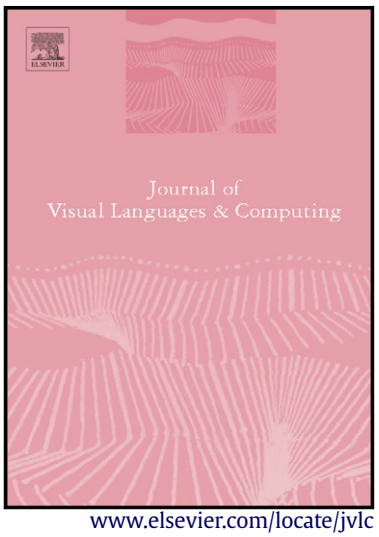

www.elsevier.com/locate/jvlc

Linked open graph: Browsing multiple SPARQL entry points to build your own LOD views

Pierfrancesco Bellini, Paolo Nesi, Alessandro Venturi

PII: S1045-926X(14)00096-2
DOI: http://dx.doi.org/10.1016/j.jvlc.2014.10.003
Reference: YJVLC643

To appear in: *Journal of Visual Languages and Computing*

Received date: 21 September 2014
Accepted date: 1 October 2014

Cite this article as: Pierfrancesco Bellini, Paolo Nesi, Alessandro Venturi, Linked open graph: Browsing multiple SPARQL entry points to build your own LOD views, *Journal of Visual Languages and Computing,* http://dx.doi.org/10.1016/j.jvlc.2014.10.003

Linked Open Graph: browsing multiple SPARQL entry points to build your own LOD views

Pierfrancesco Bellini, Paolo Nesi, Alessandro Venturi
Distributed Systems and Internet Technology Lab, DISIT Lab, Department of Information Engineering
University of Florence, Florence, Italy, tel: +39-055-4796523, fax: +39-055-4797363
http://www.disit.dinfo.unifi.it, paolo.nesi@unifi.it, pierfrancesco.bellini@unifi.it, http://log.disit.org

Abstract -- **A number of accessible RDF stores are populating the linked open data world. The navigation on data reticular relationships is becoming every day more relevant. Several knowledge base present relevant links to common vocabularies while many others are going to be discovered increasing the reasoning capabilities of our knowledge base applications. In this paper, the Linked Open Graph, LOG, is presented. It is a web tool for collaborative browsing and navigation on multiple SPARQL entry points. The paper presented an overview of major problems to be addressed, a comparison with the state of the arts tools, and some details about the LOG graph computation to cope with high complexity of large Linked Open Dada graphs. The LOG.disit.org tool is also presented by means of a set of examples involving multiple RDF stores and putting in evidence the new provided features and advantages using dbPedia, Getty, Europeana, Geonames, etc. The LOG tool is free to be used, and it has been adopted, developed and/or improved in multiple projects: such as ECLAP for social media cultural heritage, Sii-Mobility for smart city, and ICARO for cloud ontology analysis, OSIM for competence / knowledge mining and analysis.**

Keywords LOD, LOD browsing, knowledge base browsing, SPARQL entry points.

