Linked Data per i Beni Culturali

Note per il Corso Seminariale in Cultura Digitale
del
Corso di Laurea Magistrale in Informatica Umanistica

Angelica Lo Duca
Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
via G. Moruzzi 1, 56124 Pisa
angelica.loduca@iit.cnr.it

Introduzione ai Linked Open Data

Siamo circondati dai dati. Dati di diversa natura, forma e contenuto. Essenzialmente esistono due tipi di dato: i dati aperti - in inglese open - e i dati proprietari. I dati open sono quelli rilasciati secondo una licenza aperta, che generalmente ne autorizza il riutilizzo e la distribuzione. I dati proprietari, invece, sono propri di una compagnia e normalmente la loro licenza ne vieta l'utilizzo, oppure consente l'utilizzo di una piccola parte, con alcune restrizioni. E' questo il caso dei dati dei social media come Facebook o Twitter. Poiché i dati proprietari non possono essere utilizzati liberamente, in questa sede, si tratteranno solo i dati open.

I dati acquistano valore rispetto a due attori fondamentali. Da un lato i produttori di dati, dall'altro i consumatori. Esempi di produttori sono le pubbliche amministrazioni o gli istituti di statistica, che rilasciano periodicamente dei dataset relativi ad un certo argomento. Ma produttori di dati siamo anche noi quando lasciamo delle recensioni su un prodotto acquistato su Amazon o su un albergo in cui abbiamo soggiornato. Tuttavia la produzione di dati, da sola, non basta, è necessario che ci sia anche qualcuno che li consumi. Esempi di consumatori di dati sono tutte le applicazioni web che mostrano liste di prodotti oppure risultati relativi ad elaborazioni statistiche.

Spesso, però, i dati presenti sulla Rete non sono collegati tra di loro, cosicché troviamo lo stesso dato replicato su sorgenti diverse e per scoprire eventuali relazioni, spesso occorre passare da un motore di ricerca, che, attraverso dei filtri e delle parole chiave riesce - non sempre - a fare questa operazione. Al fine di migliorare l'operazione di collegamento tra i dati, nascono i Linked Data [2], cioè una serie di *best practices* per collegare attraverso il Web dati strutturati. I Linked Data rispondono a tre quesiti fondamentali:

- come fornire un accesso ai dati i modo che possano essere facilmente riutilizzati?
- Come consentire la scoperta di dati rilevanti all'interno di una moltitudine di dataset disponibili?
- Come permettere alle applicazioni di integrare dati da un gran numero di sorgenti dati prima sconosciute?

Al fine di rispondere alle domande precedenti, i Linked Data si basano su quattro principi fondamentali [2]:

1. il nome di ogni cosa che si vuole rappresentare deve essere un URI - esempi di URI sono le URL (Uniform Resource Locator) e le URN (Uniform Resource Name)
2. L'URI deve essere di tipo HTTP in modo che l'utente possa cercarla - ad esempio http://www.esempio.it/Mario_Rossi
3. Quando qualcuno guarda un URI, occorre fornire informazioni utili, utilizzando degli standard (RDF*, SPARQL)
4. Includere link ad altre URI in modo che un utente posso scoprire nuove cose.

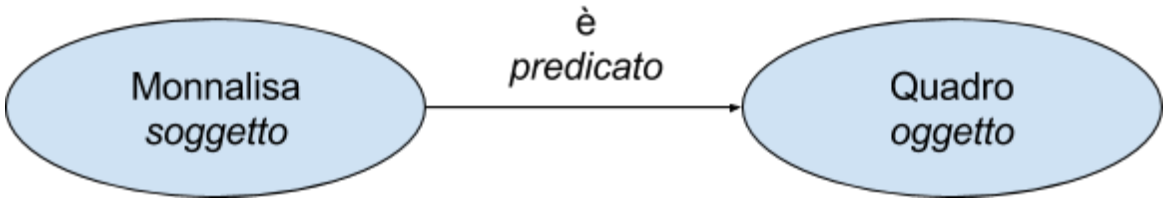
A questi quattro principi se ne aggiungono altri [3]:

1. specificare una licenza appropriata
2. utilizzare vocabolari standard per la rappresentazione dei dati
3. annunciare alla comunità di riferimento i nuovi dati
4. assumersi la responsabilità di mantenimento e aggiornamento dei dati.

