un nuovo campo chiamato *COORD_ID2*. Per ogni riga si crea l'identificativo della Junction iniziale concatenando il valore *COORD_ID* con la colonna *J*, mentre si ottiene l'ID finale tramite concatenazione tra *COORD_ID2* ed il solito *J* grazie agli ulteriori due **Concat fields** in cascata. I nuovi identificativi, salvati nelle colonne *ID_INI* e *ID_FIN* vengono liberati dagli spazi e dai caratteri di ritorno a capo che erroneamente Kettle inserisce. Infine, si possono generare le date di aggiornamento e la colonna con la sorgente dei dati, prima di salvare, finalmente, tutti i dati nella tabella MySQL *tbl_coord_el_strad*.

Prima di procedere alla generazione delle triple RDF, come ultimo passaggio, si cancellano tutti i file temporanei generati durante le precedenti trasformazioni, con lo step **Delete temporary files**, come mostrato in figura

3.47.

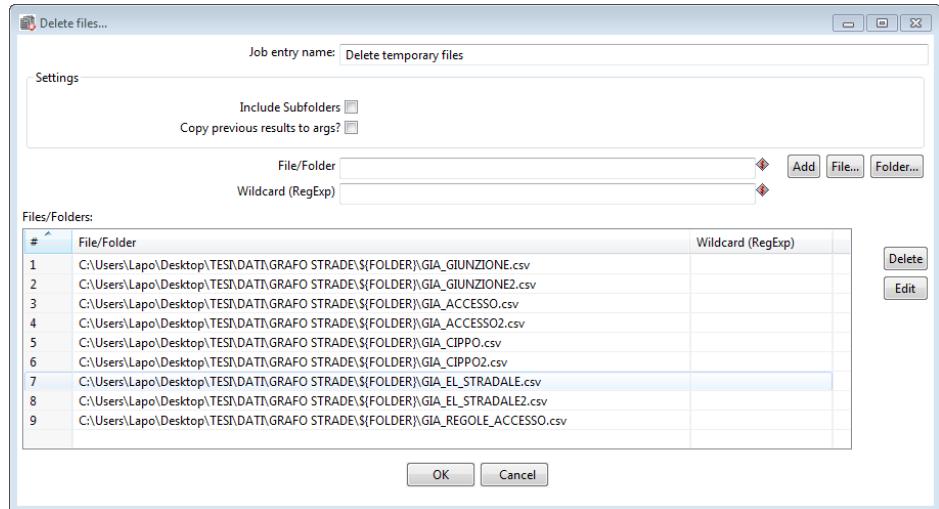


Figura 3.47. Impostazioni dello step Delete temporary files per la cancellazione dei file temporanei

Il job **MySQL_to_RDF.kjb**, a questo punto, prevede la generazione delle triple RDF a partire dai dati memorizzati nelle tabelle temporanee MySQL ed il successivo svuotamento di tali tabelle. In figura 3.48 è mostrato lo schema del job con i due step appena elencati.



Figura 3.48. Dettaglio del job MySQL_To_RDF.kjb

Lo step **Generate RDF** esegue uno script del software KARMA per la generazione delle triple RDF, come visibile dal dettaglio in figura 3.49. I dettagli di questa generazione sono approfonditi in sezione 3.2.2.

Fase di Ingestion

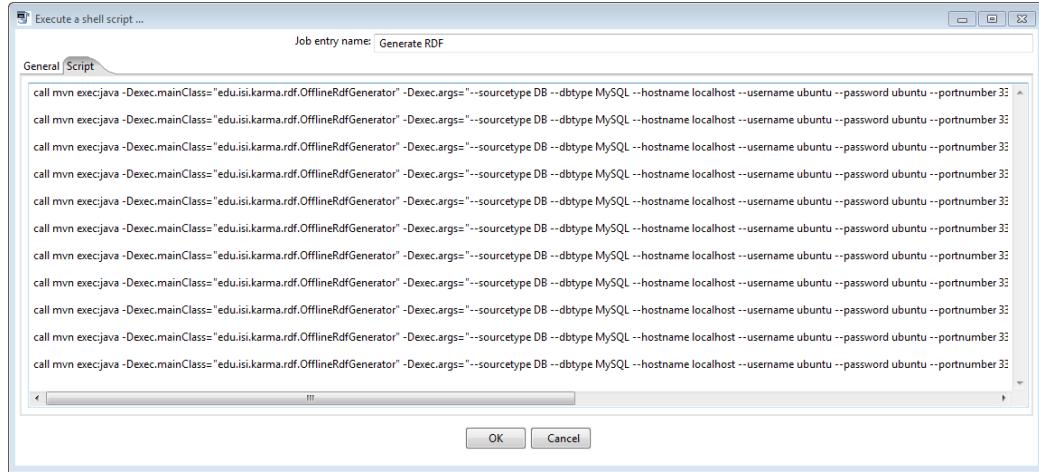


Figura 3.49. Impostazioni dello step Generate RDF per la generazione delle triple RDF a partire da tabelle MySQL

I file RDF sono generati all'interno della cartella *output* di default del software KARMA. Ciascun file è denominato come: **Nome della tabella MySQL di Origine + Nome della Provincia + .n3** (ad esempio **tbl estesa PROVINCIA FIRENZE.n3**).

Il secondo step (**Truncate MySQL Tables**), esegue il comando MySQL *Truncate Table* che svuota una tabella mantenendone inalterata la struttura. Tale comando viene eseguito sequenzialmente su tutte le tabelle fin qui generate, come mostrato in figura 3.50.

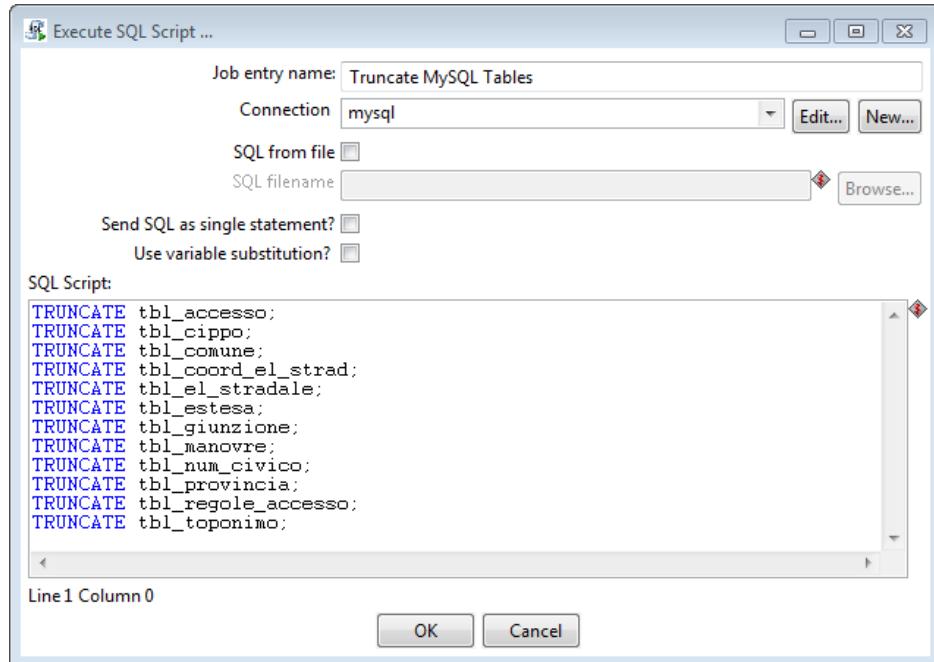


Figura 3.50. Impostazioni dello step Truncate MySQL Tables che svuota le tabelle MySQL di appoggio generate dalle trasformazioni precedenti

Una volta che le triple RDF sono state generate, poiché esse sono in numero molto elevato, è necessario effettuare un filtraggio per eliminare tutte le occorrenze di triple duplicate. In realtà OWLIM, lo store RDF, è in grado di riconoscere automaticamente le triple duplicate e di non immetterle due volte nel repository. Tuttavia, a causa del fatto che i dati relativi a differenti sezioni del grafo stradale sono memorizzati in *Context* diversi, come già spiegato in sezione 3.1.2.1, si rende necessario il filtraggio di tutte quelle triple che risulterebbero ridondanti. In questo momento, viene descritto il job **Filter_Duplicates.kjb**, il cui schema è esposto in figura 3.51.

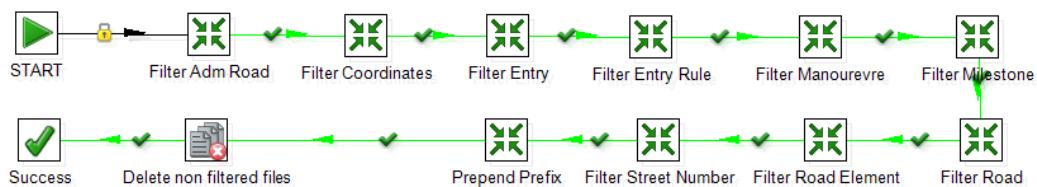


Figura 3.51. Dettaglio del job Filter_Duplicates.kjb

Come si nota dalla figura, il job di filtraggio è composto inizialmente da una sequenza di trasformazioni simili su ciascuna delle sezioni del grafo stradale che richiede il trattamento. A titolo di esempio, si mostra la prima trasformazione chiamata **Filter_Administrative_Road.ktr**. Le seguenti trasformazioni **Filter_Coordinates.ktr**, **Filter_Entry.ktr**, eccetera, sono perfettamente assimilabili alla prima e ne differiscono esclusivamente per piccole modifiche relative al tipo di oggetto elaborato. In figura 3.52, quindi, è mostrato lo schema della trasformazione di filtraggio dei dati di tipo **AdministrativeRoad**.

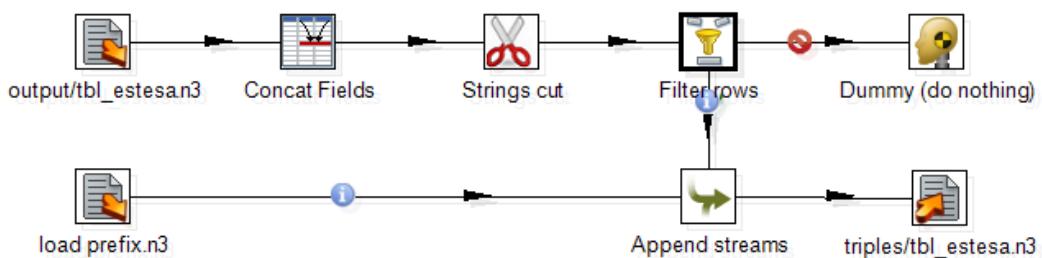


Figura 3.52. Dettaglio del job **Filter_Administrative_Road.kjb**

La trasformazione prende in input due file: il file con le triple RDF generato da KARMA ed un file denominato **prefix.n3** all'interno del quale sono memorizzati esplicitamente alcuni namespace di ontologie di uso comune (come ad esempio *xsd*, *dc* e *foaf*). Questi namespace, o *PREFIX* vengono inseriti in testa al file RDF affinché l'inserimento su repository avvenga senza problemi. OWLIM infatti, al momento del caricamento dei dati tramite console, richiede esplicitamente la definizione di ogni namespace usato nelle triple. Prima di unire il contenuto dei due file, tuttavia, il file con i dati RDF subisce alcune elaborazioni. Il primo step è **Concat fields**. Poiché il file sorgente è in formato n3, per differenziare le colonne *soggetto*, *predicato* ed *oggetto* proprie dell'RDF si usa il separatore *[SPAZIO]*; alcuni oggetti, però, al loro interno presentano proprio degli spazi (principalmente quando l'oggetto è di tipo *xsd:string* e non una URI). L'importazione di questi dati crea delle colonne fittizie, all'interno delle quali compaiono porzioni dell'oggetto. Tramite concatenazione di queste colonne fittizie, si riporta l'intero contenuto dell'oggetto nella sua colonna corretta. Dopo questa procedura si passa

al filtraggio vero e proprio. Nel caso delle Administrative Road, per ciascun record, la generazione RDF tramite KARMA produce le 8 triple mostrate in figura 3.53.

```

1 <http://www.disit.dinfo.unifi.it/SiiMobility/RT04800000001PA>
2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.disit.dinfo.unifi.it/SiiMobility#AdministrativeRoad> .
3 <http://www.disit.dinfo.unifi.it/SiiMobility/RT04800000001PA>
<http://purl.org/dc/terms/alternative> "SP1"^^xsd:string .
4 <http://www.disit.dinfo.unifi.it/SiiMobility/RT04800000001PA>
<http://purl.org/dc/terms/identifier> "RT04800000001PA"^^xsd:string .
5 <http://www.disit.dinfo.unifi.it/SiiMobility/RT04800000001PA>
<http://www.disit.dinfo.unifi.it/SiiMobility#name> "S.P. ARETINA PER
SAN DONATO (N. 1)"^^xsd:string .
6 <http://www.disit.dinfo.unifi.it/SiiMobility/048>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.disit.dinfo.unifi.it/SiiMobility#PA> .
7 <http://www.disit.dinfo.unifi.it/SiiMobility/RT04800000001PA>
<http://www.disit.dinfo.unifi.it/SiiMobility#ownerAuthority>
<http://www.disit.dinfo.unifi.it/SiiMobility/048> .
8 <http://www.disit.dinfo.unifi.it/SiiMobility/048>
<http://purl.org/dc/terms/identifier> "048"^^xsd:string .

```

Figura 3.53. Esempio di triple generate da KARMA relative a Administrative Road

Le triple dalla numero 1 alla numero 5 sono relative a DataProperties dell'oggetto `AdministrativeRoad`, la numero 7 esprime l'ObjectProperty che intercorre tra `AdministrativeRoad` e `PA`, ovvero `ownerAuthority`. Le triple numero 6 e numero 8, invece, sono proprie della classe `PA`, ma vengono lo stesso generate da KARMA, una volta che si è definita la proprietà `ownerAuthority`. Queste due triple devono essere scartate, perché le stesse triple vengono create, anche dalla generazione delle triple relative alle Province. Per questo motivo, tramite **Strings cut** si estraggono gli ultimi due caratteri della colonna soggetto e nello step successivo, cioè **Filter rows**, si confrontano con la stringa `PA`: se il confronto restituisce `true`, allora si passa alla fase di **Append streams** che, come già detto, unisce i due flussi di file (RDF e prefix), altrimenti, se il confronto restituisce `false`, quel record finisce nello step di **Dummy (Do nothing)** e di conseguenza viene scartato. Dopo

il filtraggio, i dati vengono nuovamente salvati in formato *.n3*, ma si fornisce un percorso di salvataggio diverso, ovvero la cartella in cui sono temporaneamente salvati tutti i file RDF prima che vengano inseriti nel repository semantico.

Le altre trasformazioni di filtraggio seguono lo stesso paradigma. Ovviamente, per ogni tipologia di dato si eseguono operazioni e filtri leggermente diversi, ma il pattern non cambia.

La trasformazione **Prepend _ Prefix.ktr**, per la quale non viene mostrata alcuna figura, viene eseguita in parallelo sulle tipologie di dato che non richiedono filtri di duplicati (Giunzioni, Comuni e Provincia). Semplicemente, come nelle trasformazioni precedenti, si occupa di inserire, in testa al file RDF le definizioni dei namespaces e di spostare il file *.n3* nella cartella di destinazione corretta.

L'ultimo step del job **Filter _ Duplicates.kjb** è costituito dall'operazione di **Delete non filtered files**, che rimuove tutti i file originale a partire dai quali è stato effettuato il filtraggio e che quindi non risultano più utili al lavoro.

Altro job molto importante è **Split _ Big _ Files.kjb**. La generazione RDF di triple relative a Elementi Stradali, Accessi e Numeri Civici su province molto grandi (quali ad esempio *Firenze* e *Siena*), crea dei file di diverse centinaia di MByte che comportano lunghi periodi di caricamento su OWLIM. Per questo motivo, nel caso in cui tali file risultino eccessivamente grandi, si provvede a suddividerli in file più piccoli. Il job **Split _ Big _ Files.kjb** esegue questo compito, la sequenza dei suoi passi è mostrata in figura 3.54.

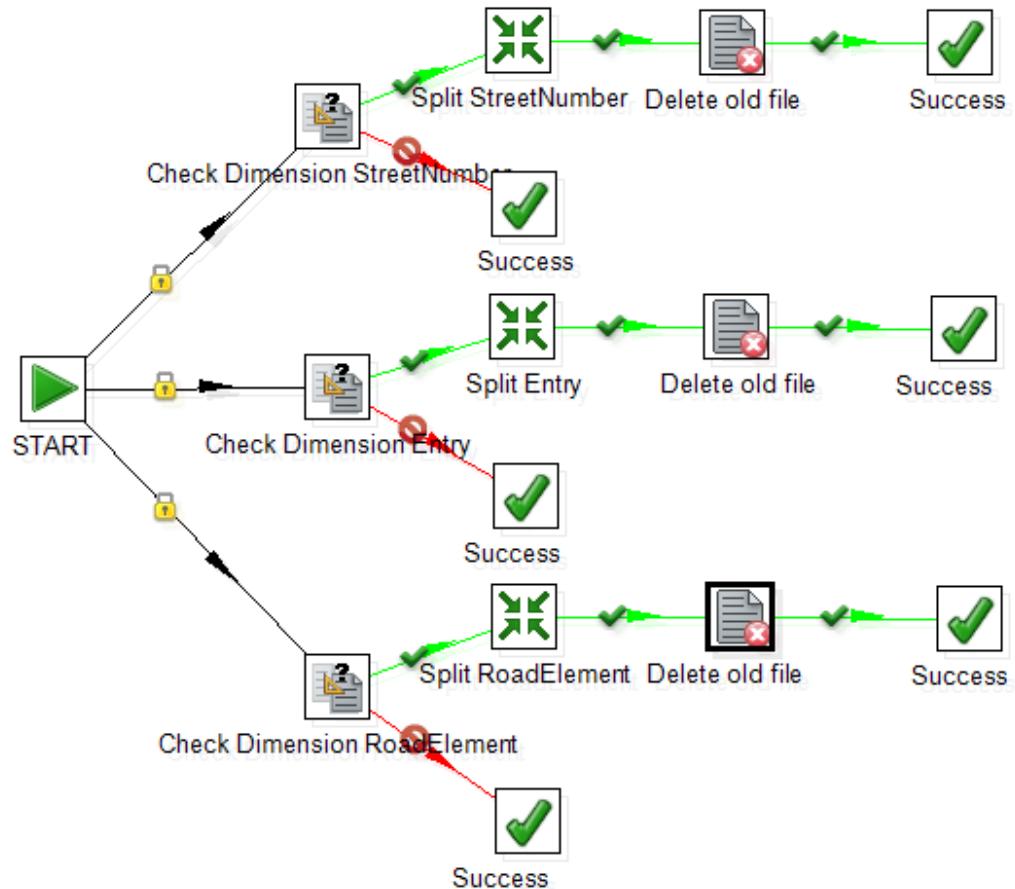


Figura 3.54. Dettaglio del job Split_Big_Files.kjb

Su ciascuno dei file corrispondente al tipo di dati appena citato, viene effettuato un controllo sulla dimensione del file tramite lo step **Check Dimension [TIPO]**. Se la dimensione del file è inferiore ai 100MB, il file non viene modificato, altrimenti viene passato alla corrispondente trasformazione **Split [TIPO]**. Ciascuna di queste trasformazioni apre il file, e lo risalva settando nelle impostazioni di suddividerlo ogni 400.000 righe e di rinominarlo, aggiungendo la stringa *_split* alla fine del nome del file più un numero progressivo.

Dopo la suddivisione, il file originale di grandi dimensioni viene cancellato tramite l'operazione **Delete original file**.

A questo punto, tutti i file in formato RDF sono memorizzati in un'unica cartella e sono pronti ad essere, finalmente, caricati sull'RDF store. L'upload di tali file è realizzato utilizzando la console di OWLIM, la quale è un eseguibile batch presente all'interno delle cartelle del software. I dettagli relativi all'inserimento di dati sul repository tramite console sono spiegati con precisione in sezione 3.2.3. Una volta aperta la console è possibile digitare i comandi di interesse quali quelli di connessione al repository e di caricamento dei dati. Per passare questi comandi è necessario creare un file testuale, e passarlo allo script tramite operatore "<". Il file testuale viene creato dallo step **Write instructions** nella stessa cartella del binario *console.bat*. Al suo interno sono presenti tutte le istruzioni per il caricamento dei dati nel repository, come mostrato in figura 3.55.

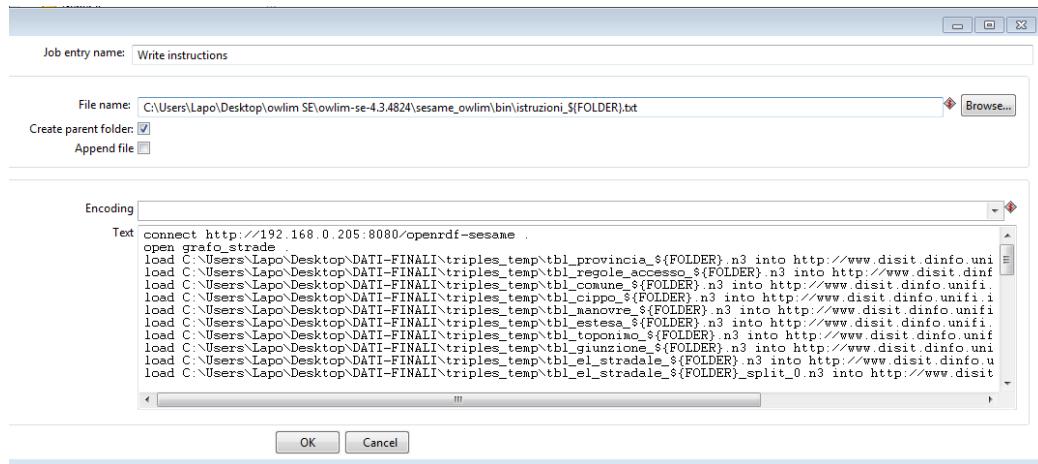


Figura 3.55. Impostazioni dello step **Write Instructions** per la generazione del file contenente le istruzioni per il caricamento dei file sul repository OWLIM

L'ultimo step del job principale **main_job.kjb** è rappresentato dal richiamo del semplice script di apertura della console, seguito dal file contenente le istruzioni. Tramite lo step **Load Triples** si esegue il comando mostrato in figura 3.56.