I. INTRODUCTION

The large publication of OD (open data) has opened the path for the information sharing. Most of the OD are published by governmental organizations, in file formats such as: html, xml, csv, shp, etc., and typically provide information that may present links to web resources. These links are typically coded as un-typed hyperlinks, URLs (Uniform Resource Locators). In 2006, Tim Berners-Lee published the LD (Linked Data) principles [1], as a model to stimulate the process of making accessible and sharing data as digital resources on the web and from them establishing links with semantically connected sources via URI (Uniform Resource Identifiers) [2]. On this wave, the data publication moved towards the diffusion of LD, opening the path for the construction of LD repositories and thus for creating a globally connected and distributed data space with integrated semantics. LD are based on documents formalized in RDF (Resource Description Framework) [3]. LD are mainly designed to be accessed and reused by machines. An RDF link leads to a triple putting in relationship two entities. For example, *Carl knows Paolo*, this consists of a subject, a predicate and an object or data value, which in turn are represented with URI. Thus, LD as triples can be accessible via specific LD Browsers, which allows following URI from one data set to the model definition and/or to another dataset. Predicates, as "*knows*", may be specified by using well-known vocabulary, such as the FOAF (Friend Of A Friend, [4]) that defines aspects and characteristics of people and their relations, and many others as mentioned in the sequel. A vocabulary defines the common characteristics of things belonging to classes and their relations. A vocabulary, also called ontology, is defined by using the RDFS (RDF Schema, RDF Vocabulary Description Language) or the OWL extension (Ontology Web Language). RDF triples can be stored in RDF stores (databases) and made accessible via SPARQL [17] entry point to pose semantic queries (SPARQL Protocol and RDF Query Language, recursive definition) on the RDF store. A network of SPARQL services and/or as LD/URI allows the creation of a network of LD, thus contributing to the construction of a global data model, which is the Linked Open Data, LOD [2].

In general, SPARQL queries are quite complex to be composed since their formalization strongly depend on the ontological structure of the RDF store model and the relationships among entities. This fact constrains the users to study the ontology in terms of entities and their relationships, also taking into account the external definition in terms of ontology segment, vocabulary, etc.

As an alternative, third parties LD search facilities based on keywords are also provided such as the semantic web crawler, such as Sindice.com [5]. Others solutions provide support to search on the semantic web via URI/LD. Other tools allow federating queries among multiple SPARQL entry points (RDF stores) have been proposed such as via Semantic Web Client Library [6]. This approach is typically applied for searching complementary aspects and composing the results in a unique semantic model.

Therefore, the complexity of accessing and using RDF stores and specifically LOD accessible via SPARQL entry point is limiting their usage. The understanding of LOD structure by using LD browsers is not an easy task and is also limited, since in most cases those browsers represent reticular relationships of LD with simplified tables and pages.

In the literature, to cope with the above mentioned problems, a large number of tools to edit and browse ontologies and knowledge bases have been proposed [7]. Most of them allow the editing of RDF stores and represent the entities by using hierarchical structures. A number of tools have been built on Protégé ontology editor [8]. Among the available tools, only a few of them present a visual representation of the RDF store directly accessing to the SPARQL entry point. iSPARQL [9] is powerful tools that allow to access to an RDF repository via a SPAQRL query that can be visually represented and extended. On the other hand, the representation of results is still in tabular form and the navigation among relationships of the identified entities is very complex for who do not know the ontology structure. A number of tools for visual definition of SPARQL queries have been proposed, as Konduit [10], NITELIGHT [11].

Gruff application allows the visual composition of semantic queries grounded on Allegro Graph. Gruff generates the SPARQL query for accessing the entry point. The usage of Gruff should accelerate the learning of SPARQL language, while the complexity of usage is quite high. Gruff is a local application and includes capabilities for RDF storage browsing and analysis. A different approach has been taken by gFacet [12], which proposed a tool for posing interactive queries on a SPARQL entry point and obtaining interactive faceted results that can be used to refine the queries. Almost all the above mentioned tools are applications that need to be downloaded and installed. On the other hand, LodLive service is a web based RDF browser based on data graph representation (http://lodlive.it) [13]. It allows to access at LD and to single SPARQL endpoints. LodLive provides a user friendly user interface for browsing and navigation on the RDF entities starting from a specific URI. Once chosen the data sets and the URI, the representation of the accessed entity is based on circle surrounded by a number of small circles that can be accessed to expand the relationships.

On the other hand, none of the above mentioned LOD browsing data tools allows to fully exploit the nature of LD/LOD by expanding their rendering and navigation on multiple SPARQL entry points, and only LodLive is accessible via web, and present relevant limitations.

In this paper, Linked Open Graph, LOG, is presented. LOG.DISIT.org is a web based application for collaborative browsing and navigation into multiple SPARQL entry points (RDF stores). The LOG tool is web accessible and it is also in use to create the Social Graph in ECLAP social network as an embedded tool: http://www.eclap.eu [14].