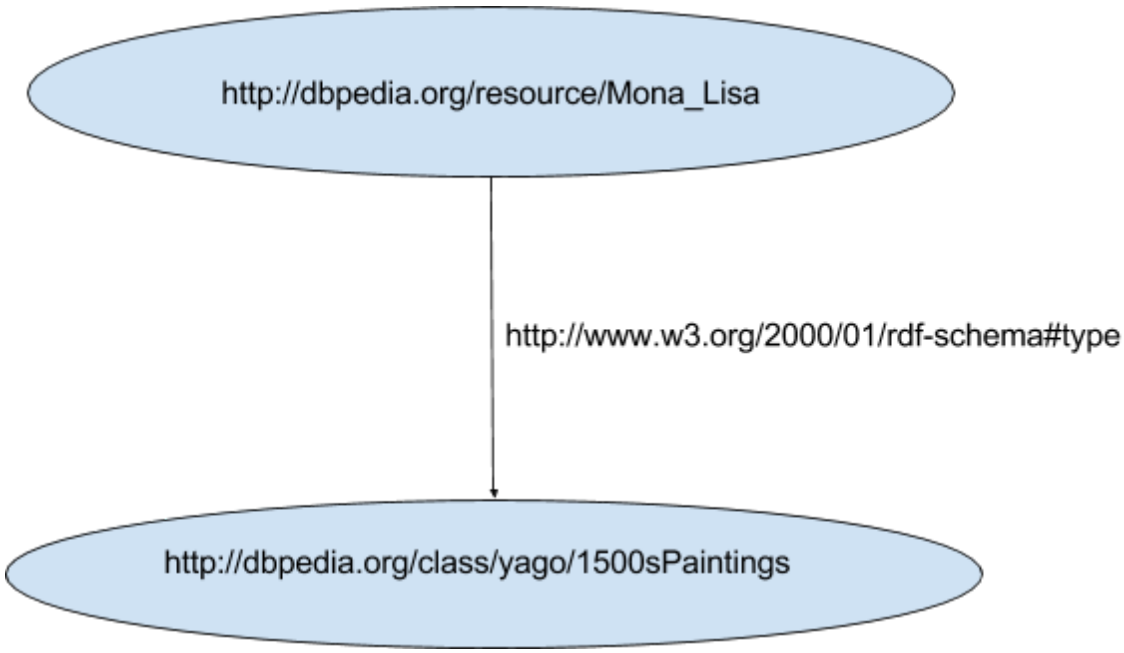
Modellazione dei Dati

La modellazione dei dati consiste nella rappresentazione dei dati secondo un paradigma. Nell'ambito dei Linked Data i dati sono rappresentati in RDF (acronimo di Resource Description

Framework). Il modello RDF si basa su triple (soggetto, predicato, oggetto). Ad esempio l'affermazione "La Monna Lisa è un quadro" può essere così trasformata:



Il soggetto di una tripla rappresenta una risorsa e deve essere sempre un URI, il predicato definisce una proprietà del soggetto e deve essere rappresentato secondo un vocabolario definito, l'oggetto specifica il valore della proprietà del soggetto e può essere un URI oppure un literal, cioè una stringa.