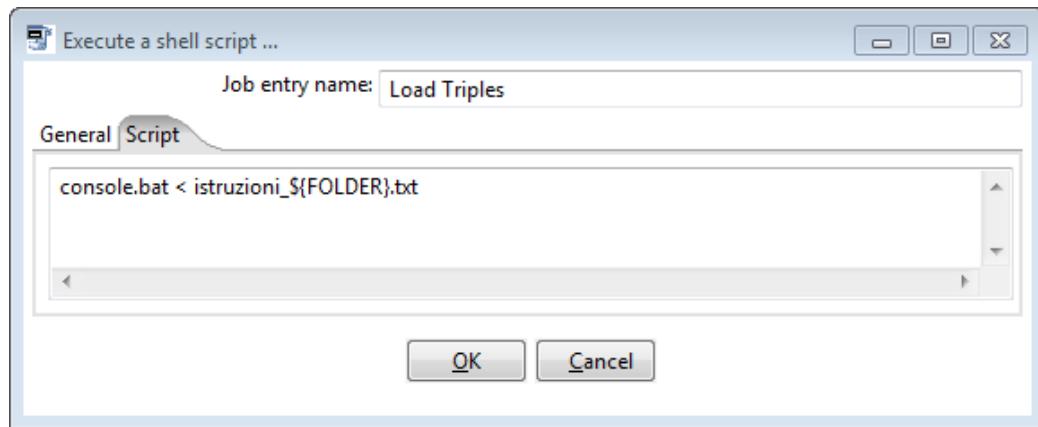


Figura 3.56. Impostazioni dello step Load Triples per il caricamento definitivo dei dati su repository RDF

Per quanto riguarda i dettagli implementativi delle trasformazioni Kettle relative alle altre macrosezioni dell'ontologia **SmartCity Ontology** si rimanda a [1] e [2].

3.2.2 Definizione modello R2RML e Generazione Triple RDF

Nel capitolo precedente, è stata descritta l'esecuzione del task principale di elaborazione dei dati del grafo strade, che conduce alla loro conversione in formato RDF ed al loro caricamento sul repository. In questa sezione vengono descritte in dettaglio le procedure di creazione del modello R2RML che converte dati memorizzati su tabelle MySQL in RDF, e la procedura per avviare tale conversione.

Per entrambi questi compiti si fa ricorso al software Karma, del quale, in figura 3.57, è mostrata l'interfaccia web iniziale. I principali dettagli tecnici ed implementativi ed alcuni esempi di utilizzo sono descritti in [6] e [7]. Una descrizione sommaria del software KARMA è fornita in Appendice B.3.

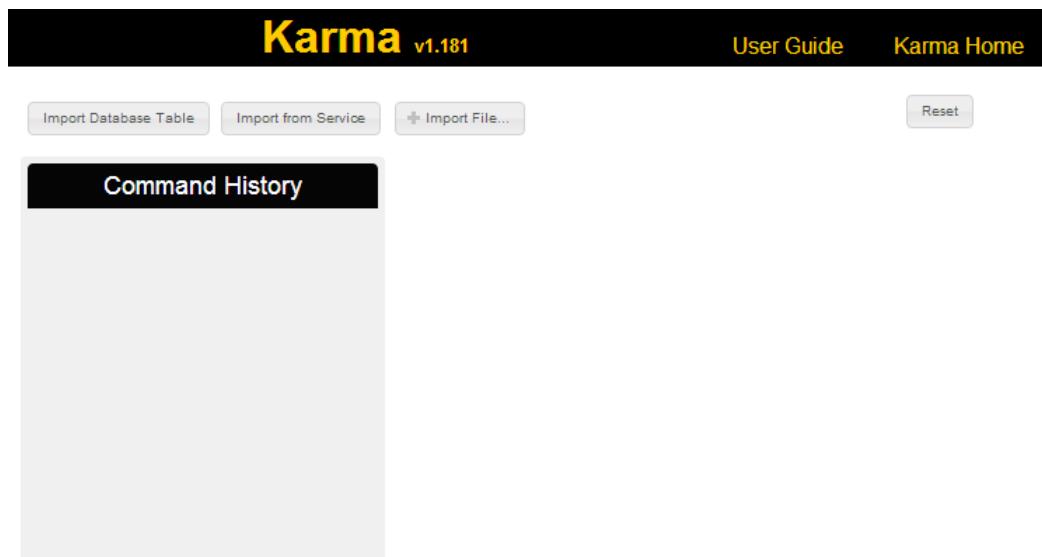


Figura 3.57. Interfaccia web iniziale del software KARMA

Per questo lavoro, si è utilizzato il seguente approccio: per ognuna delle classi di dati a disposizione del grafo strade, si sono estratte delle porzioni significative di esempi. A partire da questi dataset ridotti, si è definito manualmente, tramite l’interfaccia appena mostrata, il modello R2RML di mappatura tra i dati MySQL elaborati da Kettle e gli stessi dati in formato RDF adeguati all’ontologia **SmartCity Ontology**. Una volta generati questi modelli, essi sono stati poi utilizzati per la generazione batch delle triple RDF su tutti i dati a disposizione.

Di seguito viene mostrato un esempio di utilizzo di KARMA per la generazione del modello R2RML sui dati relativi alle Manovre stradali. Prima di avviare il software, è possibile settare alcune impostazioni di base, quali ad esempio il namespace di default degli oggetti generati, il percorso di default del database MySQL da cui attingere i dati, ma soprattutto si può specificare a KARMA quali ontologie di base deve utilizzare per la mappatura dei valori. Per farlo, è necessario inserire in una cartella denominata *preloaded-ontologies*, nella directory principale di KARMA, tutte le ontologie ed i vocabolari di interesse (quindi *SmartCity Ontology*, *FOAF*, *Time*, eccetera) in formato OWL o RDF.

Una volta terminate queste operazioni preliminari, è possibile lanciare il

software. Tramite browser si accede alla schermata iniziale, ed a questo punto è possibile importare all'interno del programma i dati MySQL di interesse. Per farlo, si clicca sul pulsante “*Import Database Table*” in alto a sinistra (come descritto in sezione B.3, KARMA permette l'importazione di dati a partire da molteplici sorgenti differenti, come ad esempio file *CSV*, *JSON*, *XML* oppure database relazionali *SQLServer*, *Oracle*, eccetera). Dopo il click, si aprirà una finestra popup in cui è possibile inserire i parametri di connessione al database, come mostrato in figura 3.58.



Figura 3.58. Finestra di impostazione di parametri KARMA per l'importazione di dati da database

Cliccando su “*OK*”, si apre l'elenco comprendente tutte le tabelle salvate in quel particolare database. Selezionando la tabella di interesse, nel caso di esempio **tbl_manovre**, e cliccando su “*Import*”, il software effettuerà l'importazione dei dati di quella tabella e li mostrerà in una tabella HTML all'interno dell'interfaccia, come mostrato in figura 3.59.

tbl_manovre								
	ID_MAN	FEATTYP	VIA_GINZ	COD_ELE	COD_ELE_1	COD_ELE_2	ISO_DATE	ORG
14380010010017500	Manovra proibita calcolata	RT04600102557GZ	RT04600114754ES	RT04600114753ES			2014-03-11T10:46:40.449+01:00	Osservatorio Trasporti Regione Toscana
14380010010031000	Manovra proibita calcolata	RT04600209432GZ	RT04600210943ES	RT04600210935ES			2014-03-11T10:46:40.449+01:00	Osservatorio Trasporti Regione Toscana
14380010010069800	Manovra proibita calcolata	RT04603307217GZ	RT04603306639ES	RT04603306647ES			2014-03-11T10:46:40.449+01:00	Osservatorio Trasporti Regione Toscana
14380010010085800	Manovra proibita calcolata	RT04603306714GZ	RT046033068356ES	RT04603306810ES			2014-03-11T10:46:40.449+01:00	Osservatorio Trasporti Regione Toscana
14380010010087400	Manovra proibita calcolata	RT04602404859GZ	RT04602404643ES	RT04602404677ES			2014-03-11T10:46:40.449+01:00	Osservatorio Trasporti Regione Toscana
14380010010092200	Manovra proibita calcolata	RT0460170570GZ	RT04601715081ES	RT04601713565ES			2014-03-11T10:46:40.449+01:00	Osservatorio Trasporti Regione Toscana

Figura 3.59. Importazione dei dati della tabella **tbl_manovre** all'interno dell'interfaccia web di KARMA

Per iniziare a mappare le colonne, è necessario cliccare su “*Show Model*” nel menu a tendina in alto a sinistra. A questo punto, sopra ogni colonna

compare un piccolo cerchio rosso. Cliccando su di esso, sarà possibile impostare il collegamento tra la colonna e le corrispondenti classi e proprietà dell'ontologia. In particolare, come mostrato in figura 3.60, si apre un piccolo popup, all'interno del quale è possibile impostare diversi parametri.

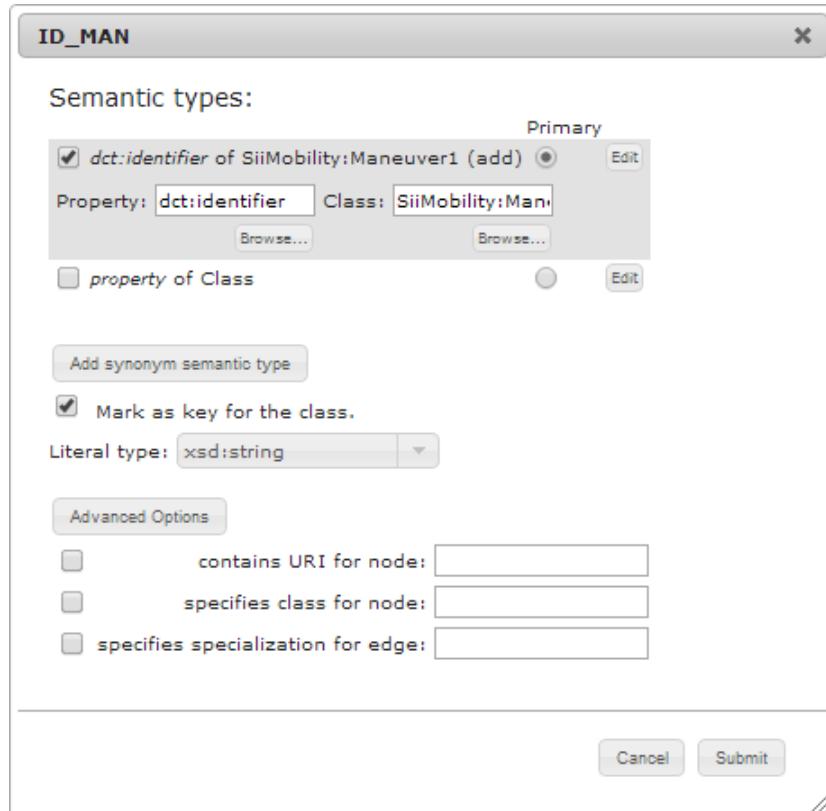


Figura 3.60. Impostazioni di mappatura della colonna ID_MAN della tabella tbl_manovre

Il campo *ID_MAN* mostrato nell'esempio contiene l'identificativo univoco regionale della manovra. All'interno dell'ontologia **SmartCity Ontology**, tale campo è espresso come DataProperty *dc:identifier* della classe **Sii-Mobility:Maneuver**. In figura si vede come viene effettuata questa associazione. Inoltre, poichè l'identificativo univoco è utilizzato per creare la URI della risorsa di tipo **Maneuver**, si deve selezionare la casella “*Mark as key for the class*”. Per ogni colonna, inoltre, è possibile impostare esplicitamente il tipo primitivo di dato, ad esempio “*xsd:string*”, “*xsd:integer*” o

“xsd:dateTime”, semplicemente selezionando il valore corretto nel menu a tendina etichettato come “*Literal Type*”. Dopo aver selezionato la corrispondenza del dato all’interno dell’interfaccia, essa viene mostrata visivamente attraverso un box grigio ed una freccia, come visibile in figura 3.61.

The screenshot shows the Karma interface with the following details:

- Top Bar:** Karma v1.181
- Toolbar:** Import Database Table, Import from Service, + Import File...
- Left Panel (Command History):**
 - Import Database Table: tbl_manovre imported
 - Show Model: tbl_manovre
 - Apply History:
 - Unassign Semantic Type: COD_ELE
 - Unassign Semantic Type: COD_ELE_1
 - Unassign Semantic Type: COD_ELE_2
 - Change Links: SiiMobility:concerning
- Right Panel (tbl_manovre Model):**
 - Class:** Maneuver1
 - Identifier:** identifier*
 - Table:** ID_MAN, FEATTYP, VIA_GNZ
 - Data Row:**

14380010010017500	Manovra proibita calcolata	RT04600102557GZ
-------------------	----------------------------	-----------------

Figura 3.61. Associazione colonna ID_MAN alla proprietà dc:identifier della classe Sii-Mobility:Maneuver

In realtà, tramite KARMA è possibile mappare dati in maniera molto complessa, ad esempio una colonna può essere associata a due o più diverse classi, più colonne con lo stesso tipo di dato possono essere associate a oggetti diversi della stessa classe, eccetera. Si rimanda nuovamente al sito internet del progetto¹⁰ o alla letteratura sull’argomento per dettagli aggiuntivi.

Al termine della mappatura di ogni colonna nelle corrette proprietà e classi dell’ontologia, è necessario definire i collegamenti tra tali entità. È quindi necessario stabilire correttamente le ObjectProperties che intercorrono tra i vari oggetti. KARMA, autonomamente, tenta di riconciliare automaticamente questi collegamenti, basandosi sulle ontologie precaricate. Talvolta, tuttavia, è necessario intervenire manualmente per perfezionare i collegamenti. Ad esempio, il campo VIA_GNZ dei dati MySQL corrisponde all’identificativo della giunzione, sulla quale è definita la manovra. Nell’ontologia, tale

¹⁰Karma - Data Integration Tool - <http://www.isi.edu/integration/karma/>

proprietà è *dc:identifier* sulla classe di riferimento **Sii-Mobility:Node**. Nell’interfaccia di KARMA compariranno quindi i due box contenenti le diverse classi ed eventualmente una freccia di associazione, come mostrato in figura 3.62.

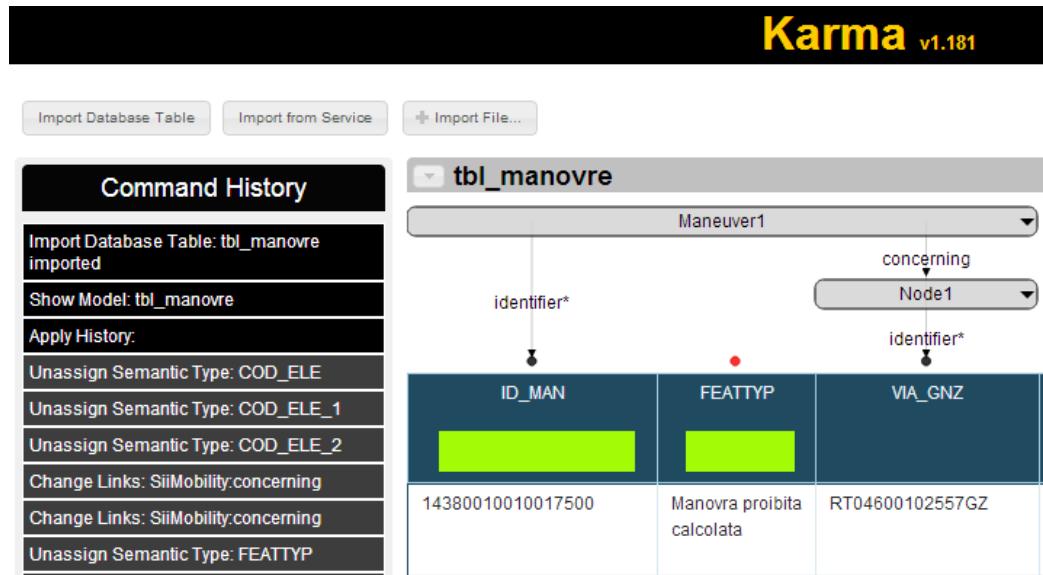


Figura 3.62. Associazione colonna VIA_GNZ alla proprietà *dc:identifier* della classe Sii-Mobility:Node e creazione del collegamento tra le classi

Tra le classi **Node** e **Maneuver** è definita la ObjectProperty *concerning*. Cliccando su uno dei due box o direttamente sulla freccia che li collega è possibile impostare il collegamento corretto, tramite un piccolo popup, come mostrato in figura 3.63.

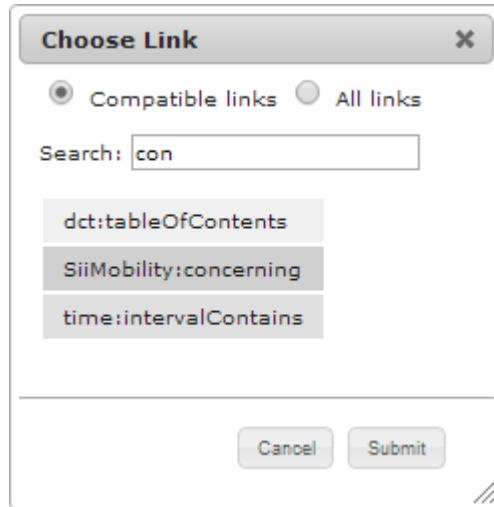
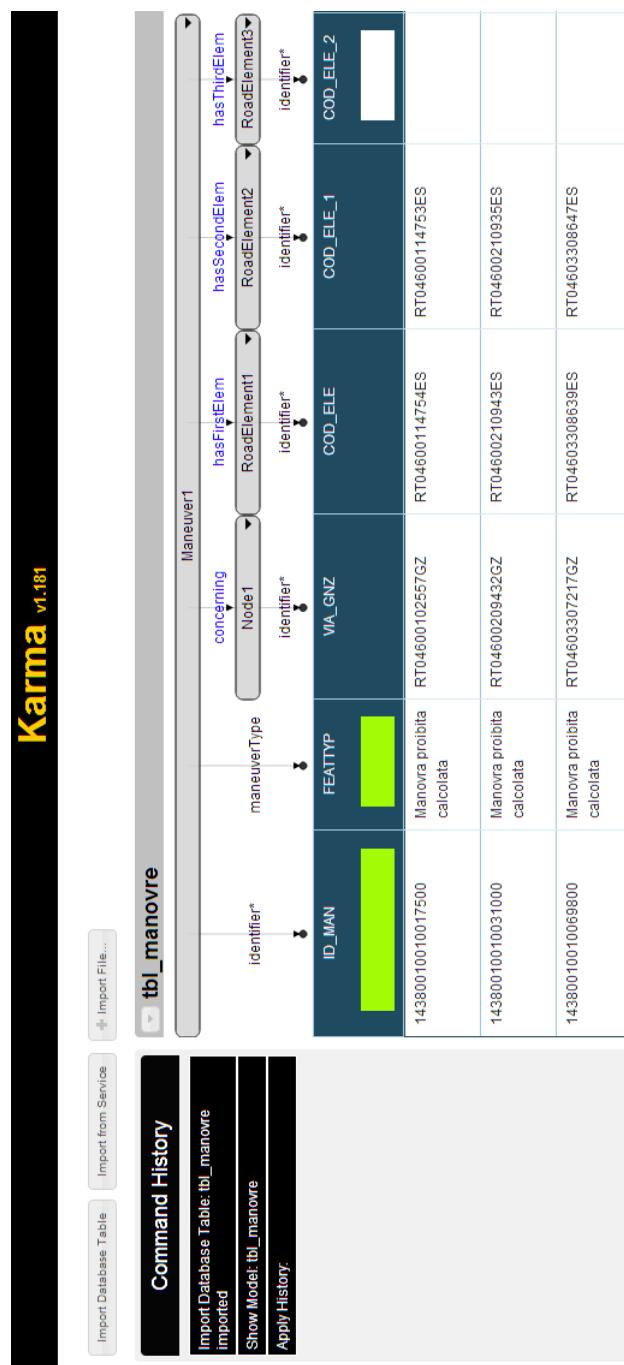


Figura 3.63. Definizione dell'ObjectProperty Sii-Mobility:concerning tra le classi Node e Maneuver

Una volta terminate le operazioni di specifica dei tipi semanticici e di collegamento tra di essi (si veda la figura 3.64), è possibile esportare direttamente i dati in RDF, oppure esportare il modello R2RML per poter successivamente generare triple RDF, attraverso la modalità batch.

Figura 3.64. Esempio di mappatura completa della tabella `tbl_manovre`

In entrambi i casi è necessario cliccare la corrispondente voce all'interno del menu a tendina in alto a sinistra. I file generati sono recuperabili all'in-

terno delle cartelle di KARMA. Un esempio di file R2RML è visibile nella figura 3.65 seguente.

```

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix dcam: <http://purl.org/dc/dcam/> .
@prefix p1: <http://www.pms.ifi.uni-muenchen.de/OTN#Traditional\_Woodland> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84\_pos#> .
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix km-dev: <http://isi.edu/integration/karma/dev#> .

_:node188mise4bx1 a km-dev:R2RMLMapping ;
  km-dev:sourceName "tbl_manovre" ;
  km-dev:modelPublicationTime "1383737473164"^^xsd:long ;
  km-dev:hasWorksheetHistory """"[

{
  \"commandName\": \"SetSemanticTypeCommand\",
  \"inputParameters\": [
    {
      \"name\": \"metaPropertyName\",
      \"type\": \"other\"
    },
    {
      \"name\": \"hNodeId\",
      \"type\": \"hNodeId\",
      \"value\": [{\"columnName\": \"ID_MAN\"}]
    },
    {
      \"name\": \"SemanticTypesArray\",
      \"type\": \"other\",
      \"value\": [
        {
          \"Domain\": \"http://www.disit.dinfo.unifi.it/SiiMobility#Maneuver\",
          \"FullType\": \"http://purl.org/dc/terms/identifier\",
          \"isPrimary\": true
        }
      ]
    }
  ]
}

```

Figura 3.65. Esempio di file R2RML generato da KARMA

Una volta generato il modello R2RML, è possibile automatizzare il processo di generazione delle triple, come già visto nella sezione 3.2.1 di Estrazione e Manipolazione dei Dati. KARMA mette a disposizione alcuni script per

la generazione di triple RDF a partire da database relazionali. Un esempio dello script è mostrato in figura 3.66.

```
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"
```

Figura 3.66. Esempio di script per la generazione batch delle triple RDF per la tabella MySQL *tbl_accesso*

Dalla figura si comprendono quali sono i parametri da passare allo script racchiusi nel macroparametro *-Dexec.args*:

- **-sourcetype**: il tipo di sorgente dei dati. Può assumere i valori *DB* (come nel caso di esempio), *CSV*, *JSON*, eccetera.
- **-dbtype**: obbligatorio se *-sourcetype* è impostato a *DB*. Rappresenta il tipo di DB dal quale si prelevano i dati. Può assumere i seguenti valori: *MySQL*, *Oracle*, *SQLServer*, eccetera.
- **-hostname**, **-dbport**, **-username**, **-password**: contengono i parametri per la connessione al database (nome dell'host su cui è ospitato il database, nome del database, nome utente e password per l'autenticazione).
- **-tablename**: specifica il nome della tabella da cui sono letti i dati da convertire.
- **-modelfilepath**: indica il percorso attraverso il quale si risale al modello R2RML, generato come descritto nel paragrafo precedente. Può indicare un percorso relativo (come nel caso di esempio, rispetto alla directory principale del software KARMA), oppure un percorso assoluto.

- **-outputfilepath:** il percorso ed il nome di destinazione del file di output. Anche in questo caso è possibile fornire percorsi relativi o assoluti.