The paper is structured as follows. In Section II, the main aspects of browsing into RDF stores are presented. Section III presents a comparative analysis of SPARQL visual browsers. It includes aspects to access and query, relationships among entities, general manipulation, URI details, and non-functional requirements. The comparison is used to put in evidence the main innovations of the proposed Linked Open Graph, LOG as: (i) management of multiple SPARQL entry points, (ii) saving and sharing of RDF graphs via web, (iii) learning and inspecting RDF graphs. Section IV presents some details about the computation of the LOG graphs and some larger and more complex example. Conclusions are drawn in Section V.

II.RDF STORE AND EXTERNAL LINKS

The example reported in the introduction "*Carl knows Paolo"* consists of a subject, a predicate and an object or data value. These elements, in turn are represented by using URI. The "*knows*" property may be defined to have as domain and range from class *foaf:Person* (from FOAF, [4]). Using this information, it can be inferred that both *Carl* and *Paolo* belong to the class *foaf:Person*. Moreover, the vocabulary states that class *foaf:Person* is a sub class of the more general class *foaf:Agent*, thus both *Carl* and *Paolo* belong to class *foaf:Agent*. The OWL version 2 language proposed by W3C allows defining disjunctive classes, union and intersection of classes, functional properties, symmetric, transitive properties, minimum and maximum cardinality of the associated elements of a property and other features. SPARQL language has been designed to query information on reticular structured information based on triples, and uses advanced algorithms to match portions of the RDF graph with a specified template. For example, the following query lists all the names of people that *Carl* knows indirectly through one or more other persons:

```
SELECT ?n WHERE {
?p1 foaf:mbox <mailto:carl@unifi.it>.
?p1 rdf:knows+ ?p2.
?p2 foaf:name ?n.
}
```

Different data sets may be defined by exploiting vocabularies (ontology segments) for defining properties and classes such as:

- *foaf:knows*, *foaf:Person [4];*
- *OTN*: [18] an ontology of traffic networks that is more or less a direct encoding of GDF (Geographic Data Files) in OWL;
- *dcterms*: [19] set of properties and classes maintained by the Dublin Core Metadata Initiative as *dc:title*;
- *vCard*: for a description of people and organizations [20];
- *wgs84_pos*: vocabulary representing latitude and longitude, with the WGS84 Datum, of geo-objects [21].

Moreover, different RDF stores may be connected each other since they share common vocabulary or since one RDF store may refers with its links/URLs to other stores. Those links could be established after a process of data enrichment. For example, to connect the names of a well-known painter into a museum representation with the painter's biography which is present on dbPedia [22]. On these bases, a number of SPARQL entry points to access at RDF stores are accessible such as: dbPedia [22], Europeana, LinkedGeoData, British Museum, Cultural Italia, Open Link LOD Cache, Linked Movie Data base, Getty vocabulary. A list of SPARQL end points can be found on http://www.w3.org/wiki/SparqlEndpoints. In addition, it is also possible to join two entities defined with different URI with a property *owl:sameAs*.

III. Analysis of LOD graph visual browsers

As described in the previous sections, the visual browsing of SPARQL entry points can be very useful for analysing the RDF store reticular structure, that is at the basis of the ontology and the related instances of predicates contained, the knowledge base. The users may use the LOD graphical viewers and browsers to (i) create RDF representations and models, (ii) save them and share with other colleagues as a basis of discussion, (iii) learn about how SPARQL queries are created. On top of these SPARQL graph viewers, reasoners on different aspects can be provided, to make analysis about geographical and geometrical relationships, temporal relationships, etc. Moreover, different SPARQL versions provide limited capabilities in executing queries, such as problems counting elements, etc. Therefore, despite the first impression, the representation of an RDF reticular structure and thus its access are not a simple neither superficial task. The visual browsing of SPARQL entry points is not a simple task, especially if this work is performed by a Web Application. Thus, as described in the next pages, the LOG.disit.org service is not a simple browsing of related resources. In particular, specific algorithms are needed to cope with complexity of obtaining and processing complex reticular structures with web based applications, removing duplications, managing multiple entry points, generating complex SPARQL queries, etc.