Nell'esempio, La Monna Lisa è rappresentata dall'URI di DBpedia[1], che è uno dei nodi Linked Data più famosi, il predicato *type* è estratto dal vocabolario RDF Schema[2]; l'oggetto, preso sempre da DBpedia, specifica che si tratta di un quadro del 1500. Al fine di compattare la notazione si possono creare dei namespaces, che consistono nella sostituzione dell'URI esteso con un un prefisso:

- dbpedia : <http://dbpedia.org/resource/>
- yago: <http://dbedia.org/class/yago/>
- rdfs : <http://www.w3.org/2000/01/rdf-schema#>

Quindi l'esempio precedente diventa:

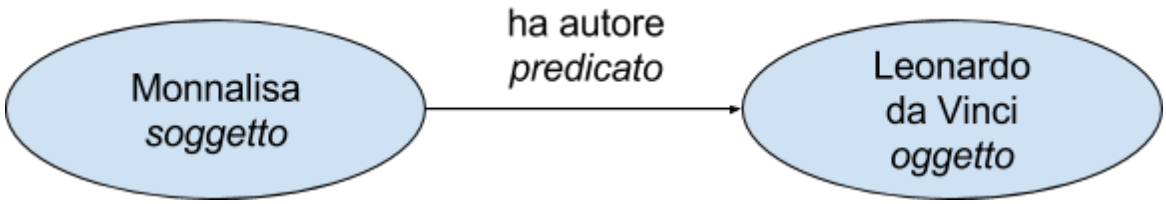
```
@prefix dbr  : < http://dbpedia.org/resource/ > .
@prefix yago : < http://dbedia.org/class/yago/ > .
@prefix rdfs : < http://www.w3.org/2000/01/rdf-schema# > .

dbr:Mona_Lisa rdfs:type yago:1500sPaintings .
```

[1] http://dbpedia.org
[2] http://www.w3.org/TR/rdf-schema/

L'esempio è stato scritto secondo la notazione Turtle[3] (RDF Triple Language). Turtle è un linguaggio per la descrizione di triple. Due osservazioni:

- per dichiarare un namespace, occorre usare la parola chiave `@prefix`
- ogni frase termina con il punto.

La descrizione di ogni risorsa può essere arricchita aggiungendo altre proprietà. Ad esempio si può aggiungere che "*La Monna Lisa ha come autore Leonardo da Vinci*".



In Turtle diventa:

```
@prefix dbr  : <http://dbpedia.org/resource/ > .
@prefix yago : <http://dbedia.org/class/yago/ > .
@prefix rdfs : <http://www.w3.org/2000/01/rdf-schema# > .
@prefix dbo  : < http://dbpedia.org/resource/classes#> .

dbr:Mona_Lisa rdfs:type yago:1500sPaintings .
dbr:Mona_Lisa dbo:author dbr:Leonardo_da_Vinci .
```

In grassetto le modifiche apportate all'esempio precedente. **dbr:Leonardo_da_Vinci** costituisce una risorsa, che va ulteriormente specificata con le sue proprietà.
Per compattare ulteriormente la notazione, Turtle permette di non ripetere lo stesso soggetto di più triple attraverso l'uso del punto e virgola:

```
@prefix dbr  : <http://dbpedia.org/resource/ > .
@prefix yago : <http://dbedia.org/class/yago/ > .
@prefix rdfs : <http://www.w3.org/2000/01/rdf-schema# > .
@prefix dbo  : <http://dbpedia.org/resource/classes# > .

dbr:Mona_Lisa   rdfs:type yago:1500sPaintings ;
                dbo:author dbr:Leonardo_da_Vinci .
```

Si possono aggiungere ancora altre proprietà, come il museo di appartenenza oppure il tema della risorsa:

```
@prefix dbr  : <http://dbpedia.org/resource/ > .
@prefix yago : <http://dbedia.org/class/yago/ > .
@prefix rdfs : <http://www.w3.org/2000/01/rdf-schema# > .
@prefix dbo  : <http://dbpedia.org/resource/classes# > .
@prefix dbp  : <http://dbpedia.org/property/> .
@prefix dbpo : <http://dbpedia.org/resource/Category> .
@prefix dc   : <http://purl.org/dc/terms/> .

dbr:Mona_Lisa   rdfs:type yago:1500sPaintings ;
                dbo:author dbr:Leonardo_da_Vinci;
```