Una volta lanciato lo script, KARMA provvede a generare le triple seguendo il modello R2RML generato in precedenza. Al termine, è possibile analizzare il risultato ottenuto come mostrato in figura 3.67.

```

1 <http://www.disit.dinfo.unifi.it/SiiMobility/RT04900707303ES>
2 <http://www.disit.dinfo.unifi.it/SiiMobility#elementClass> "extraurbana
principale"^^xsd:string .
3 <http://www.disit.dinfo.unifi.it/SiiMobility/RT04900707303ES>
4 <http://www.disit.dinfo.unifi.it/SiiMobility#composition> "carreggiata
unica"^^xsd:string .
5 <http://www.disit.dinfo.unifi.it/SiiMobility/RT04900707303ES>
6 <http://www.purl.org/dc/terms/identifier> "RT04900707303ES"^^xsd:string .
7 <http://www.disit.dinfo.unifi.it/SiiMobility/RT04900707303ES>
8 <http://www.disit.dinfo.unifi.it/SiiMobility#elementType> "di tronco
carreggiata"^^xsd:string .
<http://www.disit.dinfo.unifi.it/SiiMobility/RT04900707303ES>
<http://www.disit.dinfo.unifi.it/SiiMobility#isPartOf>
<http://www.disit.dinfo.unifi.it/SiiMobility/RT04900718705TO> .
<http://www.disit.dinfo.unifi.it/SiiMobility/RT04900707303ES>
<http://www.disit.dinfo.unifi.it/SiiMobility#starts>
<http://www.disit.dinfo.unifi.it/SiiMobility/RT04900704420GZ> .

```

Figura 3.67. Esempio di triple RDF generate in modalità batch da KARMA dalla tabella `tbl_el_stradale`

La procedura di generazione dei dati relativi alle altre sezioni del grafo strade ricalca quella appena descritta. Ovviamente, ogni sezione avrà le proprie caratteristiche e dovrà prevedere trattamenti specifici per alcuni tipi di informazioni, ma il paradigma di creazione del modello R2RML segue sempre gli stessi passi.

I dettagli sulla generazione dei modelli R2RML relativi alle altre macro-sezioni del progetto Sii-Mobility sono reperibili in [1] e [2].

3.2.3 Creazione Repository Semantico ed Inserimento delle Triple RDF

Per utilizzare l'enorme mole di triple prodotte dalla conversione in RDF dei dati relativi all'intero grafo stradale della Regione Toscana, si ha bisogno di un framework RDF che permetta il salvataggio continuativo dei dati mantenendo la scalabilità e l'usabilità, e che permetta di analizzare ed interrogare l'archivio dei dati in modo veloce, affidabile e personalizzabile.

Il miglior prodotto opensource che soddisfa tutti i requisiti è OpenRDF Sesame. Su tale framework, è possibile *installare* il software OWLIM, il quale permette di generare un repository RDF estremamente performante, scalabile e sul quale è possibile applicare dei ragionamenti semantici, quindi estendere automaticamente la base di conoscenza tramite meccanismi di inferenza basati su logiche del primo ordine.

OWLIM mette a disposizione tutti gli strumenti necessari per creare, gestire ed interrogare tali repository, principalmente attraverso due modalità: tramite interfaccia web, oppure tramite applicazione di tipo console eseguibile da terminale. Inoltre, Sesame mette a disposizione una serie di API Java per accedere e manipolare gli store RDF tramite applicazioni JAVA create ad hoc dall'utente.

Per quanto riguarda la creazione del repository semantico su cui vengono caricati i dati del grafo strade, si è fatto ricorso all'interfaccia web di OWLIM, per poter meglio settare alcuni parametri, quali ad esempio le dimensioni degli indici, il set di regole per l'inferenza OWL, la URL del namespace di default, eccetera. Per determinare quali fossero i parametri corretti, è stato effettuato un lungo studio e sono state condotte numerose prove di creazione e caricamento dei dati. In figura 3.68 è mostrata l'interfaccia OWLIM che permette di creare un nuovo repository.

The screenshot shows the OWLIM Workbench interface. On the left is a sidebar with the following menu items:

- Sesame server
- Repositories** (selected)
- New repository
- Delete repository
- Explore
- Summary
- Namespaces
- Contexts
- Types
- Explore
- Query
- Export
- Modify
- SPARQL Update
- Add
- Remove
- Clear
- System
- Information

The main area is titled "New Repository". It contains various configuration fields:

- Type: OWLIM-SE
- ID: siimobility
- Title: siimobility
- Storage folder: storage
- License file (leave blank for evaluation): (empty)
- Ruleset: OWL2-RL (optimized)
- Base URL: http://www.disit.dinfo.unifi.it/SiiMobility
- Imported RDF files(';' delimited): (empty)
- Default namespaces for imports(';' delimited): (empty)
- Entity index size: 2000000
- Total cache memory (min 20m): 3900m
- Main index memory (min 20m): 2700m
- Use predicate indices: True
- Predicate index memory (min 20m): 300m
- Full-text search memory (min 20m): 0
- Full-text search indexing policy: Never
- Full-text search literals only: True
- Use PCSOT index: True
- Use PTSOC index: True
- Cache literal language tags: False

At the bottom right are two buttons: "Create" and "Cancel".

Figura 3.68. Interfaccia web del software OWLIM per la generazione di un nuovo repository

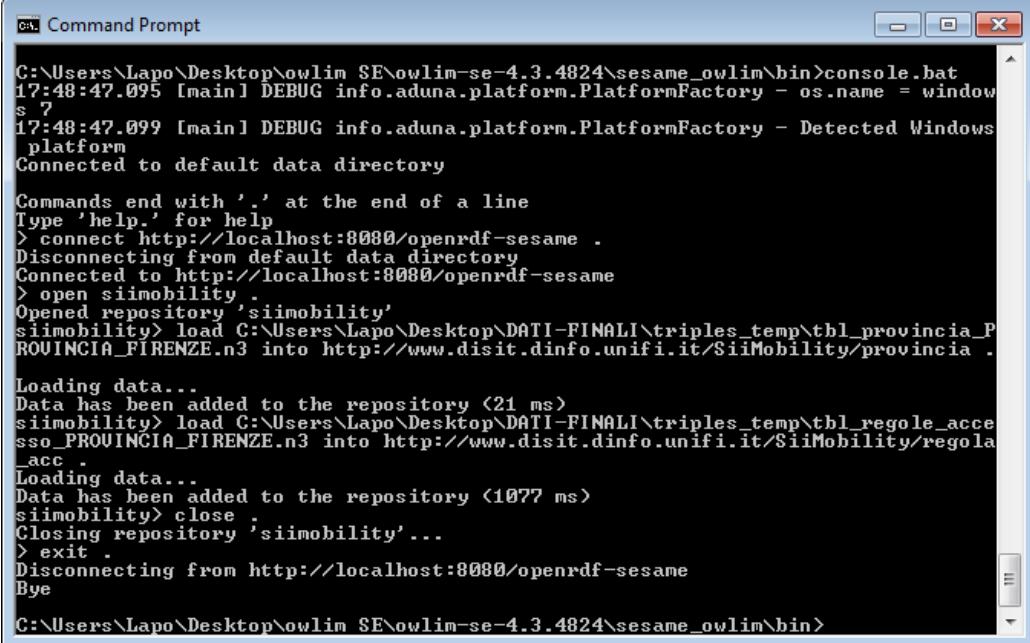
Per quanto riguarda l'inserimento delle triple RDF sul repository, pur rimanendo la possibilità di inserirle manualmente tramite interfaccia web, si è preferito utilizzare la console che OWLIM mette a disposizione tramite terminale. In questo modo, è risultato possibile effettuare il caricamento automatico tramite Pentaho Kettle, come già accennato in sezione 3.2.1. La console è un eseguibile *.bat* (nella versione per Windows, ma all'interno della cartella di OWLIM è presente anche il file *console.sh*, eseguibile da terminale su sistemi Linux), presente all'interno di una cartella della distribuzione di OWLIM e permette alcune operazioni basilari, quali ad esempio:

- **connect**: permette di collegarsi ad un repository locale o remoto spe-

cificandone l'hostname o l'indirizzo.

- **create**: crea un nuovo repository.
- **open**: apre un particolare repository in base al nome dello stesso.
- **load**: carica dati sul repository attualmente aperto. Richiede il percorso del file sorgente e, optionalmente, il *Context* cui associare i dati. Una spiegazione delle motivazioni che possono portare all'uso del *Context* è già stata fornita 3.1.2.1.
- **sparql**: esegue una query SPARQL, che ovviamente deve essere passata come parametro dell'istruzione.
- **close**: chiude la connessione con il repository attualmente in uso.
- **exit**: interrompe la connessione con il repository e chiude il programma.

In figura 3.69, è mostrata l'esecuzione di alcune istruzioni all'interno di un terminale.



```
C:\Users\Lapo\Desktop\owlim SE\owlim-se-4.3.4824\sesame_owlim\bin>console.bat
17:48:47.095 [main] DEBUG info.aduna.platform.PlatformFactory - os.name = windows
s ?
17:48:47.099 [main] DEBUG info.aduna.platform.PlatformFactory - Detected Windows
platform
Connected to default data directory

Commands end with '.' at the end of a line
Type 'help.' for help
> connect http://localhost:8080/openrdf-sesame .
Disconnecting from default data directory
Connected to http://localhost:8080/openrdf-sesame
> open siimobility .
Opened repository 'siimobility'
siimobility> load C:\Users\Lapo\Desktop\DATI-FINALI\triples_temp\tbl_provincia_P
ROVINCIA_FIRENZE.n3 into http://www.disit.dinfo.unifi.it/SiiMobility/provincia .
Loading data...
Data has been added to the repository <21 ms>
siimobility> load C:\Users\Lapo\Desktop\DATI-FINALI\triples_temp\tbl_regole_acce
sso_PROVINCIA_FIRENZE.n3 into http://www.disit.dinfo.unifi.it/SiiMobility/regola
_acc .
Loading data...
Data has been added to the repository <1077 ms>
siimobility> close
Closing repository 'siimobility'...
> exit .
Disconnecting from http://localhost:8080/openrdf-sesame
Bye

C:\Users\Lapo\Desktop\owlim SE\owlim-se-4.3.4824\sesame_owlim\bin>
```

Figura 3.69. Esempio di esecuzione della console di OWLIM

Solitamente, passando ad un terminale una sequenza di istruzioni separate dal ritorno a capo, esse vengono eseguite sequenzialmente. Tuttavia, dato che la console di OWLIM è una applicazione a sé stante, questo non è possibile. Per automatizzare il processo, quindi per poter eseguire automaticamente alcune delle istruzioni appena elencate, è necessario ridefinire lo standard input per l'eseguibile *console.bat*, che di default è la tastiera dell'utente. In particolare, si impone che gli input del programma provengano da un file di testo. Questo è possibile passando come parametro, al lancio del file batch, l'operatore “<”, seguito dal nome del file che contiene le istruzioni. Quindi, come già mostrato in sezione 3.2.1, viene creato un file di testo *istruzioni.txt* nella stessa cartella contenente l'applicazione *console.bat* e, tramite terminale, viene lanciato il comando: *console.bat < istruzioni.txt*.

Capitolo 4

Analisi e Riconciliazione dei dati

Una volta inseriti i dati del grafo strade nel repository, sono state effettuate alcune analisi per verificare la validità dei dati inseriti e l'adeguatezza degli stessi rispetto all'ontologia **SmartCity Ontology**. Le criticità principali sono sorte nella macroarea dei **Servizi**. Gli oggetti facenti parte di questa sezione, infatti, non erano collegati con alcuna entità del Grafo Stradale, quindi si è provveduto a sviluppare una procedura ad hoc per la riconciliazione di queste due aree. Analizzando anche i dati facenti parte del **Trasporto Pubblico Locale**, si è notato come neanche le Fermate degli Autobus fossero in alcun modo collegate al Grafo Stradale, benché aventi ognuna la propria georeferenziazione mediante coordinate spaziali. Pertanto, è stata eseguita anche un'altra riconciliazione che permettesse di mappare ogni fermata del bus con la corrispondente strada. La schematizzazione del processo di riconciliazione è mostrata in figura 4.1.

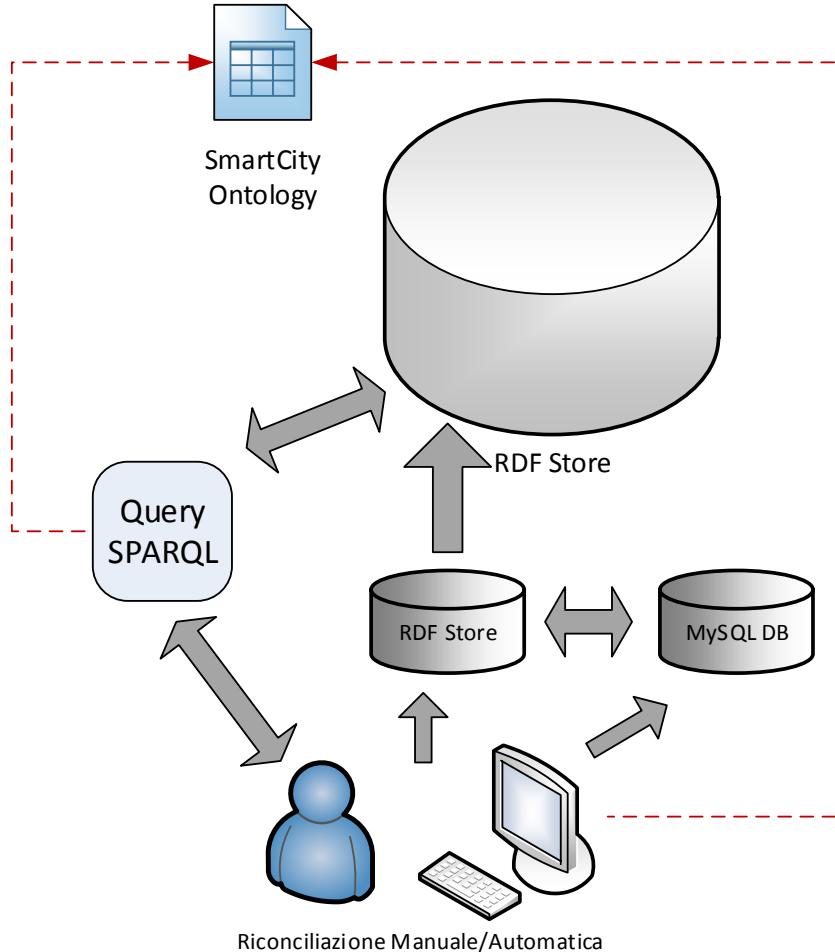


Figura 4.1. Grafico di rappresentazione della fase di Riconciliazione dei Dati

Seguendo la figura, si ha che la **Riconciliazione** è in parte **Manuale** ed in parte **Automatizzata**. Grazie anche ad un **Database MySQL** di appoggio, a partire dai dati errati, vengono generate nuove triple di riconciliazione che, prima di poter essere inserite all'interno dell'**RDF Store** globale, vengono controllate all'interno di uno store temporaneo. Le nuove triple sono generate a partire da **Query SPARQL** personalizzate, inviate al repository centrale. Attraverso le linee tratteggiate, in figura, si intende che il processo di riconciliazione può influenzare direttamente l'ontologia.

4.1 Riconciliazione Toponimi

A causa della disomogeneità con cui sono scritti gli indirizzi nei file provenienti da differenti fonti, si è deciso di applicare un criterio di riconciliazione ai vari toponimi registrati all'interno del grafo stradale. Questo processo, come il processo di riconciliazione dei servizi con i numeri civici descritto nel paragrafo 4.2 fa parte del cosiddetto problema del confronto di indirizzi o *Address Matching*, che a sua volta è solo uno dei domini della più ampia branca dello *String Matching* o confronto tra stringhe. Per l'*Address Matching* esiste un'intera letteratura dedicata all'argomento che propone molteplici soluzioni anche molto complesse. Tuttavia, dato che spesso queste soluzioni mal si adattano alla localizzazione in lingua italiana, e visto che, durante la stesura di questa tesi, il tempo e le risorse a disposizione sono state limitate, si è preferito implementare delle soluzioni ad hoc per i requisiti del progetto.

Come primo passo, si sono analizzati in dettaglio gli indirizzi dei servizi. Al termine dell'analisi sono stati redatti i principali casi di mancato matching. In particolare, si possono elencare i seguenti:

- **Abbreviazioni del DUG:** ad esempio “*VIALE*” si trova scritto “*V.LE*”, “*VLE*”, eccetera.
- **DUG differenti:** in alcuni comuni di provincia, spesso la stessa località viene indicata con DUG diversi.
- **Caratteri accentati:** a seconda dei casi è possibile trovare, per esempio, il termine “*LOCALITÀ*”, “*LOCALITA'*” oppure “*LOCALITA*”.
- **Numeri romani, arabi o scritti in lettere:** soprattutto nelle vie che hanno una denominazione basata su una data o sul nome di un papa (es. “*PIAZZA QUATTRO NOVEMBRE*”, “*PIAZZA IV NOVEMBRE*”, “*PIAZZA 4 NOVEMBRE*”).
- **Santi:** in Italia è diffusissima l'usanza di dedicare vie e piazze a santi e beati. Questo tuttavia genera dell'ambiguità al momento della scrittura dell'indirizzo. Possibili ambiguità si possono trovare ad esempio

nella scrittura dell'indirizzo “*VIA DI SANTA MARTA*”, che può essere reperito anche scritto come “*VIA DI S.MARTA*”, “*VIA DI S MARTA*”, eccetera.

- **Articoli e preposizioni:** gli articoli determinativi e le preposizioni semplici o articolate (“*IL*”, “*LA*”, “*DI*”, “*DELLO*”, eccetera), in molti casi sono rimuovibili mantenendo chiaro l'indirizzo. Ad esempio “*LOCALITA' IL MOLINO*” - “*LOCALITA' MOLINO*”, oppure “*PIAZZA DELLO STATUTO*”, “*PIAZZA STATUTO*”.
- **Nomi propri abbreviati, assenti o invertiti:** ad esempio “*VIALE FRANCESCO PETRARCA*”, “*VIALE F. PETRARCA*”, “*VIALE PETRARCA*”, “*VIALE PETRARCA FRANCESCO*”.

Seguendo questi risultati, è stato costruita una tabella MySQL di appoggio, in cui sono state memorizzate tutte queste possibili ambiguità. In figura 4.2, sono mostrati alcuni record di questa tabella.

ID	StringaDaSostituire	Sostituto1	Sostituto2	Sostituto3	Sostituto4	Sostituto5	Tipo
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	É	E			contains
9	I'	Ì	I				contains
10	O'	Ó	O				contains
11	U'	Ú	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

Figura 4.2. Tabella MySQL di appoggio per la riconciliazione dei toponimi

A partire da questa tabella, sono state ricercate all'interno del repository, tutte le **Road** che, all'interno della proprietà *extendName*, contenevano

almeno una delle stringhe contenute nella tabella. Per ogni toponimo recuperato, sono state generate un certo numero di *dc:alternative* contenenti le alternative indicate nella tabella di appoggio. Al termine, molti toponimi sono stati corredati di numerose alternative. A titolo di esempio, il toponimo “*VIA DI SANTA MARTA*”, ha ad esempio tra le tante alternative, le diciture “*VIA DI S.MARTA*”, “*V. DI SANTA MARTA*”, “*VIA S. MARTA*”, “*V. SANTA MARTA*”, eccetera. Pertanto, un utente che ricerca tale via, ha a disposizione più possibilità di scrittura per ottenere il risultato corretto (al netto degli errori di battitura ovviamente).

L’approccio appena descritto è stato applicato una tantum per generare le alternative, ma il procedimento può essere lanciato periodicamente. La tabella di appoggio MySQL, inoltre, potrebbe essere utilizzata anche con un paradigma inverso. Ovvero, ogni volta che viene effettuata una query di ricerca di un toponimo, si potrebbe confrontare l’indirizzo richiesto con tale tabella. Nel caso in cui il toponimo ricercato presenti una o più stringhe presenti all’interno della tabella, la query potrebbe essere duplicata con ogni possibile alternativa in modo da avere più possibilità di successo. Ad esempio, quando un utente dovesse ricercare il toponimo identificato dalla stringa “*VIA DI SANTA MARTA*”, il sistema dovrebbe ricercare all’interno del repository, non solo le corrispondenze di quella stringa, ma anche le stringhe “*VIA SANTA MARTA*”, “*V. DI SANTA MARTA*”, “*VIA S. MARTA*”, eccetera.

4.2 Riconciliazione dei Servizi con gli Accessi

A partire dal problema appena descritto, e a causa del fatto che i dati relativi ai Servizi provengono da una fonte diversa rispetto a quella del grafo stradale, si è incontrato un problema nell’associare i servizi con il corretto indirizzo posseduto. In particolare, i servizi non erano direttamente associati a qualche entità del grafo strade, ma presentavano il proprio indirizzo all’interno della DataProperty *vcard:street-address*. In alcuni rari casi, i servizi presentavano delle coordinate *geo:lat* e *geo:long*, ma anche in questo caso non erano collegabili direttamente ad alcun oggetto del grafo. Per questo motivo, all’interno

dell'ontologia, sono state previste le due ObjectProperties *isIn* e *hasAccess*, che permettono di collegare i servizi, rispettivamente, a **Road** ed **Entry**. Il procedimento di riconciliazione dei servizi si è rivelato particolarmente lungo e complesso, data l'enorme mole di dati relativi a strade, numeri civici ed accessi presenti all'interno del repository. A causa della dimensione di questi dati, è risultato impossibile applicare un procedimento che, attraverso una query SPARQL unica, mappasse tutti gli indirizzi. Piuttosto, si è fatto ricorso alle API Java fornite dal framework Sesame e sono state costruite delle semplici applicazioni Java per eseguire sequenzialmente delle singole query che cercassero uno ad uno gli indirizzi dei servizi.