In the following section, a number of demanded features and related problems are discussed with the aim of presenting LOG.DISIT features and comparing them with representative state of the art solutions: LodLive and Gruff. The identified features have been grouped in a few topics and discussed in different subsections: access and query, relationships versus entities, general manipulation, URI details, and non-functional features.

III.A Access and Query LD and Stores

Access and rendering of LD. This means that the visual tool should be capable to represent a LD which is publically accessible as a URI, providing a set of triples. In Figure 1, the rendering of a URI[1] is depicted via LodLive [13]. The single URI is represented with a bubble, and the other small circles are links that can be clicked to expand the visualization to other LD/bubbles. Filled small circles are outbound links to LD, while unfilled small circles are inbound links coming from other LDs towards the former *URI(a)*.

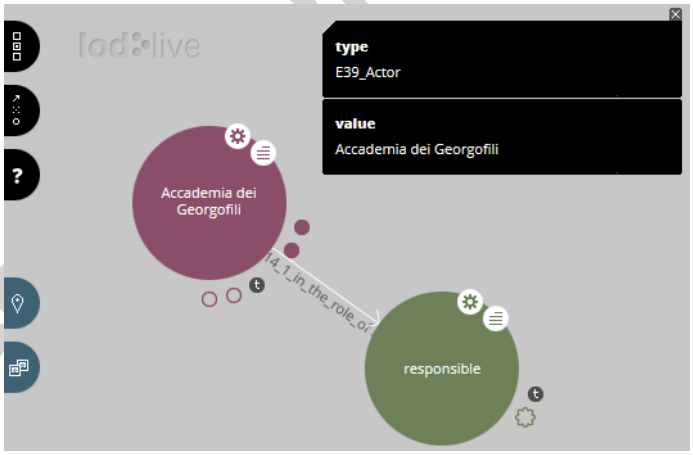

Figure 1: URI rendering with LodLive

These LD as RDF stores are accessible for the application or discovered by a semantic query to well-known SPARQL entry points. The small circles represent properties of the bubble, and some of them (presented in black) have coded letters as: "*t*" as types, "*s*" as *owl:sameAs*, "*b*" as blank nodes. The proposed rendering implies that the relationships of each single bubble/LD are not automatically explored. While their opening can be performed singularly and all together with a mouse click on the big bubble settings small icon.

Access and rendering URI from a SPARQL entry point. A visual tool for browsing SPARQL entry points extract the results by using a couple *{ URL(i), Q },* where *Q* is the semantic query or an URI. In this case, the tool needs to know both the SPARQL entry point of a given store *a* (an URL, that we can identify as *URL(a)*), and at least a URI to be searched in the store (called here

[1] http://dati.culturaitalia.it/resource/actor/accademia-dei-georgofili

as *URI(a)*) to get back the related description in terms of triples. In this case, the rendering of the triples can be similar to that of Figure 1 (representing the URI and the possible identified relationships recovered).

Managing Entry Points with different URLs in URI. In most cases, the *URI(a)* may have the same domain of their corresponding SPARQL entry point *URL(a),* but it is not mandatory. And thus, the tools have to be capable to accept to start browsing from the couple URI, URL having different domains.

Multiple SPARQL entry points. The access and browse to a RDF store via the SPARQL entry point is a way to understand the knowledge base and the relationships among the included entities. In some cases, the entities/URIs (*URI(a)*, *URI(b)*) of different RDF stores (accessible via different SPARQL entry points: *URL(a)* and *URL(b)*) may be connected each other. Typically, the connections can be via URI representing classes of common ontologies and definitions. The visualization of graphs associated with *URL(a),URI(a)* and *URL(b),URI(b)* on the same screen may allow to put in evidence the relationships among these two graphs. They may be the basis for (i) studying how to integrate different ontologies, for federating different RDF storages, (ii) understanding differences and relationships among different models, and/or (iii) for creating additional connections. For example, by creating an *owl:sameAs* relationship among two entities that represent the same concept in two models. In some cases, similar pattern have not been intentionally defined by using the same vocabulary since they are different for some aspect, while in other context they could be the same, otherwise deductions in the knowledge base would not take into account all needed facts.