[3] http://www.w3.org/TR/turtle/

```
                dbp:museum dbr:Louvre;
                dc:subject dbc:Italian_Paintings .
```

Come si vede dall'esempio, si possono utilizzare diversi vocabolari, detti in linguaggio più specifico *ontologie*. Inoltre,
Una volta specificate le varie proprietà, se ne può aggiungere una particolare, detta *sameAs*, che specifica l'equivalenza tra risorse. Ad esempio si può dire che la risorsa Monna Lisa su DBpedia è esattamente la stessa risorsa su Wikidata:

```
@prefix dbr  : <http://dbpedia.org/resource/ > .
@prefix yago : <http://dbedia.org/class/yago/ > .
@prefix rdfs : <http://www.w3.org/2000/01/rdf-schema# > .
@prefix dbo  : <http://dbpedia.org/resource/classes# > .
@prefix dbp  : <http://dbpedia.org/ property/> .
@prefix dbpo : <http://dbpedia.org/resource/Category > .
@prfix dc    : <http://purl.org/dc/terms/ > .
@prefix owl  : <http://www.w3.org/2002/07/owl#> .
@prefix wiki-data : <https://www.wikidata.org/wiki> .

dbr:Mona_Lisa    rdfs:type yago:1500sPaintings ;
                 dbo:author dbr:Leonardo_da_Vinci;
                 dbp:museum dbr:Louvre;
                 dc:subject dbc:Italian_Paintings;
                 owl:sameAs wiki-data:Q12418 .
```

Il linguaggio SPARQL

SPARQL[4] è un linguaggio per interrogare un database a triple. Come SQL è usato per interrogare un database relazionale, così SPARQL nasce per interrogare un *triplestore*, cioè un database modellato secondo il modello RDF.

Si supponga di voler interrogare la base di dati fornita da DBpedia[5] al fine di sapere chi è l'autore della Monna Lisa. Come in SQL si usano le parole chiave SELECT, per selezionare quali elementi aggiungere al risultato, e WHERE per specificare la condizione.

```
SELECT ?autore
WHERE {
      <http://dbpedia.org/resource/Mona_Lisa> dbo:author ?autore .
}
```

Il risultato della query precedente è http://dbpedia.org/resource/Leonardo_da_Vinci Il simbolo ? davanti ad una stringa è usato per specificare una variabile. Ogni variabile specificata dalla SELECT deve essere definita nella clausola WHERE. Nel corpo della WHERE si specificano delle triple, in cui si usa una variabile se la parte corrispondente (soggetto, predicato o oggetto) è sconosciuta.

[4] http://www.w3.org/TR/sparql11-query/
[5] http://dbpedia.org/sparql

Per esempio, se nella tripla precedente si vuole conoscere il tipo di relazione tra la Monna Lisa e Leonardo da Vinci, la query SPARQL assume la seguente forma:

```
prefix dbr: <http://dbpedia.org/resource/>

SELECT ?relazione
WHERE {
dbr:Mona_Lisa ?relazione dbr:Leonardo_da_Vinci .
}
```

Da notare che si è utilizzato il namespace `dbr`, dichiarato precedentemente con la parola chiave `prefix`. Rispetto alla sintassi turtle, quella SPARQL non prevede né l'uso della @ per la dichiarizione di un prefisso né il punto finale. Il risultato della precedente query è il seguente:

```
http://dbpedia.org/property/artist
http://dbpedia.org/ontology/author
```

Un'altra possibile query potrebbe essere la seguente: "selezionare tutte le risorse di cui Leonardo da Vinci è l'autore":

```
prefix dbr: <http://dbpedia.org/resource/>

SELECT ?risorsa
WHERE {
?risorsa dbo:author dbr:Leonardo_da_Vinci .
}
```

Il risultato di questa query è una lunga lista di opere di Leonardo da Vinci[6].