Sono stati effettuati tre diversi tipi principali di ricerca:

- **Ricerca per corrispondenza esatta**
- **Ricerca dell'ultima parola**
- **Ricerca tramite strumenti di geocoding di Google e OpenStreetMaps**

Il primo passo è consistito nella ricerca **esatta** degli indirizzi all'interno del grafo stradale, e per far ciò, sono stati estratti tutti gli indirizzi dei circa 30.000 servizi memorizzati nel repository. Tipicamente questi indirizzi (prelevati, come detto, dal campo *vcard:street-address*) presentano il formato *[VIA], [NUMERO CIVICO]*. Il comune di appartenenza, invece, è memorizzato nella proprietà *vcard:locality*. A partire da queste tre informazioni viene generata una query SPARQL, che ne ricerca una corrispondenza esatta all'interno del repository. In figura 4.3, è mostrato un esempio della query.

```

1 PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
2 PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
3 PREFIX foaf:<http://xmlns.com/foaf/0.1/>
4 PREFIX dcterms:<http://purl.org/dc/terms/>
5 PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
6 SELECT distinct ?entry ?elat ?elong
7 WHERE {
8     ?road SiiMobility:inMunicipalityOf ?municipality .
9     ?municipality foaf:name "FIRENZE"^^xsd:string .
10    {
11        ?road SiiMobility:extendName ?extendName .
12        FILTER (ucase(?extendName) = "VIA DELLA VIGNA NUOVA"^^xsd:string) .
13    }
14    UNION
15    {
16        ?road dcterms:alternative ?alternative .
17        FILTER (ucase(?alternative) = "VIA DELLA VIGNA NUOVA"^^xsd:string) .
18    }
19    ?road SiiMobility:hasStreetNumber ?streetNumber .
20    {
21        ?streetNumber SiiMobility:extendNumber "40"^^xsd:string .
22    }
23    UNION
24    {
25        ?streetNumber SiiMobility:extendNumber "42"^^xsd:string .
26    }
27    ?streetNumber SiiMobility:classCode "Rosso"^^xsd:string .
28    ?streetNumber SiiMobility:hasExternalAccess ?entry .
29    ?entry geo:lat ?elat .
30    ?entry geo:long ?elong .
31 }

```

Figura 4.3. Query SPARQL per la ricerca all'interno del repository di una corrispondenza per l'indirizzo VIA DELLA VIGNA NUOVA 40/R-42/R, FIRENZE

Le righe dalla numero 1 alla numero 5 contengono i namespaces utilizzati nella query con le corrispondenti sigle abbreviate. Alla riga 6 vengono definiti i campi che deve restituire la query ovvero tutti i distinti **?entry** (accessi) corrispondenti all'indirizzo richiesto corredati di **elat** (latitudine) e **elong** (longitudine) e . Dalla riga 8 in poi inizia la query vera e propria. Nelle righe 8 e 9 si richiedono tutte le **?road** appartenenti al comune **?municipality** contrassegnato dalla proprietà **foaf:name** uguale a “*FIRENZE*”. Le istruzioni contenute nelle parentesi graffe delle righe 10-13 e 15-18, suddivise dalla parola chiave *UNION*, stanno ad indicare che la **?road** ricercata deve avere **extendName** oppure **dc:alternative** uguale a “*VIA DELLA VIGNA NUOVA*”. La riga 19 estrae tutti gli **?streetNumber** (numeri civici) facenti parte della

?road. Le successive istruzioni delimitate nuovamente da *UNION*, indicano la ricerca alternativa del numero civico “40” e “42”, mentre l’istruzione alla riga 27, specifica che il numero civico ricercato deve avere la proprietà *classCode* uguale a “*Rosso*”. Infine, con le righe 28, 29 e 30 si estraggono, finalmente, gli accessi **?entry** legati alle entità filtrate nelle righe precedenti e le corrispondenti coordinate geografiche **?elat** ed **?elong**.

Grazie a questa query, si riesce a recuperare, se presente, la risorsa di tipo *Entry* univoca corrispondente all’indirizzo sopra indicato. In particolare, un esempio di ritrovamento di Accessi a partire dall’indirizzo del Servizio è riportato in figura 4.4.

Indirizzo del Servizio	Entry ID	Elat	Elong
VIA DELLA VIGNA NUOVA 40/R-42/R, FIRENZE	RT048017000746AC	43.7712868	11.2499669
VIA ARETINA 499, FIRENZE	RT048017006530AC	43.7663602	11.3147545
VIA DI SANTA MARTA 3, FIRENZE	RT048017075274AC	43.7987840	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	10.1265460

Figura 4.4. Esempi di riconciliazione tra indirizzo e accesso

La seconda tipologia di ricerca effettuata, è stata quella tramite ricerca dell’ultima parola. Questa particolare tipologia di ricerca si è svolta in due fasi ed è stata progettata a partire dalla considerazione che, tipicamente, la parte più informativa di un indirizzo corrisponde all’ultima parola della via. Pertanto, partendo dall’elenco dei servizi che ancora non erano stati riconciliati, per ognuno di essi, si è ricercato, all’interno del grafo stradale, tutte le vie che contenessero, all’interno del proprio nome, l’ultima parola dell’indirizzo del servizio. In figura 4.5 è mostrata una query di esempio per questa procedura.

```

1 PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
2 PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
3 PREFIX foaf:<http://xmlns.com/foaf/0.1/>
4 PREFIX dcterms:<http://purl.org/dc/terms/>
5 SELECT distinct ?strada ?nomeVia
6 WHERE
7 {
8   {
9     ?strada SiiMobility:inMunicipalityOf ?comune .
10    ?comune foaf:name "ABBADIA SAN SALVATORE"^^xsd:string .
11    ?strada SiiMobility:extendName ?nomeVia .
12    FILTER contains(ucase(?nomeVia), "TRENTO"^^xsd:string) .
13  }
14 UNION
15 {
16   ?strada SiiMobility:inMunicipalityOf ?comune .
17   ?comune foaf:name "ABBADIA SAN SALVATORE"^^xsd:string .
18   ?strada dcterms:alternative ?nomeVia .
19   FILTER contains(ucase(?nomeVia), "TRENTO"^^xsd:string) .
20 }
21 }

```

Figura 4.5. Query SPARQL per la ricerca all'interno del repository di tutte le vie del comune di ABBADIA SAN SALVATORE contenenti la parola TRENTO

Al solito, le prime righe della query servono per definire le abbreviazioni per i namespace usati all'interno della query, per evitare ogni volta di dover riscrivere l'intera URI della proprietà. I campi estratti in questa query sono **?road**, che contiene l'URI univoco della strada recuperata e **?name**, ovvero il nome per esteso. L'intera query può essere considerata come l'unione di due sottoquery alternative. Si recuperano sempre gli stessi campi, ma nel caso della prima sottoquery (righe 9-12), si ricerca la parola “*TRENTO*” all'interno del campo *extendName*, mentre nel secondo caso all'interno della proprietà *dc:alternative*. In entrambi i casi, ovviamente, si filtrano i risultati esclusivamente sul comune selezionato, ovvero “*ABBADIA SAN SALVATORE*”.

Tramite questa procedura, si riescono ad ottenere tutte le corrispondenze delle vie con l'ultima parola dell'indirizzo del servizio. Tuttavia, queste corrispondenze, non sempre sono esatte. Pertanto, è risultato necessario verificare manualmente queste corrispondenze. Allo stato attuale, sono stati analizzati

tutti i servizi che presentavano al massimo due corrispondenze. In figura 4.6, è mostrato un esempio di selezione delle **Road** corrette corrispondenti effettivamente all'indirizzo del servizio.

COMUNE	INDIRIZZO SERVIZIO	NOME GRAFO STRADALE	URI TOPONIMO
FIRENZE	VIA BORGO DEI GRECI	VIA GRECIA	http://www.disit.dinfo.unifi.it/SiiMobility/RT04801707463TO
FIRENZE	VIA BORGO DEI GRECI	BORGO DÉ GRECI	http://www.disit.dinfo.unifi.it/SiiMobility/RT04801701830TO
FIRENZE	VIA NUOVA D CACCINI	VIA GIULIO CACCINI	http://www.disit.dinfo.unifi.it/SiiMobility/RT04801703143TO
FIRENZE	VIA NUOVA D CACCINI	V NUOVA DEI CACCINI	http://www.disit.dinfo.unifi.it/SiiMobility/RT04801703400TO
POPKI	VIA CASA DAMIANO	VIA CASE DAMIANO	http://www.disit.dinfo.unifi.it/SiiMobility/RT05103119173TO
POPKI	VIA CASA DAMIANO	PZA DAMIANO CHIESA	http://www.disit.dinfo.unifi.it/SiiMobility/RT05103120951TO
PONTEDERA	VIA LOTTI	VIA FELICE CAVALLOTTI	http://www.disit.dinfo.unifi.it/SiiMobility/RT05002901189TO
PONTEDERA	VIA LOTTI	VIA FELICE LOTTI	http://www.disit.dinfo.unifi.it/SiiMobility/RT05002901191TO

Figura 4.6. Procedura di riconciliazione manuale dei Servizi attraverso ricerca dell'ultima parola

Nella figura di esempio, si ha che la ricerca di strade con ultima parola “*GRECI*” a partire dall’indirizzo del servizio “*VIA BORGO DEI GRECI*”, facente parte del comune di Firenze restituisce due toponimi: “*VIA GRECIA*” e “*BORGO DE’ GRECI*”. Ovviamente, il toponimo corretto è il secondo, e viene manualmente selezionato.

Come ultimo passo di questa procedura, sulle **Road** selezionate, è stata nuovamente effettuata una query di ricerca del numero civico limitata alle entità di tipo **StreetNumber** facenti parte della **Road** ritrovata.

L’ultima tipologia di ricerca che ha prodotto un numero consistente di risultati, è stata quella effettuata tramite gli strumenti e le API di **geocoding**, messi a disposizione da *Google*¹ e *OpenStreetMaps* (tramite il servizio denominato **Nominatim**²). La procedura di geocoding, è un processo che permette di convertire indirizzi (ad esempio “*Via di Santa Marta, 3, Firenze*”) in coordinate geografiche (ad esempio *43.7976054, 11.253943* in WGS84). Tramite le API messe a disposizione da questi servizi, si sono ricercate le associazioni per tutti quei servizi per i quali non si era riusciti a trovare una corrispondenza, tramite i metodi precedenti. In particolare, dopo aver recuperato le coordinate geospatiali di un servizio, si è ricercato l’oggetto di tipo **Entry**

¹The Google Geocoding API - <https://developers.google.com/maps/documentation/geocoding/>

²Nominatim - <http://wiki.openstreetmap.org/wiki/Nominatim>

geograficamente più vicino, in un raggio molto ridotto per evitare errori.

Durante il processo di riconciliazione, oltre ai tre tipi di ricerca principali appena elencati, sono anche state effettuate alcune correzioni manuali dei dati e sono state progettate alcune migliorie minori alle applicazioni di ricerca, per migliorare i risultati delle query. Alcune di queste sono visibili nella query descritta in figura 4.3 (come ad esempio la gestione dei numeri rossi, la gestione di numeri civici alternativi), ma ne sono state eseguite molte altre, quali ad esempio:

- Sostituzione del nome del comune nel caso fosse diverso dalla denominazione ufficiale (ad esempio, alcuni servizi hanno come proprietà *vcard:locality* il comune “*VICCHIO DI MUGELLO*”, mentre la denominazione ufficiale del comune è semplicemente “*VICCHIO*”).
- Gestione ed eliminazione di caratteri strani all’interno di vie o numeri civici, che impedivano il corretto ritrovamento della corrispondenza.
- Correzione di evidenti errori ortografici o di battitura.

4.3 Riconciliazione Fermate degli Autobus

I dati sulle fermate degli autobus disponibili, allo stato attuale, non permettono il collegamento diretto tra Fermate e Strade. In particolare, nei dati a disposizione riguardanti le **BusStop**, non esiste alcun indirizzo né tanto meno alcun riferimento a entità del Grafo Stradale. Per questo motivo, si è realizzata una procedura di riconciliazione che collegasse ogni fermata ad una particolare **Road** tramite la proprietà *isIn* definita nell’ontologia. Per ogni fermata non riconciliata, è stata effettuata una query SPARQL geospaziale che cercasse in un raggio ridotto, intorno alla sua posizione GPS, l’entità **Entry** più vicina. Attraverso le ObjectProperties che legano le classi all’interno del grafo stradale è quindi possibile risalire alla **Road** richiesta, ed effettuare l’associazione. Un esempio di query per questo tipo di riconciliazione è mostrato in figura 4.7.

```

1 PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
2 PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
3 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4 PREFIX omgeo:<http://www.ontotext.com/owlim/geo#>
5 SELECT distinct ?road ?distance
6 WHERE {
7 ?entry rdf:type SiiMobility:Entry .
8 ?streetNumber SiiMobility:hasExternalAccess ?entry .
9 ?streetNumber SiiMobility:belongTo ?road .
10 ?entry geo:lat ?elat .
11 ?entry geo:long ?elong .
12 ?entry omgeo:nearby(43.7754868 11.2480146 "0.1km") .
13 BIND( omgeo:distance(?elat, ?elong, 43.7754868, 11.2480146) AS ?distance) .
14 }
15 ORDER BY ?distance
16 LIMIT 1

```

Figura 4.7. Esempio di query SPARQL per il recupero della Road a cui appartiene una BusStop

Le righe dalla 1 alla 4 contengono le definizioni dei namespaces. La riga 5 stabilisce che il risultato della query è formato da un oggetto **?road** e da **?distance**. Alla riga 7 si selezionano esclusivamente gli oggetti di tipo **?entry**, alla riga 8 si risale alla classe **StreetNumber** tramite la proprietà *hasExternalAccess*, mentre alla riga 9 si arriva a **?road** tramite la Object-Property *belongTo* definita nell'ontologia **SmartCity Ontology**. Alle righe 10 e 11 si estraggono le coordinate *geo:lat* e *geo:long* degli Accessi. Alla riga 12 vengono scartate tutte le **?entry** che non rientrano in un raggio di 100 metri dalle coordinate *43.7754868, 11.2480146* (corrispondenti alla fermata “*STAZIONE SCALETTE*”), tramite l'operatore **omgeo:nearby** nativo del software OWLIM. Infine, tramite l'operatore *BIND* del linguaggio SPARQL, si associa alla variabile **?distance** il valore restituito dal calcolo della distanza dei vari **?entry** rispetto al punto di ricerca, grazie alla proprietà **omgeo:distance**, anch'essa implementata nativamente in OWLIM. La riga 17 impone un ordinamento dei risultati proprio in base al valore della variabile **?distance**, infine con l'istruzione alla riga 16 si mantiene esclusivamente il primo risultato, ovvero l'accesso più vicino alla fermata e di conseguenza la sua **?road** associata.

Il risultato di questa query è il seguente:

Il risultato della query specifica che la **Road** più vicina alla **BusStop** de-

Riconciliazione Fermate degli Autobus

Road	Distance
< http://www.disit.dinfo.unifi.it/SiiMobility/RT04801701942T0 >	0.034km

Tabella 4.1. Elenco dei risultati ottenuti in risposta alla query di figura 4.7

nominata “*STAZIONE SCALETTA*” è <<http://www.disit.dinfo.unifi.it/SiiMobility/RT04801701942T0>> e che l’accesso più vicino presente all’interno di quella strada, si trova a 34 metri di distanza dalla fermata. Il Toponimo appena recuperato, all’interno del Grafo Stradale, è denominato “*PIAZZA DELLA STAZIONE*”, quindi in questo caso di esempio la query ha fornito un ottimo risultato.

Capitolo 5

Interrogazione e visualizzazione dei dati

Dopo aver inserito i dati del grafo stradale, ed averli corredati con quelli provenienti dalle altre fonti, si sono potuti progettare ed implementare degli strumenti di visualizzazione. La base per il recupero di qualsiasi informazione contenuta in un repository semantico è il linguaggio di interrogazione SPARQL. Attraverso la definizione di query SPARQL è possibile recuperare ogni tipo di informazione in modo semplice e personalizzato.

Per il repository RDF costruito in questo lavoro di tesi, è stato predisposto un punto di accesso per la realizzazione di query SPARQL, altrimenti detto **SPARQL Endpoint**. Collegandosi a tale punto di accesso, è possibile, conoscendo l'ontologia sottostante i dati, eseguire delle query personalizzate ed ottenere risultati in modo rapido ed affidabile. L'endpoint SPARQL del repository può essere interrogato in vari modi, ad esempio tramite l'interfaccia web nativa di Sesame, come mostrato in figura 5.1.

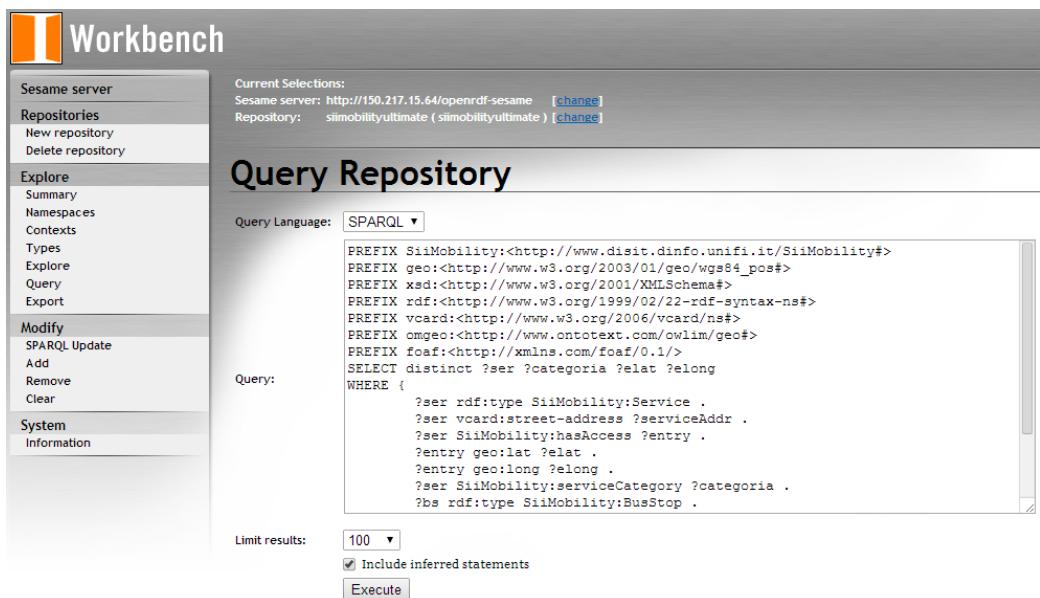
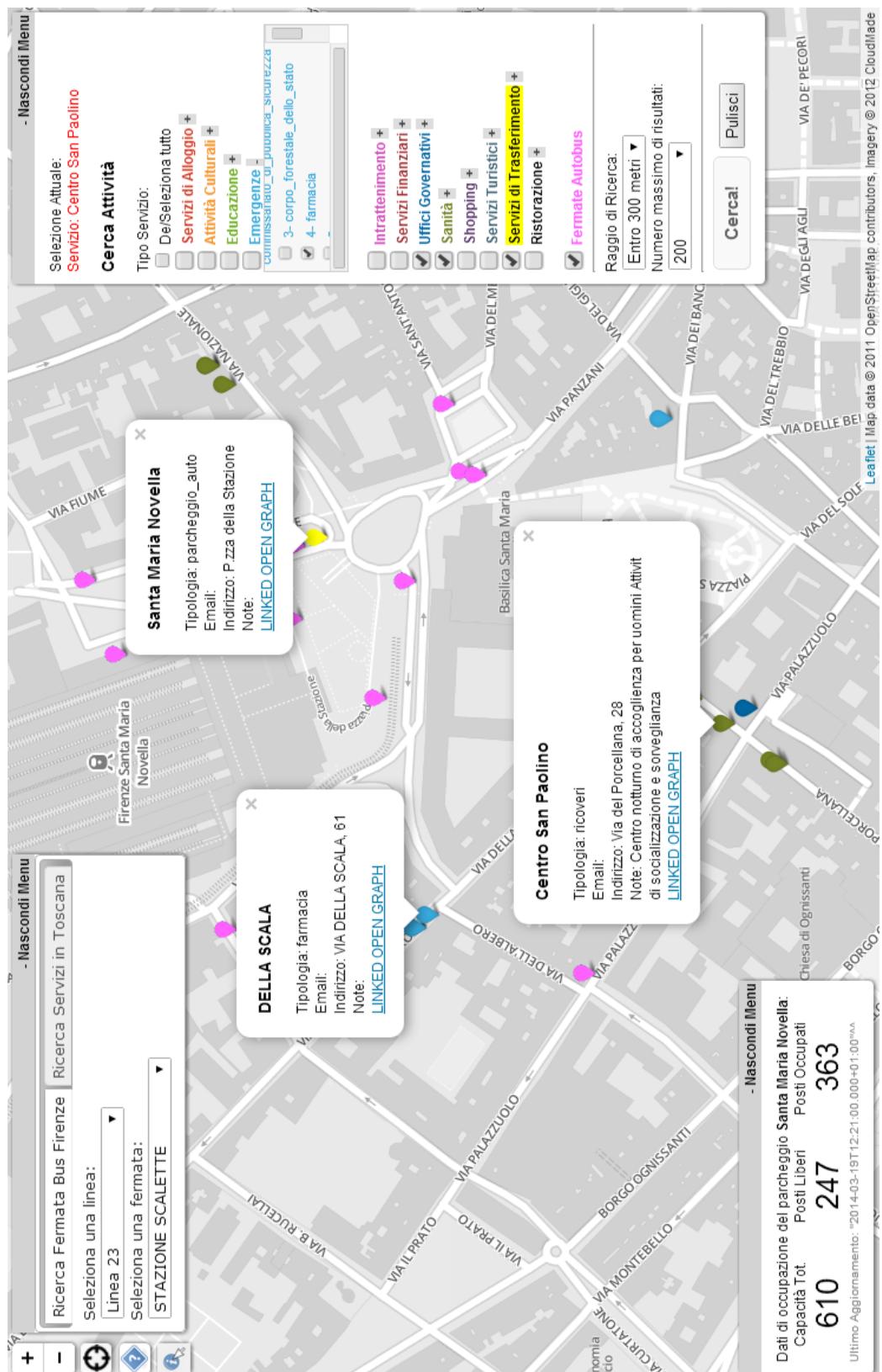


Figura 5.1. Interfaccia web di Sesame per l'interrogazione del repository con query SPARQL

5.1 Visualizzazione tramite ServiceMap

Poichè il numero di dati contenenti informazioni geografiche memorizzati all'interno del repository è molto elevato, è stata sviluppata una applicazione web sperimentale, chiamata **ServiceMap**, che consentisse la visualizzazione dei dati georeferenziati direttamente su una mappa mostrata all'utente, tramite browser. Sono stati previsti alcuni casi d'uso ed è stata sviluppata di conseguenza una interfaccia come mostrato in figura 5.2.



INTERROGAZIONE E VISUALIZZAZIONE DEI DATI 127
Figura 5.2. Interfaccia attualmente in funzione per il servizio ServiceMap

Per realizzare la mappa, è stata creata una applicazione JSP, che grazie alle API Java del framework Sesame, permette di recuperare dati dal repository. Per la creazione della mappa, si è fatto ricorso alla libreria JavaScript **Leaflet**¹. Tale libreria, attraverso degli script dalle dimensioni estremamente contenute, permette di creare delle mappe interattive all'interno di una applicazione web. Grazie a delle API ottimamente documentate, e ad una estesa lista di plugin di terze parti facilmente installabili, Leaflet è in grado di costruire una mappa estremamente usabile e versatile, facendo uso delle mappe open-source OpenStreetMap.

In figura 5.3, è mostrata una schematizzazione del processo sottostante la ServiceMap interattiva.

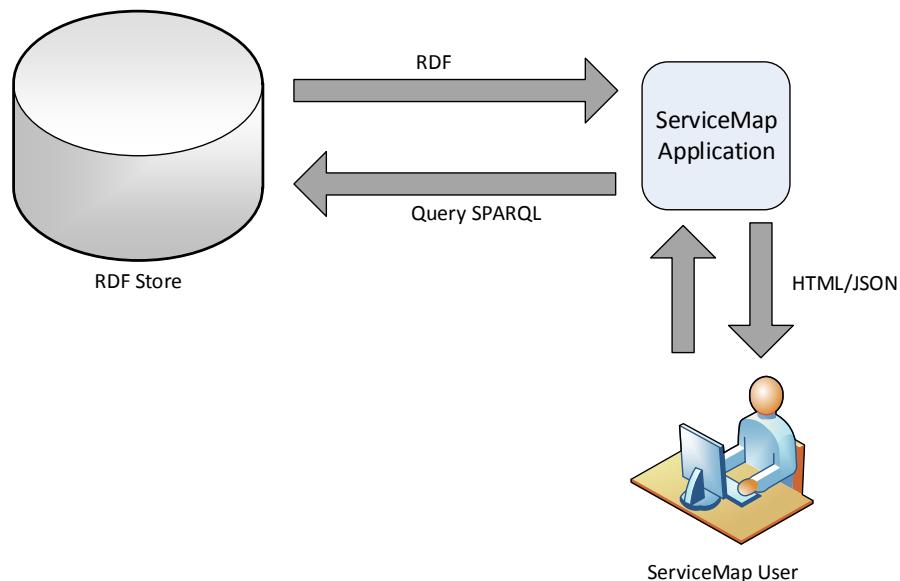


Figura 5.3. Rappresentazione grafica dell'interrogazione e della visualizzazione dei dati tramite ServiceMap

L'insieme di tutte le operazioni effettuabili attraverso la ServiceMap sono riassumibili nella figura appena mostrata. Genericamente, l'utente (**ServiceMap User**

¹Leaflet, an Open-Source JavaScript Library for Mobile-Friendly Interactive Maps - <http://leafletjs.com/>

User) si collega alla **ServiceMap Application** tramite browser web. Attraverso una interazione con la mappa, esso invia delle richieste al server. Queste richieste sono elaborate, convertite in **Query SPARQL** ed inviate all'RDF Store contenente i dati di questo progetto. Qui la query SPARQL viene processata, e restituisce dati in formato RDF. Il server, a questo punto, interpreta questi dati e, a seconda del tipo di informazioni richiesto, converte i risultati in formato HTML o in JSON e li mostra all'utente.