Making keyword based query. In order to identify a starting URI for RDF graph rendering it could be possible to pose a keyword-based query on the RDF store. This feature is not always available on the RDF store (SPARQL entry points), and may be implemented in several different manners. Some implementation provide additional full text keyword based indexes on Lucene, other simpler solutions provide only substring search facility. The keyword based query is typically performed on all or specific ontology classes. Some of the tools allow selecting the specific class on which the keyword based query is performed. Both LodLive and LOG.DISIT provide this feature.

Inspecting entry point for searching classes. Once an entry point is identified, it is possible to pose queries to inspect it to search for major classes. Thus, a textual search can be performed on the instances of one or more of those classes, in order to get back a list of entities/URI from which the graph visual browsing can start. This feature is quite difficult to use since the selection of the class(es) on which the search is performed may imply a certain knowledge about the ontology modelled in the RDF store of entry point.

III.B Relationships among entities

Showing relationships, turning them on/off, singularly and/or for category. Once the first URI (the bubble in Figure 1) and related URIs are shown several relationships may be present in the graph, maybe hundreds or thousands, see Figure 2 from LOG.disit on dbPedia for URI related to Florence, Italy (it has been searched by a keyword based query on dbPedia SPARQL entry point, and thus it works when dbPedia entry is alive). Some of the relationships may be recursive (classes defined in term of their self); other are quite frequent and multiple, such as: *owl:sameAs*, *subject*, *type*. These should be marked or treated in different manner (as in the case of LodLive mentioned above). In any case, the users should be enabled to turn on/off some of the relationship categories to make the graph more readable and focussed on the entities and relationships under analysis. A LOD graph for an URI can present hundreds of different relationships kinds, and may be millions of triples to instances. Therefore, the usage of one line for each relationship kind is preferable to have a link for triple. The possibility of disabling relationship categories would shorten and simplify the analysis, as in LOG.DISIT.

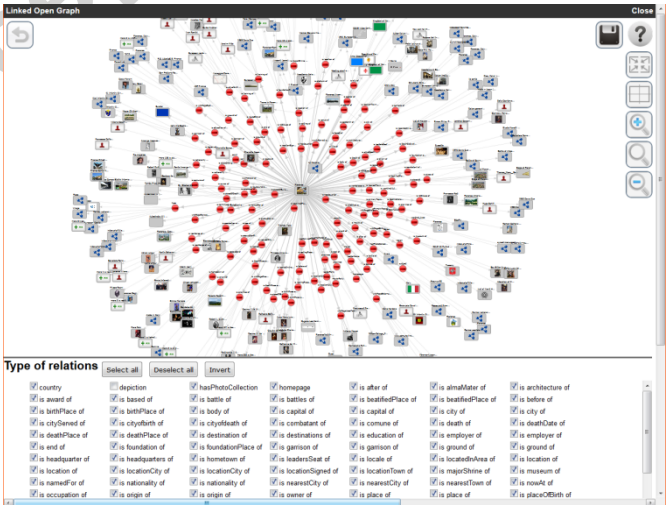

Figure 2: Florence URI on dbPedia, via LOG.DISIT, providing 364 elements: 237 entities and 127 multiple relatioships (red circles).

Moreover, each category of relationship may bring to thousands or millions of entities (see Figure 3 for Gruff). For example, a library as Europeana has millions of elements, the civic number of a national street in Sii-Mobility may be thousands, see for example [http://SiiMobility.disit.org, http://servicemap.disit.org]. This complexity has to be managed somehow, giving the possibility of accessing to a part of them for understanding the model, and to some specific relationships among the entities involved: for example by posing a specific query or faceting directly from each single entity [12]. In some cases, the instances can be easily hidden from the graph disabling specific relationships of instance of.