Come in SQL, anche in SPARQL è possibile ordinare i risultati delle query, usando la parola chiave ORDER BY. Così, se si vuole ordinare in modo crescente i risultati della precedente query, basta eseguire la seguente query:

prefix dbr: <http://dbpedia.org/resource/>

```
SELECT ?risorsa
WHERE {
?risorsa dbo:author dbr:Leonardo_da_Vinci .
}
ORDER BY ASC(?risorsa)
```

Al risultato della query si possono inoltre applicare dei filtri, usando la parola chiave FILTER. Per esempio, se si vogliono solo le opere il cui titolo inizia con la lettera 'M', si esegue la seguente query:

[6] http://dbpedia.org/sparql?default-graph-uri=http%3A%2F%2Fdbpedia.org&query=prefix+dbr%3A+%3Chttp%3A%2F%2Fdbpedia.org%2Fresource%2F%3E+%0D%0A%0D%0ASELECT+%3Frisorsa%0D%0AWHERE+%7B+%0D%0A%3Frisorsa+dbo%3Aauthor+dbr%3ALeonardo_da_Vinci+.%0D%0A%7D%0D%0A&format=text%2Fhtml&CXML_redir_for_subjs=121&CXML_redir_for_hrefs=&timeout=30000&debug=on

```
prefix dbr: <http://dbpedia.org/resource/>

SELECT ?risorsa
WHERE {
?risorsa dbo:author dbr:Leonardo_da_Vinci ;
 dbp:title ?titolo .
 FILTER regex(?titolo, "^M")
}
ORDER BY ASC(?risorsa)
```

Per cercare il match dei titoli, è stata aggiunta una nuova tripla, che collega la variabile `?titolo` alla risorsa. Come in turtle, anche in SPARQL si usa il punto e virgola per definire più triple che condividono lo stesso soggetto. Sulla variabile `?titolo` è stata poi applicata una espressione regolare (delineata dalla parola chiave regex), che specifica che il titolo inizia con la lettera M. Il risultato della query è il seguente:

```
http://dbpedia.org/resource/Madonna_Litta
http://dbpedia.org/resource/Madonna_of_Laroque
http://dbpedia.org/resource/Madonna_of_the_Yarnwinder
http://dbpedia.org/resource/Mona_Lisa
http://dbpedia.org/resource/The_Virgin_and_Child_with_St._Anne_(Leonard
o)
```

L'ultima risorsa potrebbe trarre in inganno, ma in realtà il titolo di quest'opera è *Madonna and Saint Anne* per cui inizia con la lettera M.

Il linguaggio SPARQL è molto potente. E' possibile eseguire altre operazioni, come ad esempio le unioni tra due query, includere opzionalmente dei risultati ecc. Per maggiori dettagli, si può consultare il libro di DuCharme, Learning SPARQL [4].

Caso d'Uso

In questa sezione, si spiega come creare un semplice nodo Linked Data.

Setup del nodo Linked Data

Il primo passo verso la costruzione di un nodo Linked Data consiste nell'installazione della piattaforma che verrà utilizzata per ospitare i dati. Le piattaforme esistenti si dividono in due categorie: quelle che si basano su un triplestore nativo (come ad esempio Virtuoso[7] e Apache Jena[8]) e quelli che appoggiandosi ad un database relazionale (come ad esempio D2RQ[9]), espongono solo il risultato finale in RDF.
In questa sede si considera D2RQ. Per l'installazione, basta seguire la procedura guidata specificata sul portale, facendo attenzione a rispettare i prerequisiti specificati nel file README. D2R converte i dati presenti in un database relazionale direttamente in RDF attraverso un file di

[7] http://virtuoso.openlinksw.com/linked-data/
[8] https://jena.apache.org
[9] http://d2rq.org

mapping, scritto in turtle. Tale file di mapping contiene tutte le associazioni tra le tabelle e l'ontologia scelta. Per iniziare si può generare un file di mapping a partire dal seguente comando fornito con la distribuzione di D2RQ.

```
./generate-mapping -u root -o mapping.ttl jdbc:mysql:///mydb
```

dove `mydb` è il nome del database contenente il dataset.