La progettazione dell'applicazione e dell'interfaccia ha seguito i requisiti forniti da alcuni casi d'uso previsti nell'analisi di questo lavoro. Un diagramma dei principali casi d'uso previsti è mostrato in figura 2.2. Nei prossimi paragrafi, sono descritti in dettaglio tali casi d'uso.

L'interfaccia dell'applicazione è stata concepita nel modo descritto di seguito. La mappa interattiva occupa ovviamente tutto lo schermo a disposizione, mentre i menu di interazione sono suddivisi in tre parti: in alto a sinistra si hanno alcuni filtri di partenza per i casi d'uso attualmente implementati, in alto a destra, si hanno dei filtri che permettono di specificare quali tipologie di servizi mostrare, quanti mostrarne e a che distanza massima dal punto di ricerca; infine, in basso a sinistra, si ha un menu contestuale che mostra delle informazioni aggiuntive di interesse, che possono variare dalle previsioni del tempo, alle previsioni di transito degli autobus su una determinata fermata, per finire con i dati di occupazione di un determinato parcheggio, a seconda di quale sia l'elemento attualmente selezionato sulla mappa.

Di seguito vengono mostrate alcune immagini dei menu dell'interfaccia, comprensive di spiegazioni. In figura 5.4, è mostrato il menu in alto a sinistra, aperto su entrambe le *tab* disponibili.

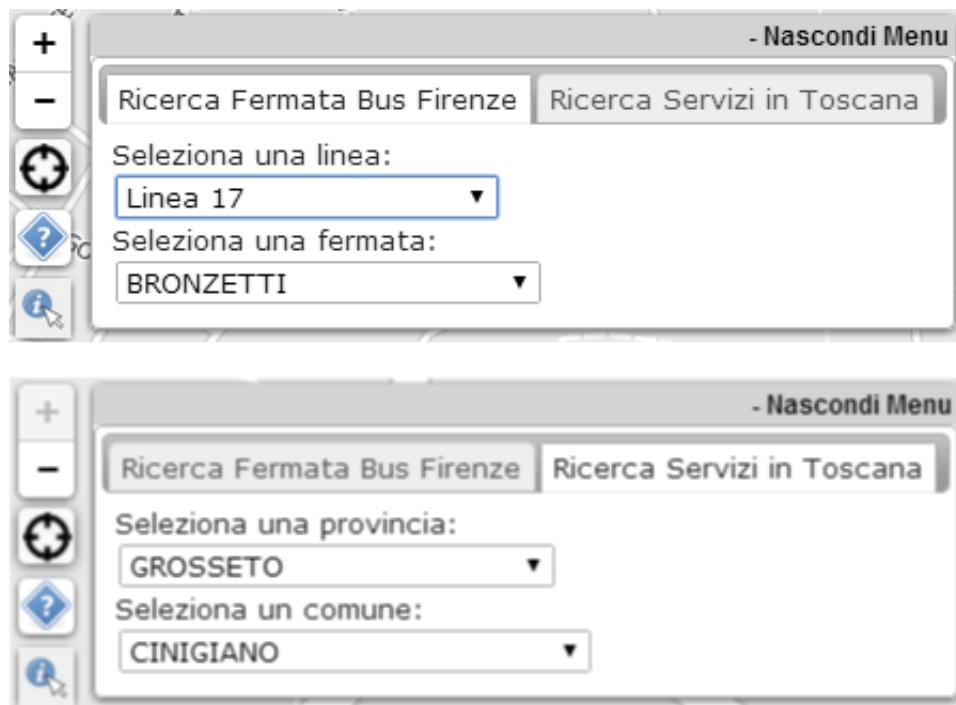


Figura 5.4. Dettaglio del menu in alto a sinistra dell'interfaccia per ServiceMap

Come si nota, a sinistra del box di selezione iniziale sono presenti alcuni pulsanti. I primi due sono nativi della libreria Leaflet e consentono il controllo sullo Zoom della mappa. Il terzo pulsante serve per attivare la ricerca della posizione attuale dell'utente. Una volta individuata la posizione, la mappa viene centrata di conseguenza ed un marker viene aggiunto in corrispondenza della propria posizione. Il quarto pulsante è semplicemente un link che rimanda ad una pagina web di descrizione dell'applicazione ServiceMap. Infine, tramite l'ultimo pulsante è possibile inserire un *pin* in un qualsiasi punto della mappa. Attraverso questa operazione è possibile ottenere due diversi tipi di risultati: si possono ricercare i servizi circostanti al punto (dettagli in sezione 5.1.3 ed ottenere l'indirizzo approssimativo del punto selezionato).

Il menu in alto a destra permette la selezione di vari parametri di ricerca, ed è mostrato in figura 5.5.



Figura 5.5. Dettaglio del menu in alto a destra dell'interfaccia per ServiceMap

In particolare, tramite questo secondo menu, è possibile, tramite selezione di *checkbox*, selezionare in dettaglio quali categorie di servizi si vogliono recuperare dalla ricerca, sia a livello di **ServiceCategory** che di classe di Servizio (ad esempio **Education**, **Entertainment**, eccetera). Oltre alle categorie di

Visualizzazione tramite ServiceMap

servizi, è mostrato un *checkbox* anche per gli oggetti di tipo Fermate Autobus. Inoltre, per le tipologie di ricerca geolocalizzate, si può specificare il raggio massimo di ricerca (fino a 500 metri) ed il numero massimo di risultati che si vogliono visualizzare (per evitare, in alcuni casi, il sovraffollamento di markers sulla mappa). Infine, grazie ai due pulsanti “*Cerca!*” e “*Pulisce!*” si può, rispettivamente, far partire la ricerca di servizi e riportare l’applicazione allo stato iniziale.

L’ultimo menu che viene descritto è il cosiddetto *menu contestuale*, che fornisce all’utente informazioni contestuali utili ai fini della propria ricerca. In figura 5.6, è mostrato tale menu con tutti i tipi di informazioni attualmente visualizzabili.

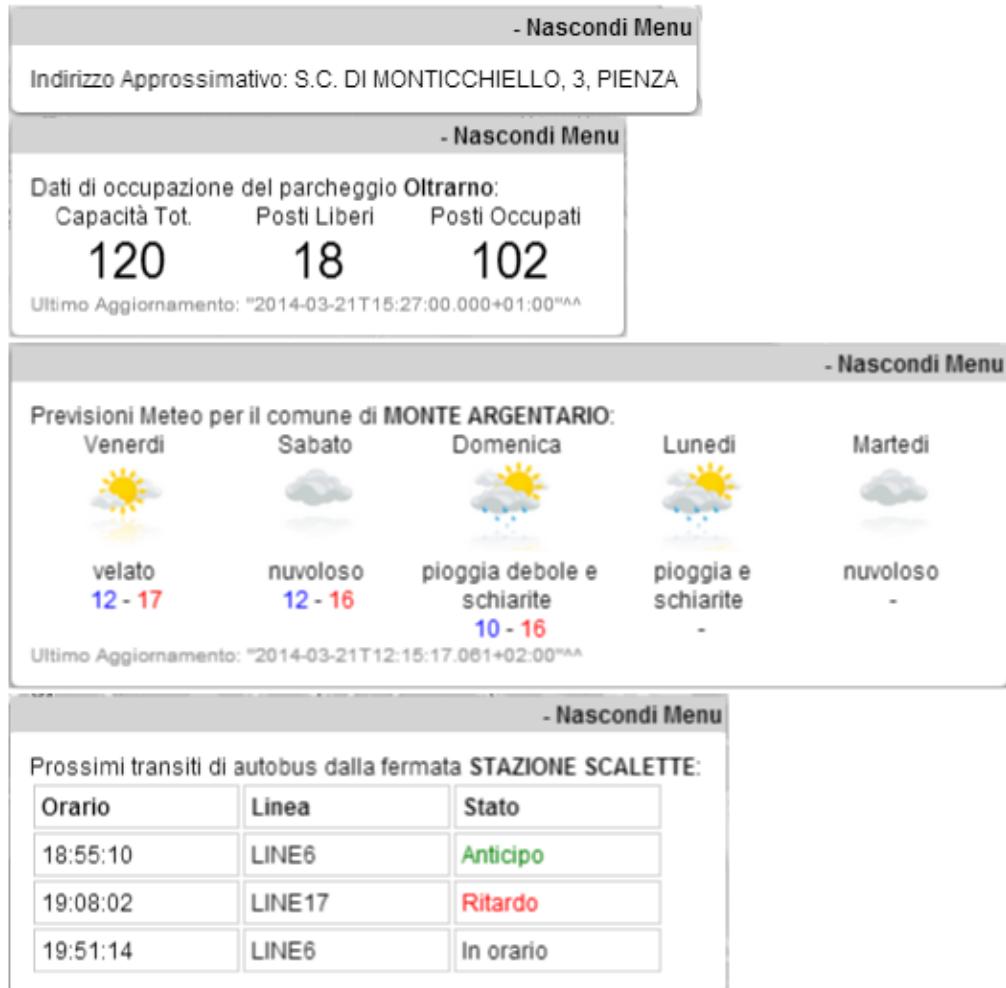


Figura 5.6. Dettaglio del menu in basso a sinistra dell'interfaccia per ServiceMap

Tutti i menu appena descritti sono nascondibili cliccando sulla scritta “*Nascondi Menu*”.

Grazie al performante motore RDF implementato in OWLIM, e grazie all’uso dei suoi indici geospatiali estremamente potenti, è possibile ottenere risultati dalle interrogazioni con un tempo di risposta molto breve, nono-

stante la consistente mole di triple georeferenziate presenti all'interno del repository.

L'applicazione **ServiceMap** è disponibile all'indirizzo: <http://servicemap.sii-mobility.org/>.

5.1.1 Caso d'uso 1: ricerca Servizi appartenenti ad un Comune

Il primo caso d'uso prevede il recupero di Servizi appartenenti ad un determinato comune. In questo caso, non si ha una ricerca prettamente geografica, ma si filtrano i risultati in base al comune di appartenenza. In questo contesto, la prima operazione da svolgere, è quella di scegliere il Comune all'interno del quale si debbono ricercare i dati. Per farlo, è sufficiente selezionare prima la provincia (tramite il dropdown “*Seleziona una provincia*”), e successivamente il comune esatto. Se non si conosce la provincia di appartenenza di un determinato comune, è possibile impostare, come prima scelta, il valore “*TUTTE LE PROVINCE*”, per avere l'elenco di tutti i 286 comuni toscani. A questo punto, devono essere selezionate le categorie di interesse ed il numero massimo di risultati che si vogliono ottenere. Il menu di filtraggio è mostrato in figura 5.15, mentre la visualizzazione della procedura di selezione delle categorie e del numero massimo di risultati è descritto nel prossimo paragrafo 5.1.2. Cliccando sul consueto pulsante “*Cerca!*”, si visualizzeranno sulla mappa i servizi richiesti.

Un esempio di query SPARQL per il ritrovamento di dati all'interno di un comune è mostrata in figura 5.7.

```

1 PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
2 PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
3 PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
4 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
5 PREFIX vcard:<http://www.w3.org/2006/vcard/ns#>
6 PREFIX foaf:<http://xmlns.com/foaf/0.1/>
7 SELECT distinct ?service ?serviceAddress ?elat ?elong ?serviceName
8 ?serviceType ?email ?note
9 WHERE {
10     ?service rdf:type SiiMobility:Service .
11     {
12         ?service SiiMobility:serviceCategory "farmacia"^^xsd:string .
13         BIND ("farmacia"^^xsd:string AS ?serviceType) .
14     }
15     UNION
16     {
17         ?service SiiMobility:serviceCategory "guardia_medica"^^xsd:string .
18         BIND ("guardia_medica"^^xsd:string AS ?serviceType) .
19     }
20     ?service vcard:organization-name ?serviceName .
21     ?service vcard:street-address ?serviceAddress .
22     OPTIONAL {?ser vcard:note ?note} .
23     OPTIONAL {?ser SiiMobility:email ?email} .
24     ?service SiiMobility:hasAccess ?entry .
25     ?entry geo:lat ?elat .
26     ?entry geo:long ?elong .
27     ?streetNumber SiiMobility:hasExternalAccess ?entry .
28     ?streetNumber SiiMobility:belongsTo ?road .
29     ?road SiiMobility:inMunicipalityOf ?municipality .
30     ?municipality foaf:name "EMPOLI"^^xsd:string .
31 }

```

Figura 5.7. Query SPARQL per la ricerca dei Servizi di tipo FARMACIA e GUARDIA MEDICA nel comune di EMPOLI

La descrizione della query è riportata di seguito. Le prime 6 righe contengono le definizioni dei namespaces, con le righe 7 e 8 si richiede di recuperare le informazioni di interesse del servizio, ovvero: URI, indirizzo, nome (o ragione sociale), categoria di servizio, indirizzo email, latitudine e longitudine dell'accesso corrispondente, ed eventuali note. Con la riga 10 si selezionano, inizialmente, tutte le entità di tipo **?service**. Attraverso le due coppie di istruzioni, delimitate dall'operatore alternativo *UNION*, si selezionano esclusivamente i servizi di tipo “farmacia” e “guardia medica”, e si mappa il tipo richiesto all'interno della variabile **?serviceType**, altrimenti non sarebbe recuperabile direttamente all'interno dei campi restituiti dalla query. Dalla riga 20 alla riga 23 si estraggono le informazioni di interesse del servizio

(**?serviceName**, **?serviceAddress**, **?email** e **?note**). L'indirizzo email e le note, poichè non sempre sono valorizzate, sono richieste attraverso la parola chiave *OPTIONAL*, che permette di ottenere risultati anche in assenza di tali dati. Altrimenti, il procedimento di *triple matching* del linguaggio SPARQL scarterebbe dal risultato tutti quei servizi sprovvisti del campo *email* o del campo *skos:note*. Con la riga 24 si estrae l'accesso **?entry** collegato al servizio, ed alle righe successive le sue coordinate geografiche (**?elat** ed **?elong**). Righe 27-30: a partire dall'oggetto **?entry**, si recuperano sequenzialmente, il numero civico relativo, il toponimo a cui esso appartiene ed infine il comune all'interno del quale si trova il toponimo.

I dati recuperati dalla query sono mostrati di seguito in figura 5.8.

ServiceAddress	Elat	Elong	ServiceName	ServiceType
PIAZZA SAN ROCCO, 10	43.7193405	10.9393343	NUOVA DOTTOR VALOROSI	farmacia
PIAZZA DELLA VITTORIA, 26	43.7202826	10.9489318	BIZZARRI	farmacia
VIA CAPPUCCHINI, 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	10.9528700	BOLOGNESI	farmacia

Figura 5.8. Dati recuperati dalla query SPARQL di figura 5.7

Attraverso l'applicazione, i dati recuperati in formato RDF sono convertiti in JSON. Tale formato è richiesto dalla libreria Leaflet per semplificare l'inserimento dei *markers* corrispondenti ai servizi, all'interno della mappa. In figura 5.9, è mostrato una porzione del codice JSON corrispondente alla query di esempio.

Visualizzazione tramite ServiceMap

```
{  
  "type": "FeatureCollection",  
  "features": [  
    {  
      "geometry": {  
        "type": "Point",  
        "coordinates": [  
          10.9393343965,  
          43.7193405384  
        ]  
      },  
      "type": "Feature",  
      "properties": {  
        "popupContent": "NUOVA DOTTOR VALOROSI - farmacia",  
        "nome": "NUOVA DOTTOR VALOROSI",  
        "tipo": "farmacia",  
        "email": "",  
        "note": "",  
        "serviceId": "http://www.disit.dinfo.unifi.it/SiiMobility/NUOVA\_DOTTOR\_VALOROSI-PIAZZA\_SAN\_ROCCO\_10",  
        "indirizzo": "PIAZZA SAN ROCCO, 10"  
      },  
      "id": 1  
    },  
  ]  
}
```

Figura 5.9. Esempio di dati JSON corrispondenti ai risultati della query di figura 5.7

I servizi recuperati della query, sono quindi disposti sulla mappa dagli script di Leaflet, come mostrato in figura 5.10.

Visualizzazione tramite ServiceMap

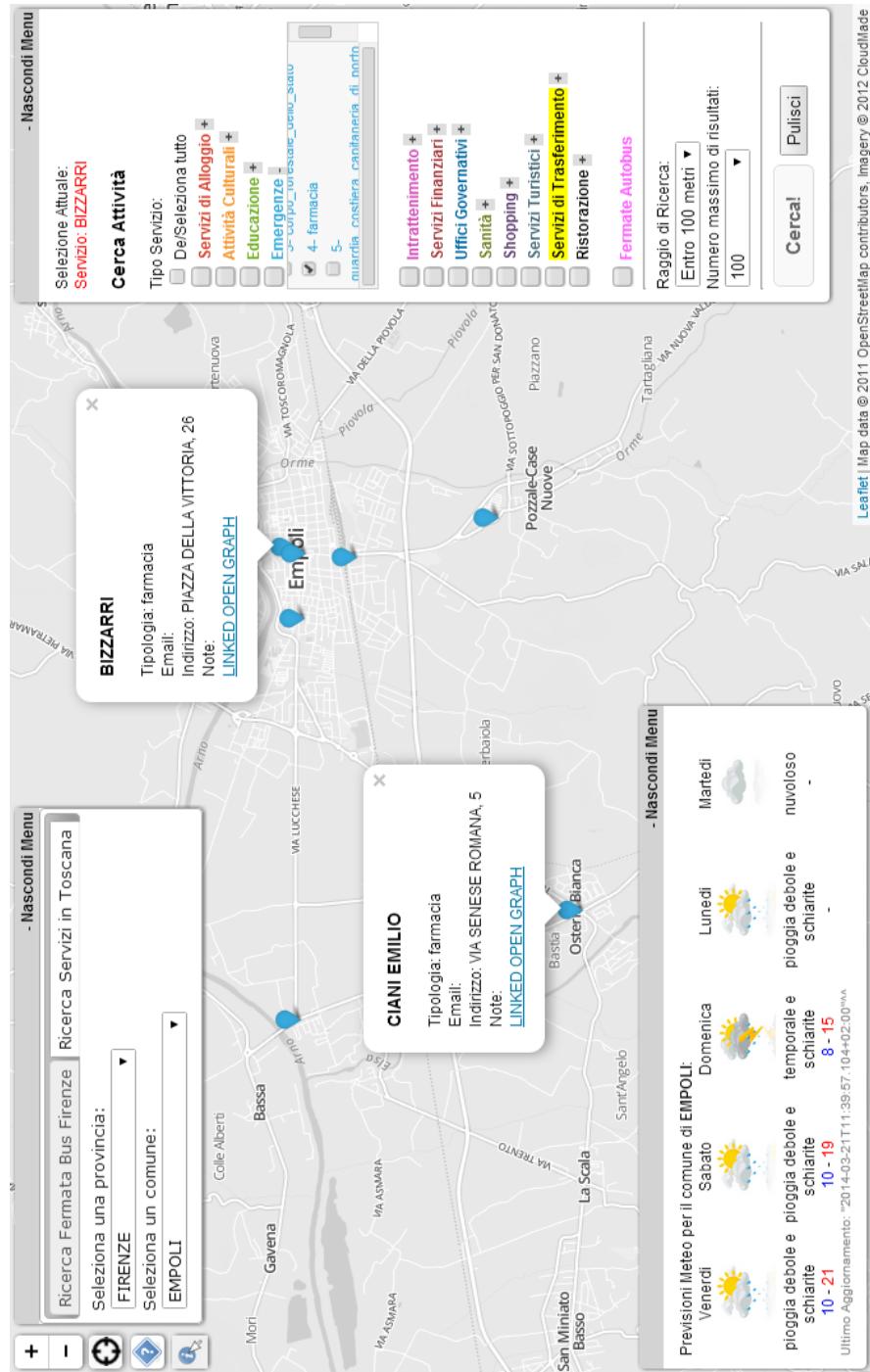


Figura 5.10. Risultati della ricerca di servizi nel Comune di EMPOLI

Il diagramma di sequenza tra lato client e lato server della procedura

appena descritta è mostrato in figura 5.11.