Representing relationships. In the rendering of the RDF graph, a large number of entities (URI) and the relationships among them may be present. In most cases, the URI may have 1:N relationships that should be represented in different manner (some of them are very frequent such as *owl:sameAs*, *type, or those that bring to a blank node*). The high number of graphical elements can be reduced allowing closing/opening, expanding/compressing relations, filtering some relationships from the visualization (i.e., limiting the rendering to selected relationships) and may be also graphically representing entities and relationships by using coded styles.

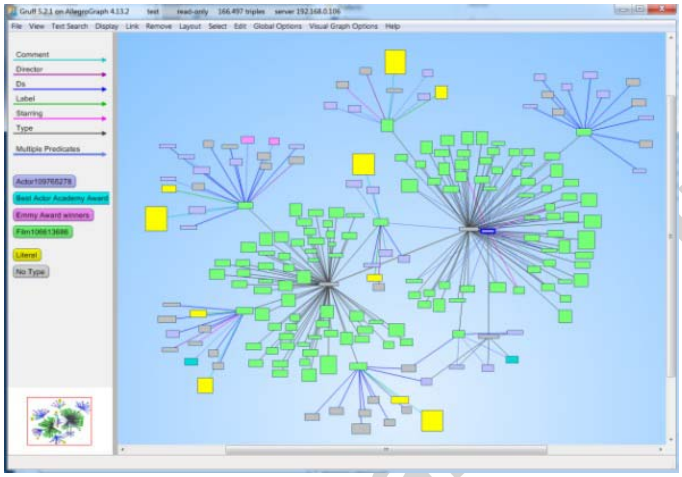

Figure 3: URI on a dbPedia segment, via Gruff.

On this regard, LodLive and Gruff assign a different colour to each URI according to their *type* (see Figure 3). When multiple types are present the colour can be determined by the first one, and thus the assignment may be misleading. In LodLive the color code is not constant so that at each graph reload the same graph may present totally different colors. A different approach could be to assign different icons according to their *type*, as in LOG.disit, and adopted in ECLAP social graph [14].

Discovering inbound/outbound relationships, URI and queries. At each URI a number of semantic queries can be associated with, for example, to recover the relationships:

(A) towards other entities (outbound, *as subject in Gruff*), it can be used:

```
SELECT ?object ?property WHERE {  <http://dati.culturaitalia.it/resource/actor/accademia-dei-georgofili> ?property ?object.
FILTER(isURI(?object))
}
```

(B) coming from other entities towards the former URI (inbound, also called as *object relationships* in Gruff), it can be used:

```
SELECT ?subject ?property WHERE { ?subject ?property <http://dati.culturaitalia.it/resource/role/isProducedBy>
 FILTER(isURI(?subject))
}
```

In some tool, the contextualized text of the query declined for a specific entity is accessible. It can be very useful for training the users in using the SPARQL and for shortening the data exploitation in external applications accessing to the SPARQL entry point API.

The inbound relationships can come from other SPARQL entry points, different from the one under inspection. This analysis implies to have a list of SPARQL entry points: as performed by LodLive and LOG.DISIT. In both cases, the list of accessible entry points for the tools is available for selection. Therefore, the set of SPARQL entry points allows for each URI to make this analysis, see Figure 4 for LOG.DISIT. The analysis allows counting the number of relationships in the different case, and for each of them to see sample the related query performed to get them. The query can be used to get all of them. In the case of Figure 4, 6 inbound and more than 5.6 million of outbound links have been found. In addition, also The British Museum is using that entity in about 40 million of triples, and dbPedia for more than 4.7 million etc. The discovered links can be opened to expand the browsing of the RDF graph to internal and/or external URIs, also belonging to other and multiple SPARQL entry points such as in LOG.DISIT.org, only, using multiple SPARQL rendering.