Preparazione dei dati

Si supponga di avere a disposizione un dataset relativo ad un carteggio di lettere e si supponga di voler esporre tale dataset come nodo Linked Data. Il dataset è formato dalle seguenti tabelle con i seguenti campi:

Letter

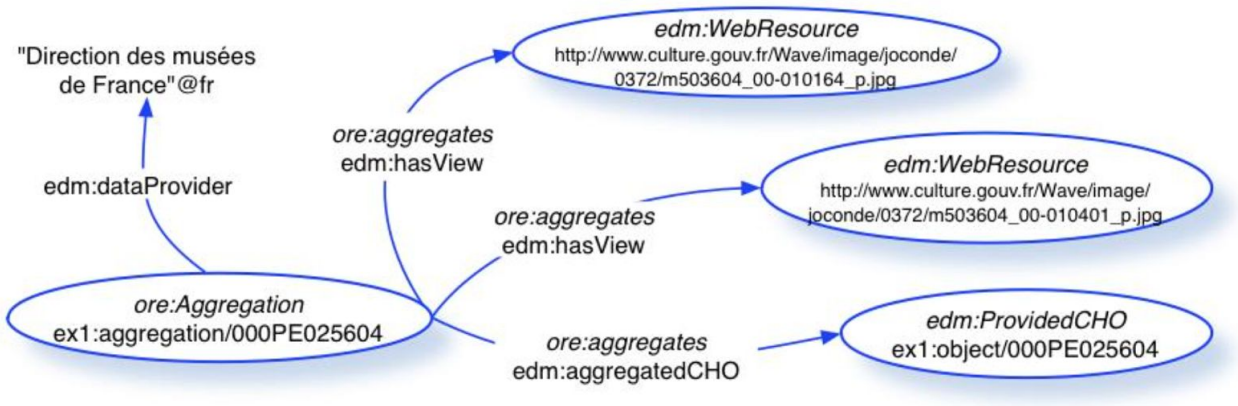
- local_name: numero progressivo associato alla lettera dall'archivio
- author: id relativo all'autore della lettera
- title: titolo della lettera

Person

- person_id: identificativo univoco della persona
- name: nome della Persona

Si supponga inoltre di disporre della scansione di ogni lettera in formato JPEG.
Per convertire queste tabelle in Linked Data, in primo luogo occorre definire il modello di dati che verrà utilizzato per rappresentare il dataset. Al fine di rispettare uno dei principi dei Linked Data - il riutilizzo delle informazioni - bisogna individuare il dominio di appartenenza del dataset (turismo, cultura, salute ecc) e cercare in rete se esistono delle ontologie già definite e consolidate per rappresentare i dati. Nel caso dell'esempio, il dominio è quello culturale. Una delle ontologie più famose in questo caso è l'Europeana Data Model (EDM)[10]. Europeana[11] è un portale che raccoglie oltre 50 milioni di opere d'arte provenienti da tutta Europa, forniti da diversi musei, gallerie ecc.
Nell'ambito dell'EDM, ogni risorsa corrispondente ad un lavoro artistico (quadro, manoscritto ecc) è definita attraverso la classe `ore:Aggregation`, che a sua volta è collegata ad altre due classi: `edm:ProvidedCHO` e `edm:WebResource` (vedi figura sottostante[12]).