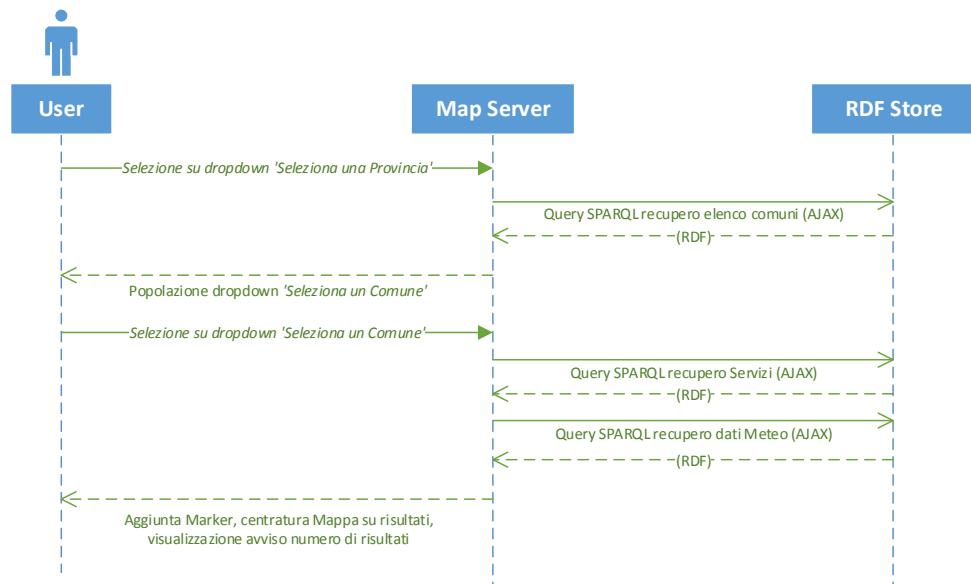


Figura 5.11. Diagramma di sequenza client-server per il caso d'uso numero 1

Contemporaneamente alla ricerca di servizi all'interno del comune, viene lanciata anche una interrogazione che recupera le previsioni meteo più aggiornate per quel determinato comune. La procedura di ricerca è suddivisa in query sequenziali: nella prima si estrae il **WeatherReport** con data maggiore (quindi quello più recente), nella seconda, a partire dal Report appena ottenuto si ricercano i singoli **WeatherPrediction** con le previsioni dei 5 giorni successivi alla data attuale. Le due query SPARQL per il recupero di questi dati sono mostrate in figura 5.12.

```

1 PREFIX foaf:<http://xmlns.com/foaf/0.1/>
2 PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
3 PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
4 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
5 PREFIX time:<http://www.w3.org/2006/time#>
6 SELECT distinct ?weatherReport ?instantDateTime
7 WHERE{
8     ?municipality rdf:type SiiMobility:Municipality .
9     ?municipality foaf:name "LUCCA"^^xsd:string .
10    ?municipality SiiMobility:has ?weatherReport .
11    ?weatherReport SiiMobility:updateTime ?updateTime .
12    ?updateTime time:inXSDDateTime ?instantDateTime .
13 }
14 ORDER BY DESC (?instantDateTime) LIMIT 1
15
16 <http://www.disit.dinfo.unifi.it/SiiMobility/Lucca1395390606000>
17
18 PREFIX dcterms:<http://purl.org/dc/terms/>
19 PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
20 PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
21 PREFIX time:<http://www.w3.org/2006/time#>
22 SELECT distinct ?day ?description ?minTemp ?maxTemp
23 WHERE{
24     <http://www.disit.dinfo.unifi.it/SiiMobility/Lucca1395390606000>
25     . . . SiiMobility:isComposedOf ?weatherPrediction .
26     ?weatherPrediction dcterms:description ?description .
27     ?weatherPrediction SiiMobility:day ?day.
28     ?weatherPrediction SiiMobility:hour "giorno"^^xsd:string .
29     OPTIONAL { ?weatherPrediction SiiMobility:minTemp ?minTemp . }
30     OPTIONAL { ?weatherPrediction SiiMobility:maxTemp ?maxTemp . }
31 }

```

Figura 5.12. Query SPARQL per il recupero delle previsioni meteo più recenti per il comune di LUCCA

La prima query è composta dalle righe 1-14. Alla riga 16 è mostrato il valore assunto dalla variabile **?weatherReport** nel risultato della prima query. La seconda query (righe 18-31), utilizzerà proprio tale valore come filtro per recuperare gli oggetti di tipo **?weatherPrediction**. Sempre dalla prima query si estraе il valore **?instantDateTime** che corrisponde all'orario relativo al report. Con le righe 8 e 9 si seleziona il comune di *LUCCA*, alla riga 10 si estraggono tutti i report del comune, mentre alla riga 11 si estraе l'oggetto di classe **time:Instant** collegato al **WeatherReport** che ne rappresenta la data di aggiornamento, o **?updateTime**. L'orario vero e proprio, in formato

xsd:dateTime, viene estratto dalla DataProperty *time.inXSDDdateTime* di **?updateTime**. Infine, con l'istruzione alla riga 14 si ordinano in modo decrescente i report in base alla loro data e si seleziona esclusivamente il primo, che di conseguenza è il più recente in ordine temporale.

Passando alla seconda query, si nota come alla riga 24 e 25 vengano estratte tutte le **?weatherPrediction** a partire dal report meteo appena recuperato (<http://www.disit.dinfo.unifi.it/SiiMobility/Lucca1395390606000>). Alle righe 26, 27, 29 e 30 si recuperano gli attributi di interesse della previsione: **?day**, **?description**, **?minTemp** e **?maxTemp**, ovvero il giorno della settimana (“Lunedì”, “Martedì”, eccetera), la descrizione letterale della previsione (“Sole”, “Pioggia”, eccetera), e le temperature minime e massime previste per quel giorno. Le variabili **?minTemp** e **?maxTemp** sono ricercate attraverso l'operatore *OPTIONAL* che permette anche la mancanza di tali informazioni. Questo perché, tipicamente, le previsioni del tempo hanno associate le previsioni di temperatura esclusivamente fino ad un massimo di 3 dei 5 giorni seguenti. Infine, alla riga 28 si impone di selezionare esclusivamente le previsioni per il “giorno”. Anche in questo caso si è scelto di prendere le previsioni globali, invece di quelle a granularità più fine (“mattina”, “pomeriggio”, “sera”), perché esse non sono presenti nelle previsioni di tutti e 5 i giorni seguenti.

Il risultato della seconda query è formattato per renderlo maggiormente leggibile e mostrato nel menu contestuale come mostrato in figura 5.13.



Figura 5.13. Visualizzazione nel menu contestuale di ServiceMap delle previsioni del tempo per il comune di LUCCA

5.1.2 Caso d'uso 2: ricerca Servizi prossimi a Fermate Autobus

Il secondo caso d'uso che è stato delineato è quello della ricerca di determinati tipi di servizi, vicini ad una fermata dell'autobus, all'interno dell'area metropolitana fiorentina. Per selezionare la fermata di partenza è necessario, come primo passo, selezionare la linea dell'autobus attraverso il menu *Seleziona una linea*. Una volta selezionata una linea (oppure il valore “*TUTTE LE LINEE*”), nel dropdown sottostante verranno caricate tutte le fermate corrispondenti a quella particolare linea. Selezionando una fermata, la mappa si centerà sulle sue coordinate e verrà mostrato un piccolo marker sulla mappa. Contemporaneamente, il sistema caricherà, nel menu contestuale, le informazioni relative ai dati AVM dei transiti su quella particolare fermata. Una descrizione più accurata delle query per il recupero dei dati AVM sono descritte in fondo a questa sezione. In figura 5.14 è mostrato questo primo passo.

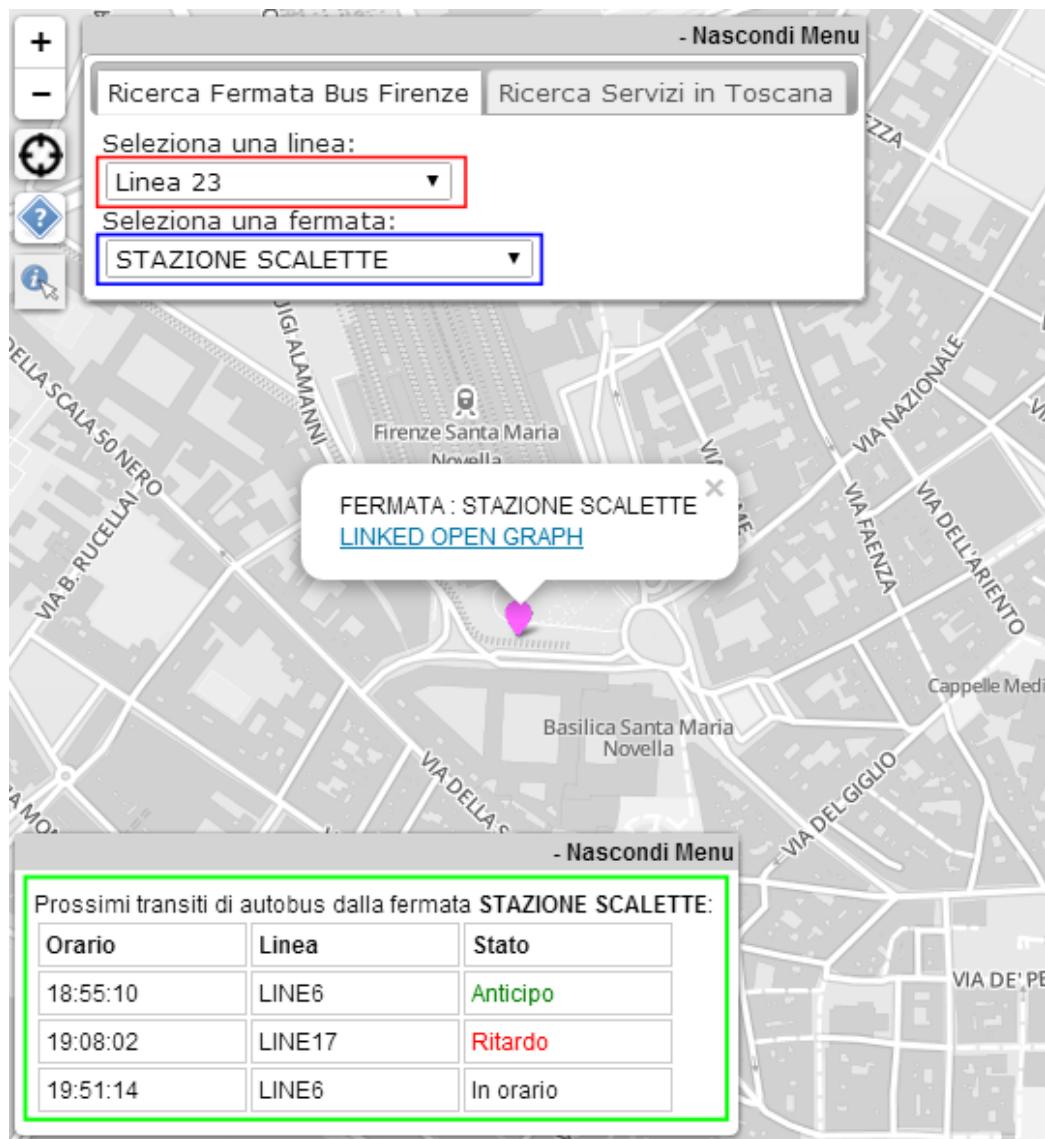


Figura 5.14. Selezione di una fermata dell'autobus e visualizzazione dei corrispondenti dati AVM

A partire dalla fermata selezionata, è possibile effettuare una ricerca di tutti i servizi di interesse presenti in un certo raggio. Per selezionare le categorie di servizi di interesse, si devono spuntare le caselle corrispondenti nel menu di destra. Prima di effettuare la ricerca, è possibile selezionare anche il numero massimo di risultati ottenibili e la distanza massima dalla fermata. In figura 5.15 è mostrato il procedimento di filtraggio dei possibili

risultati.

- Nascondi Menu

Selezione Attuale:
Fermata Bus: STAZIONE SCALETTE

Cerca Attività

Tipo Servizio:

- De/Selezione 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

Raggio di Ricerca:
Entro 300 metri

Numero massimo di risultati:
200

Cerca! **Pulisci**

Leaflet | Map data © 2011 OpenStreetMap contributors, Imagery © 2012 Cloudmade

Figura 5.15. Selezione delle categorie di servizi di interesse, del numero massimo di risultati e della distanza massima a cui ricercare i servizi

Visualizzazione tramite ServiceMap

Cliccando sul pulsante “*Cerca!*”, infine, si otterranno i servizi richiesti, accompagnati da un messaggio di avviso che informa di quanti servizi siano stati effettivamente recuperati. In figura 5.16, è possibile vedere il risultato della ricerca all’interno della mappa.

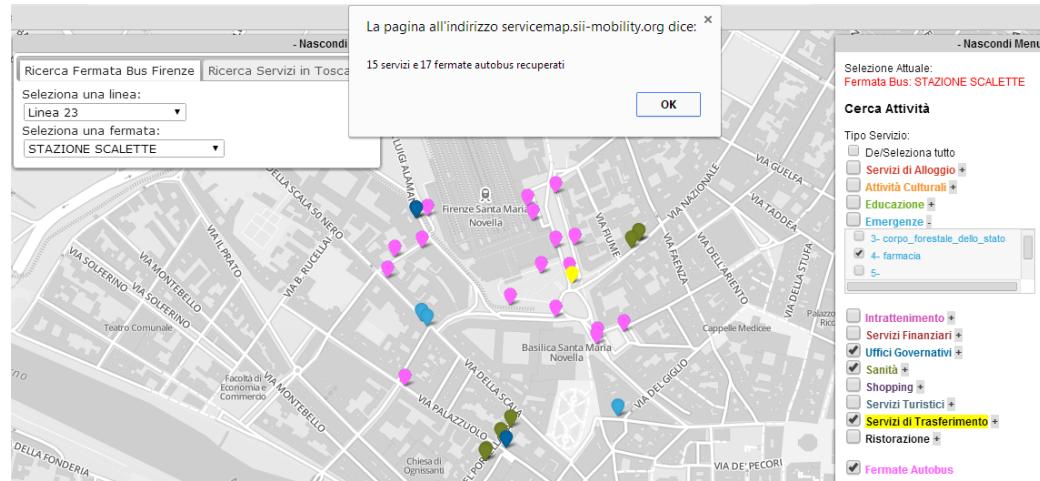


Figura 5.16. Risultati della ricerca di Servizi prossimi ad una Fermata Autobus

La query SPARQL per una ricerca come quella appena descritta è mostrata in figura 5.17.

```

1 PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
2 PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
3 PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
4 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
5 PREFIX vcard:<http://www.w3.org/2006/vcard/ns#>
6 PREFIX omgeo:<http://www.ontotext.com/owlim/geo#>
7 SELECT distinct ?ser ?serAddress ?elat ?elong ?sName ?email ?note
8 WHERE {
9     ?ser rdf:type SiiMobility:Service .
10    ?ser SiiMobility:serviceCategory "farmacia"^^xsd:string .
11    ?ser vcard:organization-name ?sName .
12    ?ser vcard:street-address ?serAddress .
13    ?ser SiiMobility:hasAccess ?entry .
14    ?entry geo:lat ?elat .
15    ?entry geo:long ?elong .
16    ?entry omgeo:nearby(43.7754868 11.2480146 "0.3km") .
17    OPTIONAL {?ser vcard:note ?note} .
18    OPTIONAL {?ser SiiMobility:email ?email} .
19 } LIMIT 200

```

Figura 5.17. Query SPARQL per la ricerca dei Servizi di tipo FARMACIA prossimi alla fermata STAZIONE SCALETTE

Le prime 6 righe servono per determinare i namespaces. La riga 7 specifica quali campi si vogliono ottenere in risposta alla query. In particolare, sono richiesti i campi **?ser**, che contiene la URI univoca del servizio, **?serAddress**, che ne contiene l'indirizzo, **?elat** ed **?elong** contengono le coordinate geografiche del servizio e servono per poterlo correttamente referenziare sulla mappa, infine **?sName** contiene il nome del servizio (o la ragione sociale, se commerciale), **?email** l'indirizzo email e **?note** le eventuali note. Le righe seguenti stabiliscono i filtri: la riga 9 indica che si cercano esclusivamente oggetti di tipo **Service**, la riga 10 specifica che i servizi devono essere della tipologia *“farmacia”*. Con la riga 11 si estrae il nome del servizio, mentre con la 12 l'indirizzo completo. La riga 13 filtra esclusivamente i servizi che abbiano la proprietà *hasAccess* che li collega ad un Accesso, e le coordinate di tale accesso (prelevate alle righe 14 – 15), vengono utilizzate alla riga 16 dall'istruzione **omgeo:nearby**, che permette il filtraggio geografico in base alla distanza dal punto centrale. Il punto centrale (*43.7754868, 11.2480146*) corrisponde alle coordinate esatte della fermata dell'autobus denominata *“STAZIONE SCALETTE”*. L'ultimo parametro dell'istruzione

`omgeo:nearby` indica che si stanno cercando Servizi in un raggio di 300 metri dalla fermata. Infine, alle righe 17 e 18, si estraggono, se presenti, le DataProperties `vcard:note` e `email`. Con l'istruzione `LIMIT 200`, si impone a 200 il numero massimo di servizi visualizzabili.

Una tabella contenente una parte dei risultati di questa query, è visualizzata in figura 5.18.

ServiceAddress	Elat	Elong	ServiceName	Service Type
VIA DEI BANCHI, 18/R	43.7736090	11.2505591	DEI BANCHI	farmacia
VIA DELLA SCALA, 61	43.7752329	11.2459294	DELLA SCALA	farmacia
VIA DELLA VIGNA NUOVA, 54/R	43.7711800	11.2492657	SAN GIORGIO	farmacia
PIAZZA DEGLI OTTAVIANI, 8/R	43.7726956	11.2494195	CAMILLI	farmacia

Figura 5.18. Dati recuperati dalla query SPARQL di figura 5.17

In figura 5.19, è mostrata una parte dei risultati della query convertita in formato JSON.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "geometry": {
        "type": "Point",
        "coordinates": [
          11.2505591461,
          43.7736090236
        ]
      },
      "type": "Feature",
      "properties": {
        "popupContent": "DEI BANCHI - farmacia",
        "nome": "DEI BANCHI",
        "tipo": "farmacia",
        "email": "",
        "note": "",
        "serviceId": "http://www.disit.dinfo.unifi.it/SiiMobility/DEI\_BANCHI-VIA\_DEI\_BANCHI\_18/R",
        "indirizzo": "VIA DEI BANCHI, 18/R"
      },
      "id": 1
    }
  ]
}
```

Figura 5.19. Esempio di dati JSON corrispondenti ai risultati della query di figura 5.17

I servizi richiesti sono quindi disposti sulla mappa e la stessa viene centrata e zoomata, affinché sia si possano apprezzare più definitamente i risultati. In figura 5.20, è possibile vedere il risultato della query descritta in precedenza.

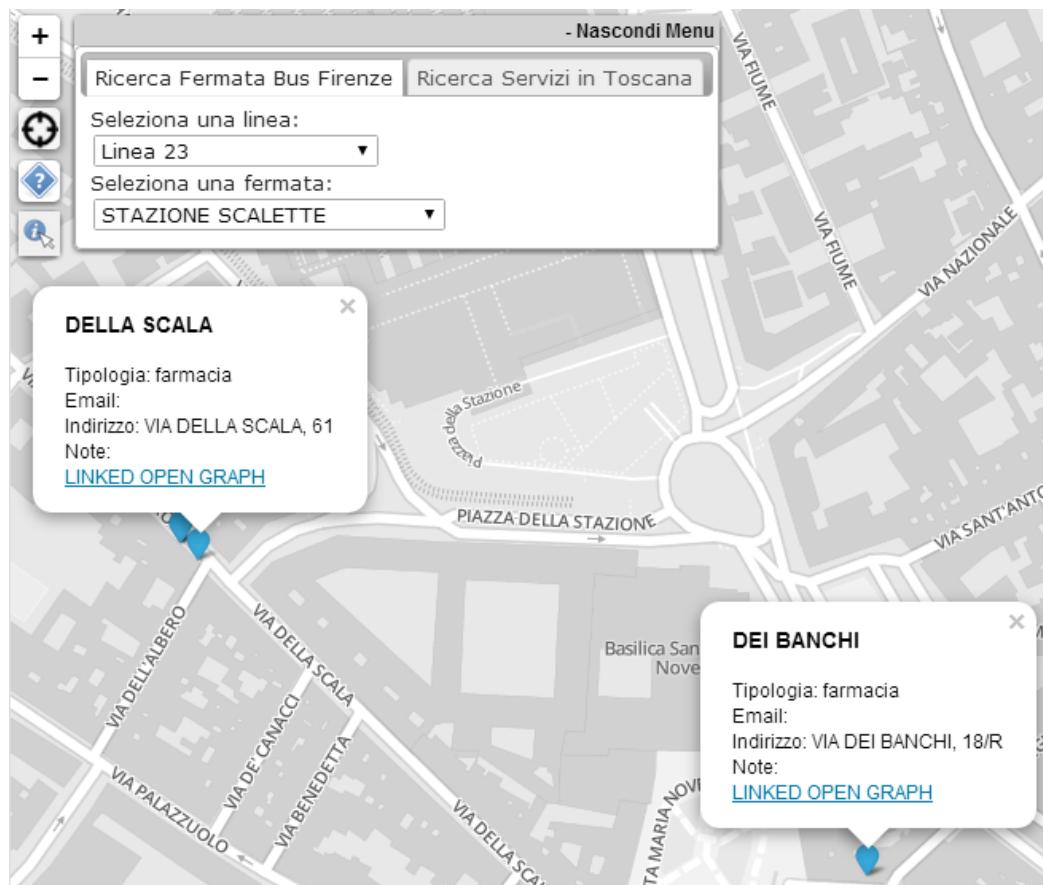


Figura 5.20. Visualizzazione su mappa dei risultati ottenuti in risposta alla query di figura 5.17

Il diagramma di sequenza tra lato client e lato server della procedura appena descritta è mostrato in figura 5.21.

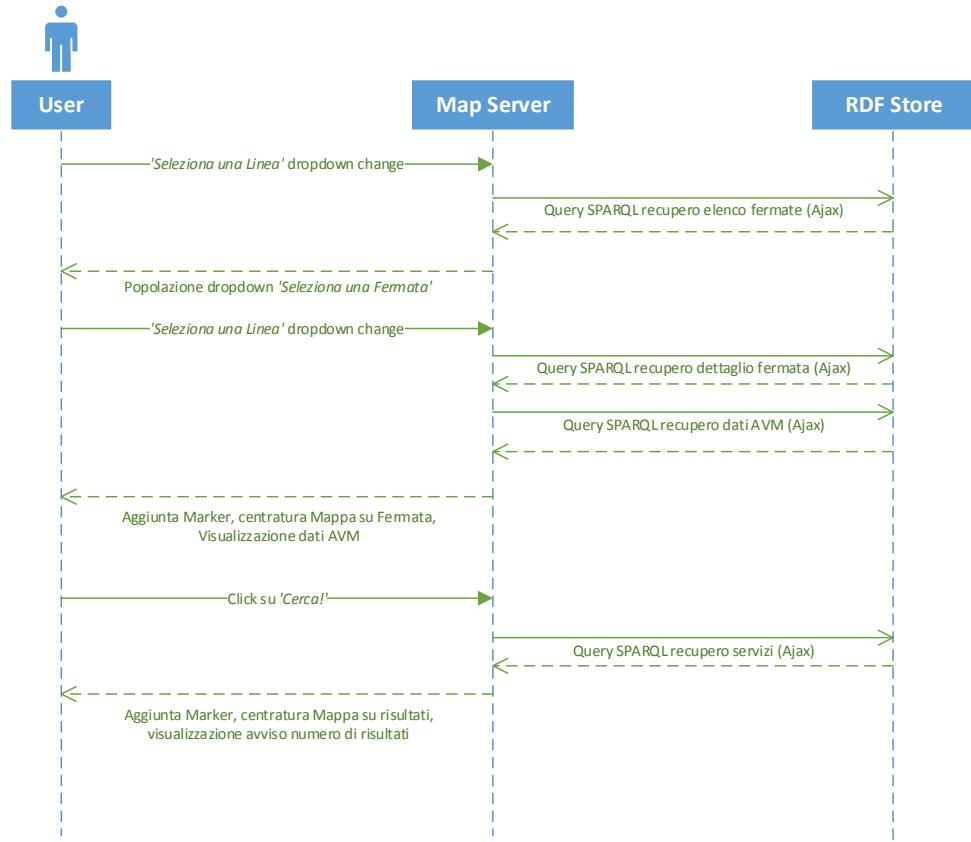


Figura 5.21. Diagramma di sequenza client-server per il caso d'uso numero 2

Come accennato all'inizio di questa sezione, nel momento in cui l'utente seleziona la fermata di base per la ricerca dei Servizi circostanti, vengono richiesti al repository anche i dati AVM di quella particolare fermata. La query per il recupero di questi dati è abbastanza complessa, e prevede due step. La prima query è mostrata in figura 5.22.