[10] http://pro.europeana.eu/files/Europeana_Professional/Share_your_data/Technical_requirements/EDM_Documentation/EDM_Mapping_Guidelines_v2.3_042016.pdf
[11] http://www.europeana.eu/portal/en
[12] Fonte: http://pro.europeana.eu/files/Europeana_Professional/Share_your_data/Technical_requirements/EDM_Documentation/EDM_Primer_130714.pdf



La classe `edm:ProvidedCHO` descrive le proprietà legate alla risorsa culturale (ad esempio titolo, autore, data di creazione ecc), mentre la classe `edm:WebResource` specifica l'identificatore della risorsa Web associata alla risorsa digitale (ad esempio la scansione di una lettera o l'immagine relativa ad un quadro).
Nel caso del dataset di lettere, in D2R si definisce in primo luogo la classe Aggregation (in questa sede per semplicità vengono omessi alcuni campi obbligatori della classe `ore:Aggregation`)

```
@prefix map: <#> .
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
@prefix edm: <http://www.europeana.eu/schemas/edm/> .
@prefix ore: <http://www.openarchives.org/ore/terms/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

# Aggregation
map:Aggregation a  d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "aggregation-@@Letter.local_name|urlify@@";
      d2rq:class ore:Aggregation;
      .
# URI of the web resource
map:aggregation_has_view a  d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:Aggregation;
      d2rq:property edm:hasView;
      d2rq:uriPattern
"http://<something_over_the_web>/@@Letter.local_name|urlify@@.JPEG";
      .
map:aggregation_aggregated_cho a  d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:Aggregation;
      d2rq:property edm:aggregatedCHO;
      d2rq:uriPattern "letter-@@Letter.local_name|urlify@@";
      .
```

D2RQ usa delle triple per specificare il mapping da database relazionale a RDF. Per definire una classe in D2RQ va usata la parola chiave `d2rq:ClassMap`, mentre per specificarne le proprietà

va usata la parola chiave `d2rq:PropertyBridge` associata alla classe attraverso la relazione `d2rq:belongsToClassMap`.
Per specificare che una risorsa appartiene ad una data classe (ad esempio che la lettera è una `edm:Aggregation`), si usa la proprietà `D2RQ d2rq:class`, mentre per specificare il tipo di proprietà di una risorsa, si usa la proprietà D2RQ `d2rq:uriPattern` (nel caso di un URI) o `d2rq:column`, entrambe aventi come oggetto la colonna della tabella desiderata.
Una volta specificata la classe aggregation, si possono specificare le altre classi (Letter e Agent):

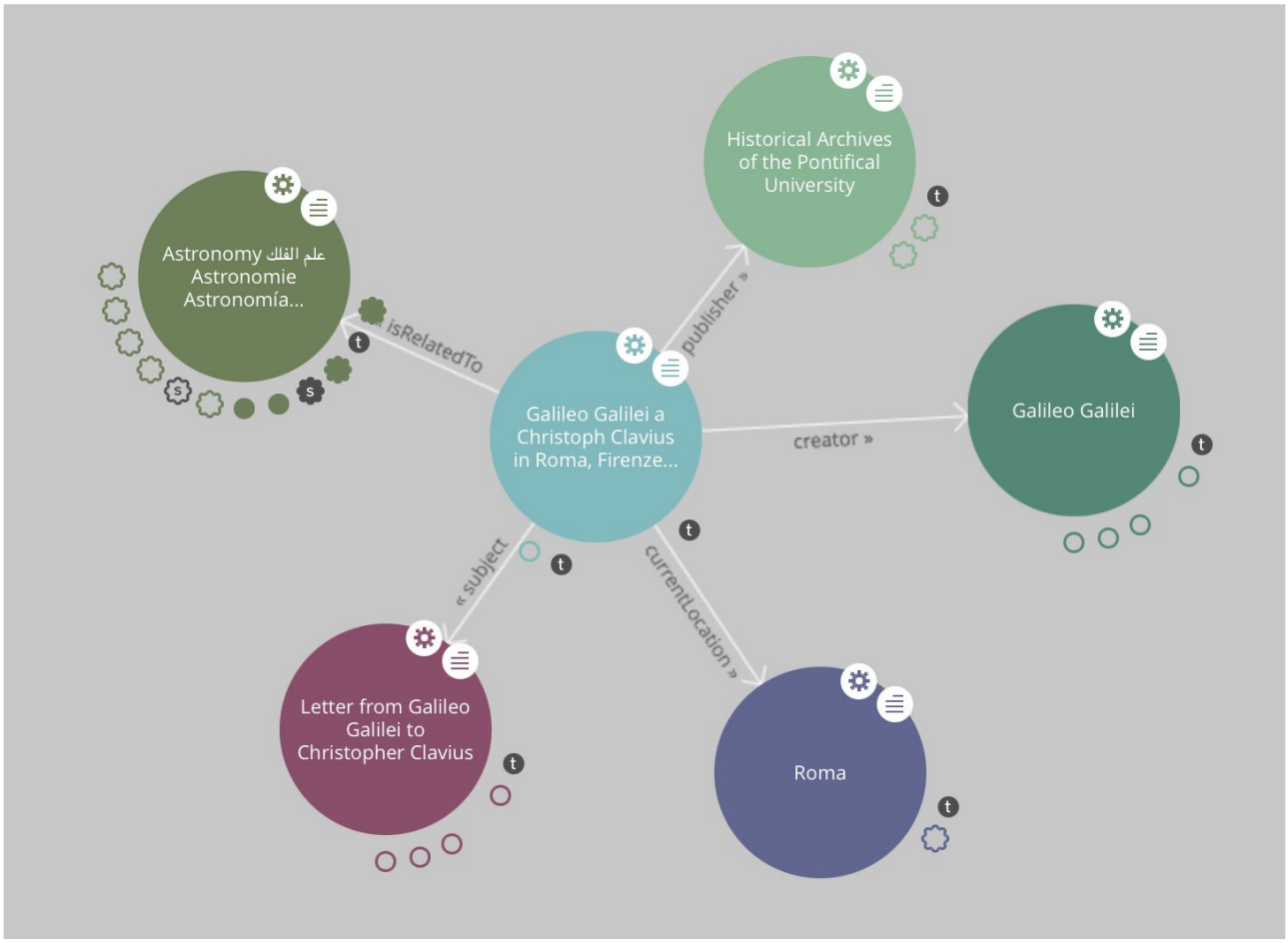
```
# Letter
map:Letter a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "letter-@@Letter.local_name|urlify@@";
      d2rq:class edm:ProvidedCHO;
      .

map:letter_creator a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:Letter;
      d2rq:property dc:creator;
      d2rq:condition "Letter.author = Person.person_id";
      d2rq:uriPattern "@@Person.name_surname|urlify@@";
      .

map:letter_title a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:Letter;
      d2rq:property dc:title;
      d2rq:column "Letter.title";
      .

# Agent
map:Agent a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "@@Person.name|urlify@@";
      d2rq:class edm:Agent;
      .

map:agent_name a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:Agent;
      d2rq:property foaf:name;
      d2rq:column "Person.name";
      .
```

Navigazione dei dati con LodLive

Per la navigazione dei dati, si possono fare delle query SPARQL, oppure si possono utilizzare altri strumenti, come LodLive[13], che è un'applicazione web per la navigazione dei dati. La seguente figura mostra la rappresentazione del dataset attraverso LodLive. Da notare che sono state

[13] http://lodlive.it

aggiunte altre proprietà, come l'ente proprietario del dataset (identicato attraverso la proprietà edm:publisher), il subject (`dcterms:subject`), che corrisponde all'argomento delle lettere, e il concetto collegato (`edm:isRelatedTo`).



Riferimenti bibliografici

[1] Tom Heath and Christian Bizer (2011) Linked Data: Evolving the Web into a Global Data Space (1st edition). Synthesis Lectures on the Semantic Web: Theory and Technology, 1:1, 1-136. Morgan & Claypool.

[2] Tim Berners-Lee (2006-07-27). "Linked Data". *Design Issues*. W3C. Ultimo accesso 19 Settembre 2016

[3] W3C Working Group (2014) . Best practices to publish Linked Data. Technical Report. http://www.w3.org/TR/ld-bp/

[4] Bob DuCharme (2011). Learning SPARQL. O'Reilly Media, Inc..