```

1 PREFIX foaf:<http://xmlns.com/foaf/0.1/>
2 PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
3 PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
4 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
5 PREFIX time:<http://www.w3.org/2006/time#>
6 SELECT distinct ?avmRecord ?tplLine ?ride
7 WHERE{
8     ?busStop rdf:type SiiMobility:BusStop .
9     ?busStop foaf:name "STAZIONE SCALETTE"^^xsd:string .
10    ?busStop SiiMobility:hasForecast ?busStopForecast .
11    ?avmRecord SiiMobility:include ?busStopForecast .
12    ?avmRecord SiiMobility:concern ?tplLine .
13    ?ride SiiMobility:hasSurvey ?avmRecord .
14    ?avmRecord SiiMobility:hasLastStopTime ?time .
15    ?time time:inXSDDateTime ?timeInstant .
16 }
17 ORDER BY DESC (?timeInstant)
18 LIMIT 10

```

Figura 5.22. Query SPARQL per la prima fase di recupero di dati AVM sulla fermata STAZIONE SCALETTE

La query prevede di restituire le ultime 10 combinazioni univoche di **?avmRecord**, **?line** e **?ride** relative alla fermata selezionata. Attraverso i molteplici collegamenti che legano le entità facenti parte delle aree di **Trasporto Pubblico Locale** e di **Sensoristica (AVM)**, si estraggono gli oggetti corretti (la descrizione approfondita di queste classi e delle proprietà è già stata fornita nei capitoli 3.1.2.3 e 3.1.2.5). Tramite le ultime due righe di filtraggio (14 e 15), si preleva anche l'informazione temporale relativa alla data di generazione di ciascun report AVM. Infine, alle righe 17 e 18, si ordinano i risultati in base alla data decrescente e si mantengono esclusivamente gli ultimi 10 report in ordine temporale. Il risultato di questa prima query è mostrato in figura 5.23.

AvmRecord	TplLine	Ride
2014-03-07T17:58:25.4584782+01:00+4737229	LINE23	4737229
2014-03-07T17:58:25.4584782+01:00+4764073	LINE17	4764073
2014-03-07T17:58:25.4584782+01:00+4764311	LINE17	4764311
2014-03-07T17:58:25.4584782+01:00+4737165	LINE23	4737165
2014-03-07T17:58:25.4584782+01:00+4764055	LINE17	4764055
2014-03-07T17:58:25.4574781+01:00+4737239	LINE23	4737239
2014-03-07T17:58:25.4584782+01:00+4737032	LINE23	4737032
2014-03-07T17:58:25.4584782+01:00+4737114	LINE23	4737114
2014-03-07T17:58:25.4574781+01:00+4764259	LINE17	4764259
2014-03-07T17:50:25.2114583+01:00+4764311	LINE17	4764311

Figura 5.23. Primi risultati della query di figura 5.22

I distinti AVMRecord ottenuti dalla prima query compongono la seconda query, che ricerca l'orario di presunto arrivo del Bus per ogni particolare AVMRecord univoco dei 10 registrati. La seconda query è mostrata in figura 5.24.

```

1 PREFIX foaf:<http://xmlns.com/foaf/0.1/>
2 PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
3 PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
4 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
5 PREFIX time:<http://www.w3.org/2006/time#>
6 SELECT DISTINCT ?expectedTimeInstant ?tpline ?rideState
7 WHERE {
8   ?busStop rdf:type SiiMobility:BusStop .
9   ?busStop foaf:name "STAZIONE SCALETTE"^^xsd:string .
10  ?busStop SiiMobility:hasForecast ?forecast .
11  (
12    <http://www.disit.dinfo.unifi.it/SiiMobility/2014-03-06T16:02:13.9528154+01:00+4764309> SiiMobility:include ?forecast .
13    <http://www.disit.dinfo.unifi.it/SiiMobility/2014-03-06T16:02:13.9528154+01:00+4764309> SiiMobility:concern ?tpline .
14    <http://www.disit.dinfo.unifi.it/SiiMobility/2014-03-06T16:02:13.9528154+01:00+4764309> SiiMobility:rideState ?rideState .
15  )
16  UNION
17  (
18    <http://www.disit.dinfo.unifi.it/SiiMobility/2014-03-06T16:01:14.1308338+01:00+4764137> SiiMobility:include ?forecast .
19    <http://www.disit.dinfo.unifi.it/SiiMobility/2014-03-06T16:01:14.1308338+01:00+4764137> SiiMobility:concern ?tpline .
20    <http://www.disit.dinfo.unifi.it/SiiMobility/2014-03-06T16:01:14.1308338+01:00+4764137> SiiMobility:rideState ?rideState .
21  )
22  ?forecast SiiMobility:hasExpectedTime ?expectedTime .
23  ?expectedTime time:inXSDDateTime ?expectedTimeInstant .
24  FILTER (xsd:dateTime(?expectedTimeInstant) >= now()) .
25 }
26 ORDER BY ASC (?expectedTimeInstant)
27 LIMIT 4

```

Figura 5.24. Query SPARQL per la seconda fase di recupero di dati AVM sulla fermata STAZIONE SCALETTE

Nella query di esempio in figura sono stati inseriti soltanto 2 dei 10 record precedentemente generati. In particolare, essi sono visibili alle righe 11-15 e 17-21. Anche in questa seconda parte si estraggono le **?forecast**, ma poichè sono state precedentemente definite come *distinct*, non si corre il

rischio di mostrare dati duplicati relativi ad una stessa corsa. Alla riga 24 si filtrano esclusivamente le previsioni che risultano successive alla data e ora attuali, mentre le ultime righe di questa query (26 e 27) filtrano, come in precedenza, le previsioni ottenute in ordine cronologico (ma in questo caso l'ordine è impostato ascendente) e ne estraggono le prime 4. In questo modo, si recuperano le previsioni di transito in un ordine che mostra, come primo risultato, la previsione più prossima all'orario attuale.

I dati correttamente recuperati, a questo punto, sono inseriti nel riquadro contestuale dell'applicazione **ServiceMap** e mostrati all'utente, come mostrato in figura 5.25. I dati

Orario	Linea	Stato
17:54:14	LINE23	Ritardo
17:59:06	LINE17	In orario
18:06:24	LINE23	In orario
18:14:12	LINE17	Ritardo

Figura 5.25. Visualizzazione su ServiceMap delle previsioni di transito sulla fermata autobus FRATELLI ROSSELLI

5.1.3 Caso d'uso 3: ricerca Servizi prossimi ad un punto generico

In realtà, la ricerca di Servizi può essere generalizzata, e passare da una ricerca circostante ad una fermata del bus, ad una ricerca nei dintorni di un punto generico. In particolare, è possibile ricercare servizi attorno a:

- La propria posizione GPS, cliccando sul pulsante corrispondente in alto a sinistra.
- Un qualsiasi servizio ottenuto con una precedente ricerca, cliccando sul marker corrispondente.

- Un punto qualsiasi della mappa, cliccando sul pulsante corrispondente in alto a sinistra.

La ricerca di servizi per questo caso d'uso è assimilabile alla ricerca vista nel paragrafo 5.1.2. La differenza rispetto al metodo precedente è nel punto centrale della ricerca, ma le query SPARQL di ricerca dei servizi rimangono pressochè inalterate. Tuttavia, contestualmente alla ricerca di servizi di questo tipo, si possono ottenere informazioni aggiuntive di interesse, non descritte in precedenza. In particolare, vi sono due ricerche contestuali importanti:

- Il controllo dell'occupazione di un determinato **Parcheggio auto**.
- Una procedura di geocoding inversa che, a partire da un punto qualsiasi sulla mappa, restituisce il suo **indirizzo approssimativo**.

Per ottenere all'interno dell'applicazione i dati di occupazione aggiornati di un determinato parcheggio, è sufficiente cliccare sul marker corrispondente sulla mappa. In questo modo il sistema richiede al repository di fornire i dati dell'ultimo aggiornamento e li mostra nel menu contestuale in basso a sinistra come mostrato in figura 5.26.



Figura 5.26. Dati sull'occupazione del parcheggio G.Guerra di Empoli

La query che permette il recupero di questi dati è mostrata in figura 5.27.

```

1 PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
2 PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
3 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4 PREFIX vcard:<http://www.w3.org/2006/vcard/ns#>
5 PREFIX time:<http://www.w3.org/2006/time#>
6 SELECT distinct ?situationRecord ?instantDateTime ?free ?occupied ?capacity
7 WHERE {
8     ?park rdf:type SiiMobility:TransferService .
9     ?park vcard:organization-name "Stazione Binario 16" .
10    ?carParkSensor SiiMobility:observe ?park .
11    ?carParkSensor SiiMobility:capacity ?capacity .
12    ?situationRecord SiiMobility:relatedTo ?carParkSensor .
13    ?situationRecord SiiMobility:observationTime ?time .
14    ?time time:inXSDDateTime ?instantDateTime .
15    ?situationRecord SiiMobility:free ?free .
16    ?situationRecord SiiMobility:occupied ?occupied .
17 }
18 ORDER BY DESC (?instantDateTime)
19 LIMIT 1

```

Figura 5.27. Query per il recupero dell'ultima situazione conosciuta riguardante l'occupazione del parcheggio Stazione Binario 16

La query recupera le seguenti informazioni: l'URI dell'ultimo **?situationRecord**, l'istante in cui è stato generato (**?instantDateTime**) e i dati veri e propri di occupazione, cioè **?free**, **?occupied** e **?capacity**, ovvero rispettivamente il numero di posti liberi, di posti occupati e la capienza massima del parcheggio.

La seconda funzionalità attivabile in questo caso d'uso, è la ricerca dell'indirizzo approssimativo di un punto sulla mappa. Cliccando sul corrispondente pulsante in alto a sinistra, si attiva la ricerca. Cliccando successivamente su un punto qualsiasi della mappa, si ottengono le informazioni desiderate nel menu contestuale in basso a sinistra. Un esempio di questa procedura è mostrato in figura 5.28.



Figura 5.28. Esempio di ricerca di un indirizzo approssimativo di un punto sulla mappa

La query SPARQL di ricerca dell'indirizzo approssimativo è simile a quella descritta per la riconciliazione delle Fermate degli Autobus con la propria Road (si veda il capitolo 4.3). Viene mostrata di seguito in figura 5.29, e successivamente ne vengono descritte le principali differenze rispetto a quella appena citata.

```

1 PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
2 PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
3 PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
4 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
5 PREFIX vcard:<http://www.w3.org/2006/vcard/ns#>
6 PREFIX omgeo:<http://www.ontotext.com/owlim/geo#>
7 PREFIX foaf:<http://xmlns.com/foaf/0.1/>
8 SELECT distinct ?extendName ?extendNumber ?municipalityName
9 WHERE {
10     ?entry rdf:type SiiMobility:Entry .
11     ?streetNumber SiiMobility:hasExternalAccess ?entry .
12     ?streetNumber SiiMobility:extendNumber ?extendNumber .
13     ?streetNumber SiiMobility:belongTo ?road .
14     ?road SiiMobility:extendName ?extendName .
15     ?entry geo:lat ?elat .
16     ?entry geo:long ?elong .
17     ?road SiiMobility:inMunicipalityOf ?municipality .
18     ?municipality foaf:name ?municipalityName .
19     ?entry omgeo:nearby(43.7712673 11.256777 "0.1km") .
20     BIND( omgeo:distance(?elat, ?elong, 43.7712673, 11.256777) AS ?distance) .
21 }
22 ORDER BY ?distance
23 LIMIT 1

```

Figura 5.29. Query SPARQL per la ricerca di un indirizzo approssimativo a partire da un punto sulla mappa

In entrambe le query si esegue una sorta di ricerca *nearest neighbor* all'interno di un cerchio di raggio 100 metri. La query di riconciliazione precedente restituisce esclusivamente l'ID della **Road** e la distanza rispetto alla fermata dell'accesso corrispondente. In questo caso, invece, si eseguono delle operazioni aggiuntive perché i dati di interesse sono in numero maggiore. In particolare, si richiede l'**?extendName** della strada, l'**extendNumber** del numero civico ed il nome del comune (**?municipalityName**). In questo modo, concatenando tali informazioni si riesce ad ottenere l'indirizzo completo che si andava ricercando.

5.2 Visualizzazione tramite Linked Open Graph

A partire dalla mappa descritta nella sezione precedente, cliccando su uno dei marker corrispondenti ad un servizio o ad una fermata dell'autobus, si apre un piccolo popup, come mostrato in figura 5.2. Cliccando successivamente sul link “*LINKED OPEN GRAPH*” presente all'interno del popup,

Visualizzazione tramite Linked Open Graph

è possibile visualizzare i dati legati all'oggetto di interesse, attraverso una visualizzazione di tipo **Faceted Graph**. Tale termine indica un approccio per la visualizzazione di dati, memorizzati in formato RDF, di tipo **Graph-Based**, collegato a tecniche di filtraggio di tipo **Faceted Search**. Sempre all'interno del laboratorio DISIT, è stato sviluppato un software, chiamato **LOG - Linked Open Graph** ([8] e [9]), che permette la visualizzazione di triplestores o di dati in formati *Linked Data* seguendo l'approccio di tipo *Faceted*. In figura 5.30 è mostrato un esempio di visualizzazione di questo tipo a partire da una fermata dell'autobus.

Visualizzazione tramite Linked Open Graph

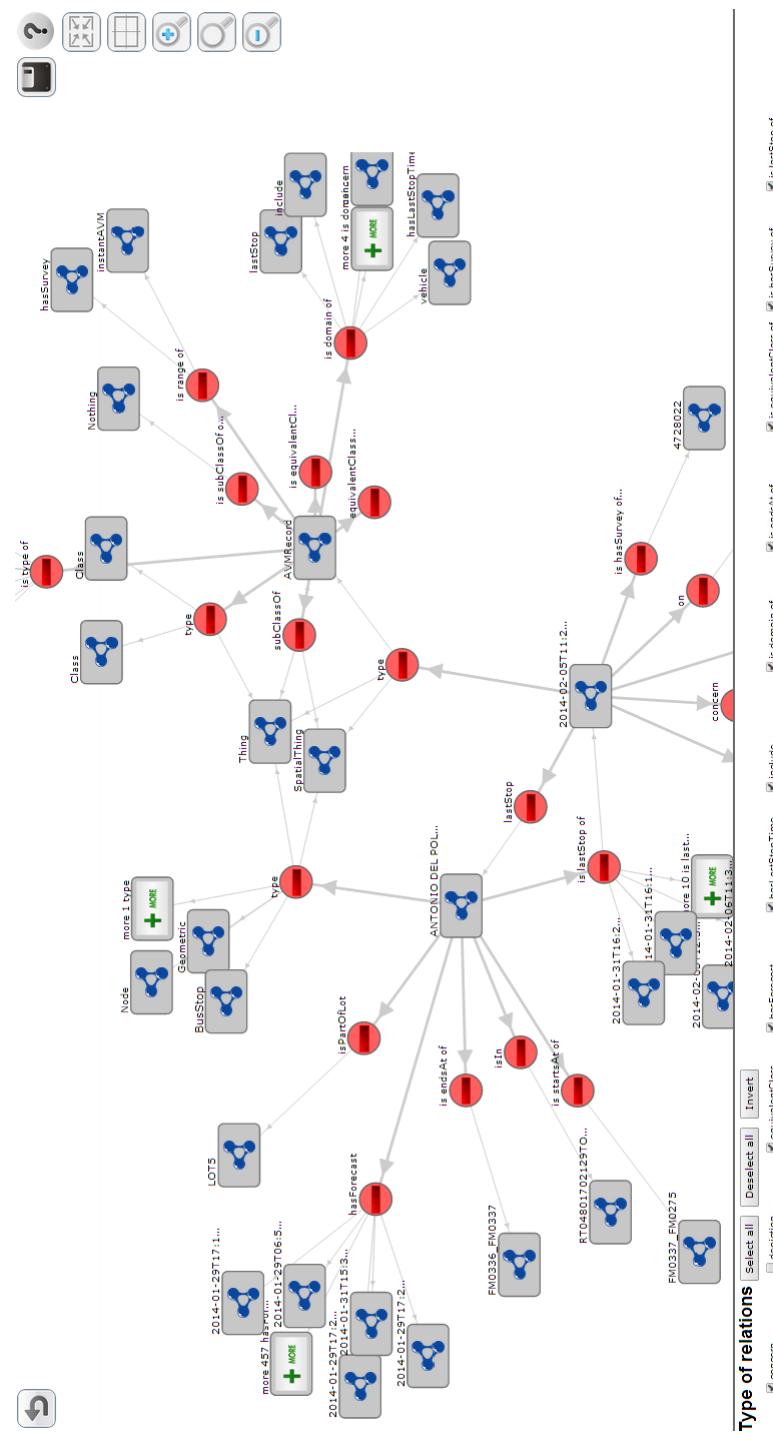


Figura 5.30. Esempio di visualizzazione tramite il software LOG dei dati RDF a partire dalla fermata dell'autobus FM0275 - ANTONIO DEL POLLAIOLI 05

Più precisamente, **LOG** è uno strumento web di navigazione che permette di esplorare servizi SPARQL di tipo Linked Data tramite il loro endpoint. Il grafico viene popolato attraverso nodi e archi. I nodi possono essere di più tipi, ovvero possono contenere risorse (rettangoli), oppure relazioni tra risorse o ObjectProperties (cerchi). Attraverso gli archi si collegano gli elementi e le loro relazioni.

Tramite questo strumento è possibile navigare il grafo RDF, partendo da una risorsa ed esplorando il grafo seguendo le risorse collegate. Cliccando su un nodo viene visualizzato un menu di navigazione che permette molte operazioni, tra le quali:

- Esplorazione o collassamento di un nodo nel grafo.
- Centratura della visuale su un particolare nodo.
- Apertura e visualizzazione dei dettagli del nodo (DataProperties).
- Filtraggio delle relazioni.

Inoltre, sono presenti alcune funzionalità per il salvataggio del grafo attualmente visualizzato e delle preferenze utente, per la personalizzazione nella visualizzazione di determinate risorse o relazioni, per la condivisione della ricerca, eccetera. L'applicazione **LOG**, è reperibile in rete all'indirizzo: <http://log.disit.org/service/>. Come detto, questo software è stato realizzato dal laboratorio DISIT. Ai fini di questa tesi, il suo utilizzo è servito per avere una evidenza visuale della correttezza dei dati inseriti. Inoltre, questo strumento fornisce, ad un utente esperto, un metodo alternativo di navigazione del grafo RDF.

Capitolo 6

Conclusioni e sviluppi futuri

Il progetto di questa tesi ha richiesto un impegno notevole, ma al termine del lavoro, ha permesso di raggiungere un ottimo risultato. I dati provenienti da molteplici fonti (Regione Toscana, Comune di Firenze, MIIC, eccetera) sono stati correttamente elaborati. L'ontologia **SmartCity Ontology** costruita permette di gestire dati molto specifici ma, al tempo stesso, ha mantenuto una sufficiente generalità per poter essere ampliata ed esportata al contesto di interoperabilità a cui mira il progetto Sii-Mobility.

Le procedure ETL create per la manipolazione dei dati funzionano correttamente, anche in presenza di moli di dati notevoli. L'intero processo di Ingestion richiede del tempo quando viene eseguito sull'intero grafo stradale, ma in considerazione delle dimensioni dei dati originali il risultato raggiunto è molto positivo. Grazie alle performance dei software per lo storage e l'interrogazione di repository RDF, attraverso le query SPARQL e le interfacce di visualizzazione create, è possibile recuperare tutte le risorse di interesse in modo semplice e veloce.

6.1 Risultati

Al termine della stesura di questa tesi, la quantità di dati inserita nel repository attualmente in funzione è particolarmente interessante. Di seguito sono presentati alcuni dati numerici di rilievo.

L'intero **Grafo Stradale** occupa un numero di triple pari a circa **68.000.000**. Sempre per quanto riguarda il solo Grafo Stradale, il numero di entità univoci delle classi maggiormente di interesse è elencato di seguito:

- Oggetti di classe **Road** univoci: **137.745**
- Oggetti di classe **RoadElement** univoci: **393.606**
- Oggetti di classe **Node** univoci: **321.611**
- Oggetti di classe **StreetNumber** univoci: **1.432.223**
- Oggetti di classe **Entry** univoci: **1.474.050**

I dati provenienti dalle altre macroaree dell'ontologia hanno contribuito ad incrementare giornalmente il numero di triple. Nell'arco di pochi giorni le triple complessive nel repository sono arrivate a **81.339.192** (dato relativo all'ultimo conteggio effettuato in data 14/03/2014).

La sezione del **Trasporto Pubblico Locale** conta **2.366** fermate di autobus, soltanto nell'area metropolitana fiorentina. Il numero di linee di autobus differenti salvate è pari a **85**, ma le linee *attive*, cioè le linee delle quali si hanno informazioni su percorsi e dati AVM, sono soltanto **5**. Questo, tuttavia, non ha impedito di generare complessivamente **48.693** AVMRecord, composti da **511.815** BusStopForecast distinte.

Ogni giorno, vengono inseriti 2 WeatherReport per ognuno dei **286** comuni presenti in Toscana. Questo comporta, allo stato attuale, un numero di **8.512** bollettini attualmente inseriti nel repository, mentre il numero di Previsioni Meteo si attesta su **135.472**.

Per quanto riguarda il processo di riconciliazione strade, si sono generate **188.992** triple contenenti la DataProperty *dc:alternative* su oggetti di tipo Road.

Grazie ai dati provenienti dalla Regione Toscana, al momento nel repository sono stati inseriti **30.182 Servizi**, appartenenti alle 12 macrocategorie descritte in precedenza. L'intera procedura di riconciliazione dei servizi ha prodotto i risultati riportati in tabella 6.1.

Metodo	Triple <i>hasAccess</i> <i>create</i>	Triple <i>isIn</i> create
Ricerca Esatta	7.325	15.300
Ricerca dell'ultima Parola	5.160	5.415
Ricerca tramite Geocoding	552	492
Altre ricerche (numeri civici con caratteri strani, toponimi con caratteri strani, comuni con denominazione errata, eccetera)	148	105
Totali	13.185	21.207

Tabella 6.1. Risultati dei singoli processi di riconciliazione dei servizi e risultati totali

Per quanto riguarda la riconciliazione delle fermate degli autobus, attualmente sono state generati **2.067** nuovi collegamenti *isIn* tra **BusStop** e **Road**, sulle **2.366** fermate presenti.

6.2 Sviluppi Futuri

Il lavoro svolto rappresenta soltanto una prima parte dell'intero percorso da svolgere per questa sezione del progetto Sii-Mobility. Nel prossimo futuro, sono possibili molti sviluppi, tutti rivolti nella direzione di contribuire a migliorare i trasporti e la mobilità in Toscana e di fornire un servizio che fornisca informazioni utili ai principali attori del sistema, quali ad esempio le imprese o i privati cittadini.

Il prossimo step, già predisposto, sarà quello di inserire molti altri nuovi dati provenienti da fonti nuove o già conosciute, e di integrarli ed accorparli con i dati attualmente esistenti. Tra i dati che sarà possibile inserire a breve, è necessario citare:

- Dati delle singole coordinate degli elementi stradali.
- Dati del trasporto pubblico, sia municipale, su comuni differenti da *FIRENZE*, che regionale. Inoltre, sarà vitale ottenere l'accesso ai dati completi degli AVM, per servire un servizio real-time sull'intera rete degli autobus.
- Dati più dettagliati riguardanti i sensori del traffico.
- Dati relativi ad un numero più elevato di parcheggi auto, e più dettagliati.
- Dati relativi ad attività commerciali e di servizio presenti in Toscana.

In aggiunta a questi dati, per i quali è già stata predisposta correttamente l'ontologia, sarà necessario aggiungerne altri, quali ad esempio i dati della rete ferroviaria toscana, oppure i molti dati facenti parte del cosiddetto *Data Pack Accessorio* del Grafo Stradale. Tra questi, si possono citare i dati relativi agli incroci semaforizzati oppure ad altre regole di accesso che definiscano orari di accesso o mezzi consentiti. In questo caso, sarà necessario ingrandire o ritoccare l'ontologia, pur mantenendone la struttura di base attuale.

Un altro aspetto fortemente migliorabile è quello della riconciliazione, sia quella sui Toponimi (sezione 4.1), che quella sui Servizi (sezione 4.2). In prima battuta, si potrebbe pensare di ingrandire la tabella di generazione dei toponimi alternativi (ad esempio con un elenco di nomi propri e relative iniziali), ma in un'ottica più ampia, dovrà essere necessariamente rivista la procedura di *Address Matching*, per fornire risultati più affidabili e risposte in un contesto più generale.

Per quanto riguarda la visualizzazione dei dati, sicuramente, dovranno essere implementate molte novità nella web application contenente la mappa interattiva. Dovrà essere svincolato il suo utilizzo dai casi d'uso previsti in questa tesi e, di conseguenza, avere un comportamento maggiormente di tipo general purpose, nel quale l'utente possa personalizzare al massimo la propria esperienza d'uso. Inoltre, sarà necessaria una localizzazione della mappa in lingue differenti dall'italiano, per poter essere utilizzata anche da persone di altre nazionalità.

Infine, per seguire le linee guida dei Linked Data, sarà utile collegare i dati del repository con altri dati, sempre facenti parte del mondo Linked Data (ad esempio Geonames, DBpedia, eccetera).

Appendice

A

Web Semantico

Web Semantico è un termine coniato da Tim Berners-Lee [10], considerato l'inventore del moderno *World Wide Web*. Esso vuole indicare l'evoluzione che dovrebbe trasformare l'attuale web in un ambiente in cui, ai documenti attualmente pubblicati, vengano associate delle informazioni e dei *metadati*, grazie ai quali si potrà associare un **contesto semantico** ai dati, che permetta la loro elaborazione automatica da parte di software e macchine.

Ad oggi, infatti, la maggior parte delle informazioni presenti su Internet è disponibile in formato di documenti (pagine HTML, Immagini, Video) che sono facilmente interpretabili dagli umani, ma che, a causa della loro eterogeneità, non sono facilmente processabili in maniera automatica.

Le tecnologie fondamentali per la costruzione del Web Semantico sono molteplici: RDF, RDF Schema, OWL, SPARQL, e sono descritte in dettaglio nei prossimi paragrafi.

A.1 RDF - Resource Description Framework

Il Web Semantico ha come base il **Resource Description Framework** (RDF). RDF è uno strumento proposto dal W3C¹ che definisce lo standard per la codifica di relazioni tra informazioni e la loro interoperabilità seguendo i principi della logica dei predicati. Si basa sul linguaggio **XML**, e l'entità di base per una informazione è la cosiddetta **tripla**, una istruzione formata da *Soggetto*, *Predicato* e *Oggetto* (o *Valore*). Il *Soggetto* deve necessariamente essere una **Risorsa**, unità fondamentale del modello, che, a sua volta, deve poter essere univocamente identificata attraverso una **URI (Universal Resource Identifier)**. Tipicamente, le URI delle Risorse RDF vengono definite come indirizzi Web, quindi associati al protocollo HTTP, ad esempio <http://www.disit.dinfo.unifi.it/SiiMobility/RT04801700001ES>. Il *Predicato* è obbligatoriamente espresso da una **Proprietà**. Una *Proprietà RDF* è una relazione che associa una *Risorsa* ad un'altra *Risorsa* oppure ad un determinato *Valore*. Anche le *Proprietà* devono necessariamente essere identificate tramite URI, ad esempio <http://www.disit.dinfo.unifi.it/SiiMobility#isPartOf>. Infine, l'*Oggetto* di una tripla, in base alla proprietà espressa dal *Predicato* può essere un'altra *Risorsa* oppure un *Valore*, cioè un tipo di dato primitivo, come una stringa, un intero, un booleano, eccetera. Nel Web Semantico tutte le informazioni sono codificate da sequenze di queste triple. Ad esempio è possibile esprimere il concetto che l'entità con identificativo RT04801700001ES è *di tipo* Elemento Stradale e *fa parte* dell'oggetto RT04801700001TO che a sua volta è *di tipo* Toponimo tramite le seguenti triple:

```
<http://www.disit.dinfo.unifi.it/SiiMobility/RT04801700001ES> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.disit.dinfo.unifi.it/SiiMobility/RoadElement> .
<http://www.disit.dinfo.unifi.it/SiiMobility/RT04801700001ES> <http://www.disit.dinfo.unifi.it/SiiMobility#isPartOf>
<http://www.disit.dinfo.unifi.it/SiiMobility/RT04801700001TO> .
<http://www.disit.dinfo.unifi.it/SiiMobility/RT04801700001TO> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.disit.dinfo.unifi.it/SiiMobility#Road> .
```

Figura A.1. Esempio di informazioni in formato di triple RDF

RDF permette molteplici sintassi per la serializzazione dei dati. La sintassi mostrata in figura A.1 è chiamata **N3** (o **N-Triples**). Essa permette

¹World Wide Web Consortium - <http://www.w3.org/>

una rapida visualizzazione delle informazioni con lo svantaggio di generare, talvolta, una certa ridondanza nelle informazioni. Molte altre sintassi sono state proposte, tra cui **RDF/XML** (che è la sintassi utilizzata comunemente per la definizione di ontologie), Turtles, Notation3, eccetera.

Utilizzando come base le triple RDF è possibile definire un livello di astrazione superiore. In particolare, è possibile modellare concetti come *Classi*, *Proprietà*, *Sottoclassi*, eccetera. Questo è ottenuto tramite il cosiddetto **RDFS** (o **RDF-Schema**). RDFS è un vocabolario generico che permette la definizione di specifici vocabolari fornendo un sistema di tipi semantici per RDF. Nell'esempio in figura A.1 è stata definita la Classe **Road**. La classe **Road** secondo RDFS è una risorsa che ha una relazione *rdf:type* con la risorsa <http://www.w3.org/2000/01/rdf-schema#Class>. La proprietà *isPartOf*, per il vocabolario RDFS è una istanza della Classe **Property** (<http://www.w3.org/2000/01/rdf-schema#Property>). Tramite le possibilità fornite da RDFS è quindi possibile delineare un contesto dentro il quale saranno inseriti i dati di interesse.

A.2 Ontologie e OWL - Ontology Web Language

Il termine Ontologia, nel campo dell'informatica, è definito come “*una rappresentazione formale, condivisa ed esplicita di una concettualizzazione di un dominio di interesse*”. Fondamentalmente, una ontologia è una base di conoscenza a partire dalla quale è possibile, per i moderni software, estrarre informazioni, risolvere problemi e addirittura generare nuova conoscenza mediante un procedimento detto di inferenza. Dal punto di vista logico, una ontologia è una teoria assiomatica del primo ordine esprimibile attraverso una logica descrittiva.

Per quanto riguarda il Web Semantico, lo strumento utilizzato per definire una ontologia è **OWL (Ontology Web Language)**. OWL è una famiglia di linguaggi nata come estensione di RDF e serializzata in formato RDF/XML che permette la definizione di ontologie utilizzabili dai principali framework

RDF. Esistono molteplici versioni di OWL, ciascuna con uno specifico trade-off tra espressività e decidibilità.

Sia OWL che RDF-Schema permettono di creare vocabolari o ontologie. Tuttavia, OWL estende l'espressività di RDFS, introducendo alcuni costrutti quali ad esempio le *restrizioni di proprietà*, le *proprietà equivalenti* o *inverse*, l'*identità* espressa tramite l'operatore **owl:sameAs**, eccetera.

A.3 SPARQL - SPARQL Protocol and RDF Query Language

Nell'elenco di tecnologie utilizzate per la costruzione del Semantic Web, non può mancare un linguaggio di interrogazione specifico per collezioni di dati memorizzati in formato RDF. Il linguaggio di riferimento, allo stato attuale è **SPARQL**. Esso permette di recuperare informazioni da un qualsiasi repository semantico basandosi sul concetto di *Pattern Matching* ed in particolare su un costrutto, il *Triple Pattern*, che ricalca la configurazione a triple delle asserzioni RDF, fornendo un modello flessibile per la ricerca di corrispondenze. In estrema sintesi, una query in SPARQL non è altro che un insieme di triple in cui almeno uno tra *Soggetto*, *Predicato* o *Oggetto* è una variabile. Il motore di interrogazione quindi estrapola dal repository di interesse tutte quelle triple il cui grafo si sovrappone alle informazioni richieste. In figura A.2, è mostrato un piccolo esempio di una query SPARQL.

```

1 PREFIX SiiMobility:<http://www.disit.dinfo.unifi.it/SiiMobility#>
2 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 SELECT DISTINCT ?ser ?road
4 WHERE{
5     ?ser rdf:type SiiMobility:Service .
6     ?ser SiiMobility:isIn ?road .
7 }
8 LIMIT 100

```

Figura A.2. Query di esempio SPARQL

Nell'esempio appena mostrato in figura, si ricercano tutte le occorrenze distinte di oggetti di tipo **Service**, che posseggano la proprietà *isIn* che li collega ad altri oggetti di tipo **Road**.

A.4 Open Data - Linked Data

Nell'introduzione si è accennato al fatto che una parte dei dati elaborati in questo progetto fossero **Open Data**. Con il termine Open Data si intendono tipi di dati di libero accesso, che non presentano alcun tipo di copyright o di forme di controllo che ne limitino la riproduzione e la distribuzione. I principali fornitori di Open Data sono (o dovrebbero essere) le Pubbliche Amministrazioni che, in un'ottica più ampia di *Open Government*, dovrebbero rilasciare grandi quantità di dati al fine di perseguire trasparenza e partecipazione nei confronti dei cittadini.

Dagli Open Data si possono ottenere informazioni importanti, ma se i dataset sono isolati gli uni dagli altri, non assumono valore. Per questo motivo, il modello Open Data trova il suo naturale sbocco nel paradigma **Linked Data**, in cui dataset differenti, pubblicati indipendentemente, vengono resi **interoperabili** tra loro. Per collegare tra loro dati provenienti da fonti diverse, si utilizzano delle linee guida, anch'esse inizialmente tracciate da Tim Berners-Lee:

1. Ogni oggetto deve essere identificato univocamente da una URI.
2. Le URI devono essere definite su protocollo HTTP per poter essere referenziate da persone e user-agents.
3. Quando si ricerca una URI, si devono fornire informazioni utili nei formati standard come RDF, RDFS o SPARQL.
4. Inserire, dove possibile, link ad altre URI esterne relative ai dati esposti per migliorare la ricerca di informazioni contestuali nel Web.

Appendice B

Software Utilizzati

Per il lavoro svolto in questa tesi sono stati utilizzati diversi software open source reperibili in rete. L'utilizzo di tali software ha aiutato nella realizzazione delle principali fasi del lavoro, in particolare quella di elaborazione dei dati grezzi provenienti dalle diverse fonti, quella di conversione semiautomatica dei dati in formato RDF, nella memorizzazione su repository e nella successiva interrogazione di tali dati.

B.1 PENTAHO Kettle

Uno dei primi compiti che si è reso necessario per questo lavoro è stato quello di elaborare i dati provenienti dalle varie fonti per adattarli all'ontologia **SmartCity Ontology**. Poichè il volume dei dati da trattare è risultato elevato e le procedure da applicare relativamente comuni (manipolazione di stringhe, unione, suddivisione e filtraggio di file, conversioni di formato, eccetera), si è preferito ricorrere a software già esistenti in rete e collaudati, per velocizzare il lavoro e per evitare di incorrere in problemi di scalabilità o di errori su software realizzati ad hoc.

Pertanto, per il compito appena descritto è stata effettuata una ricerca ed una analisi approfondita sui principali software **ETL (Extract, Transform, Load)** open source presenti online. I software ETL, come facilmente intuibile dal nome, sono prodotti che **Estraggono** dati da molteplici possibili sorgenti, li elaborano attraverso una concatenazione di **Trasformazioni** ed infine effettuano il **Salvataggio** di tali dati in un formato scelto dall'utente.

I principali candidati sottoposti ad analisi sono stati CloverETL¹, Kettle Pentaho Data Integration², Talend Open Studio³.

A seguito di una approfondita analisi, la scelta è ricaduta su Pentaho Kettle. Le principali caratteristiche prese in esame che hanno fatto propendere la scelta per questo prodotto sono state molteplici. In particolare, per Kettle Pentaho è risultato molto importante:

- L'ampio spettro di possibili sorgenti di input e di formati di output lavorabili (CSV, JSON, Shapefile, KML, Database Relazionali, Repository di tipo Big Data, eccetera).
- La possibilità di esportare le trasformazioni e di eseguirle in modalità batch tramite gli scheduler integrati nel sistema operativo.
- La possibilità di manipolare le trasformazioni native, iniettando direttamente del codice JAVA (o Javascript a seconda dei casi) nelle trasformazioni per ottenere dei risultati altamente personalizzati.
- L'ampio bacino di utenti che animano la community online di Kettle e la ricca documentazione reperibile online.

Kettle prevede due possibili modalità di lavoro: tramite interfaccia grafica (il processo è chiamato Spoon) e tramite terminale attraverso l'eseguibile Kitchen. Tramite interfaccia grafica, è possibile definire le due componenti principali di Kettle, ovvero le **Transformations** e i **Jobs**. Le trasformazioni sono gli elementi fondamentali del software, esse permettono di definire le

¹Clover ETL - <http://www.cloveretl.com/>

²Kettle Pentaho Data Integration - <http://wiki.pentaho.com/display/BAD/Downloads>

³Talend Open Studio - <http://www.talend.com/download/data-integration>

principali operazioni di trasformazione su alcuni flussi di dati. I Jobs invece sono, in estrema sintesi, delle sequenze di trasformazioni, ma tramite di essi è possibile definire in modo dettagliato il flusso delle operazioni. Se all'interno delle trasformazioni non vi è un necessario ordinamento nell'esecuzione degli step, tramite Jobs è possibile definire la sequenzialità nelle operazioni, eventuali parallelismi, loop, la gestione delle eccezioni, eccetera. Tramite interfaccia è ovviamente possibile eseguire le trasformazioni ed i jobs: Kettle mette a disposizione molteplici strumenti di analisi e di logging per controllare accuratamente lo svolgersi delle operazioni richieste e per intervenire su eventuali errori. Salvando le trasformazioni ed i jobs in file dal formato specifico per Kettle, è possibile eseguirli successivamente anche tramite linea di comando. In questo modo, è possibile schedulare i vari processi per farli eseguire automaticamente, senza bisogno di un utente che interagisca direttamente con il programma.

B.2 QGIS (Quantum GIS)

Quantum GIS⁴ è un software GIS (Geographical Information System) open source che permette l'acquisizione, la manipolazione e la visualizzazione di informazioni su dati geolocalizzati. Il software mette a disposizione una interfaccia grafica molto evoluta ed una serie di eseguibili, da richiamare tramite linea di comando, per alcune delle sue principali funzionalità. QGIS, per il lavoro di questa tesi, è stato utilizzato in un compito fondamentale: quello di riportare tutti i dati provenienti dalle diverse fonti ad un unico sistema di riferimento e proiezione geografico. In particolare, la maggior parte dei dati facenti parte del Grafo Stradale proveniente dall'Osservatorio Trasporti della Regione Toscana, è codificato secondo lo standard Gauss-Boaga (per la Toscana EPSG 3003, chiamato anche Monte Mario 1)⁵. Ad oggi, il sistema di coordinate utilizzato prevalentemente (ad esempio da servizi come la

⁴QGIS A Free and Open Source Geographic Information System - <http://www.qgis.org/>

⁵Proiezione di Gauss-Boaga - http://it.wikipedia.org/wiki/Proiezione_di_Gauss-Boaga

navigazione satellitare GPS o dai principali fornitori di mappe online quali Google Maps o OpenStreetMap) è WGS84 (EPSG 4326)⁶. Tramite QGIS è infatti possibile convertire i dati geografici attraverso diverse proiezioni o sistemi di riferimento. Quindi è stato utilizzato QGIS per convertire tutti i dati associati al sistema Gauss-Boaga nel formato WGS84. In realtà, QGIS ha permesso anche un altro risultato: durante la conversione da Gauss-Boaga a WGS84 è stata implicitamente effettuata anche la conversione di formato da Shapefile a KML (Keyhole Markup Language). KML è un linguaggio derivante da XML che permette di gestire dati geospaziali tridimensionali e che viene utilizzato dai principali software di rendering geografici (ad esempio Google Earth, o NASA World Wind). Questa conversione si è resa necessaria dopo aver riscontrato che il software Pentaho Kettle presentava dei bug nell'importazione di file di tipo Shapefile (per alcuni tipi di dati).

B.3 KARMA Data Integration Tool

Karma⁷ è un software sviluppato da un team della University of Southern California, che permette di mappare in maniera semi-automatica dati provenienti da file, da servizi o da database relazionali in formato RDF (Resource Description Framework). È un software molto recente, al punto che durante lo svolgimento di questa tesi, il prodotto si è evoluto, anche grazie ad alcune segnalazioni inviate dalle persone che assieme al sottoscritto, hanno lavorato a questa prima fase del progetto Sii-Mobility.

Il software KARMA viene rilasciato sotto licenza Apache 2.0⁸ e permette di essere utilizzato in due modi differenti: come Web Application da eseguire tramite la piattaforma Apache Tomcat, oppure tramite linea di comando per generare offline le triple RDF.

Tipicamente il flusso di lavoro risulta essere il seguente: tramite interfaccia web si definiscono le mappature tra i dati sorgente e le relative classi e

⁶WGS84 - <http://it.wikipedia.org/wiki/WGS84>

⁷Karma - Data Integration Tool - <http://www.isi.edu/integration/karma/>

⁸Apache License, Version 2.0 - <http://www.apache.org/licenses/LICENSE-2.0.html>

proprietà delle ontologie di riferimento. Una volta terminata la mappatura, è possibile ottenere il modello della trasformazione. Tale modello è esportabile in due differenti formati: il modello R2RML e quello che secondo la terminologia di Karma viene chiamato Service Model. Ai fini di questa tesi è stato utilizzato il modello R2RML che il W3C definisce come “*a language for expressing customized mappings from relational databases to RDF datasets*”, ovvero un linguaggio per la mappatura di dati dal formato relazionale a quello RDF. Una volta generato il modello R2RML, è possibile riutilizzarlo in seguito per generare, in modalità batch, le triple a partire da una base dati relazionale (nel caso di questo lavoro MySQL). In questo modo si rende non necessario l'intervento diretto da parte di un utente, ma si automatizza il processo e si rende disponibile anche al lancio da parte di software come Kettle.

B.4 OpenRDF Sesame

Sesame è un framework open source per la creazione e l'interrogazione di repository RDF. Esso permette il salvataggio di database RDF (chiamati anche *triplesstores*) persistenti localmente in memoria o su disco, oppure in remoto su altri server. Sesame può essere utilizzato in vari modi: tramite interfaccia Web (implementata come applicazione eseguita su Web Server Tomcat), tramite una applicazione di tipo console da lanciare attraverso terminale, oppure attraverso API JAVA. Sesame infatti mette a disposizione molteplici API utilizzabili tramite una qualsiasi applicazione Java che permettono di personalizzare al massimo l'utilizzo del framework. Tramite tutti e tre i casi appena elencati è possibile accedere ad un'altra funzionalità molto importante di Sesame: la possibilità di interrogare il repository tramite linguaggio SPARQL.

B.5 OWLIM

OWLIM⁹ è una famiglia di sistemi per la gestione di database RDF semantici. Viene tipicamente utilizzato come plugin di Sesame per aggiungere importanti caratteristiche riguardanti la gestione della conoscenza. Utilizzando OWLIM come repository semantico infatti, è possibile generare nuova conoscenza sui dati tramite il procedimento di inferenza legato alla logica del primo ordine. In aggiunta a questo, OWLIM è sviluppato per ottenere la massima scalabilità anche in presenza di molti importanti di dati, e di conseguenza permette ottime performance nel caricamento e nella valutazioni di query. La versione utilizzata nel progetto, rilasciata sotto licenza al laboratorio DISIT, mette a disposizione anche strumenti per la realizzazione di query di tipo geospaziale, per la creazione di indici per la ricerca di tipo Full-Text e molte altre funzioni che incrementano la velocità e l'usabilità del software.

⁹Ontotext OWLIM - <http://www.ontotext.com/owlim>

Bibliografia

- [1] G. Ortolani, “Analisi e sviluppo di un sistema di integrazione e modellazione della conoscenza da open data provenienti dalla pubblica amministrazione toscana per smart city e info-mobilità,” Tesi di Laurea Triennale - Anno Accademico 2012-2013.
- [2] F. Tuveri, “Analisi e sviluppo di un sistema di integrazione e modellazione della conoscenza da open data a supporto ed analisi di soluzioni smart city,” Tesi di Laurea Triennale - Anno Accademico 2012-2013.
- [3] B. Lorenz, H. J. Ohlbach, and L. Yang, “Ontology of transportation networks,” 2005.
- [4] “Requisiti tecnici per l’interoperabilità degli enti federati al mobility information integration center - gestione parcheggi e sensori traffico - release 1.1,” 14/09/2010. Regione Toscana Direzione Generale Politiche Territoriali e Ambientali.
- [5] P. Nesi, P. Bellini, N. Rauch, and M. D. Claudio, “Tassonomy and review of big data solutions navigation,” 2013.
- [6] C. A. Knoblock, P. Szekely, J. L. Ambite, S. Gupta, A. Goel, M. Muslea, K. Lerman, M. Taherian, and P. Mallick, “Semi-automatically mapping structured sources into the semantic web,” in *Proceedings of the Extended Semantic Web Conference*, (Crete, Greece), 2012.

BIBLIOGRAFIA

- [7] P. Szekely, C. A. Knoblock, F. Yang, X. Zhu, E. Fink, R. Allen, and G. Goodlander, “Connecting the Smithsonian American Art Museum to the Linked Data Cloud,” in *Proceedings of the 10th Extended Semantic Web Conference*, (Montpellier), May 2013. Awarded Best In-Use Paper at ESWC 2013.
- [8] P. Nesi, P. Bellini, and N. Rauch, “Smart city data via lod/log service,” 2014.
- [9] P. Nesi and P. Bellini, “Modeling performing arts metadata and relationships in content service for institutions,” 2013. Accepted for publication on Multimedia Systems Journal, Springer.
- [10] T. Berners-Lee, J. Hendler, and O. Lassila, “The semantic web,” *Scientific American*, vol. 284, pp. 34–43, May 2001.