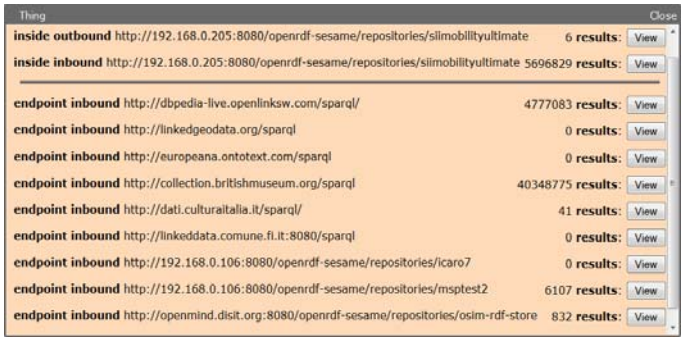

Figure 4: Results of the relationships analysis for http://www.w3.org/2002/07/owl#Thing URI LOG.DISIT.org

Discovering /searching single element from 1:N relation, or sample. Once an entity is identified, it may have thousands or millions of instances connected to it via given relationship. LOD.disit allows the users to make a local search on the connected entities and browsing among the results to look for a specific entity that may be useful to discover a connection/link. This feature is not present in any of the other tools.

Discover paths between URI. As a support to the analysis of the RDF graph, the identification of possible paths between two identified URI can be very useful. This analysis is a complex job to be performed in exhaustive manner for non-trivial cases, see for example the implementation of Gruff. Once identified the possible paths, the user would have to decide to see one or more of them according to some criteria.

Creating triples/relationships. An interesting feature that is moving towards the structural change of the RDF under analysis would be the insertion of new triples / new relationships, as in Gruff. This is possible in several RDF store editor and typically not available in browsers since the new triple should be stored somehow and would not be fine to store in a third party RDF store, even having the authorisation. Nevertheless, the creation of additional triples could be interesting to trial model integration among multiple SPARQL entry points.

A number of other features are also associated with the RDF relationships such as: the possibility of expanding and closing all the relationships, the possibility of counting the number of relationships, and the special management of some of them *owl:sameAs*, links to blank nodes.

III.C General Manipulation

Undo of the actions performed, "back". The users may manipulate the LOD graph by means of several different actions such as opening/expanding URI and/or their relationships, turning on/off some relationship, etc. In the RDF visual graph manipulation, the possibly of undoing the actions performed with a back buttons may be very useful, together with the possibility of saving the reached status.

Save and load LOD graphs. The main aim of graph tools for visual browsing RDF stores is the construction of LOD graph rendering a situation for study and analysis entities and their relationships. The study of knowledge base as well as of ontology is frequently a long process in which several different navigations and openings are performed to explore relationships among the several entities/URIs. Therefore, a very valuable feature is the possibility of saving the status of the graph with all its linked URIs, and the relationships exploded (taking into account their on/off status). This graphical context should be the starting point for further analysis and not a simple image snapshot. Once saved the RDF graph analysis, it could be useful to be reloaded for further elaboration, and/or for sharing it with other colleagues in read or read/write modalities, thus enabling the collaborative work on the same RDF graph analysis. Only Gruff and LOG.DISIT allow saving and loading RDF graphs.

Share and collaborate on LOD graphs. The RDF graph sharing is based on saving the RDF graph on some cloud to provide the possibility to share it to other colleagues to make changes or simple access at the graph via web. Among the tool analysed, only LOG.DISIT provide this collaborative feature on LOD RDF graphs. LOG.DISIT allows sharing the RDF graph as web data on the cloud, in read and read/write modalities.

Export of RDF graph triples. The export of the RDF entities involved in the visualized/pruned RDF graph can be a very useful feature for study the model in other tools.

A number of other features are also associated with the general manipulation of the visual graph such as: Re-layouting the graph the screen rearranging automatically the graphical elements, focusing on an URI (identifying an URI and restarting the navigation

from that point), zooming and panning the graph, centering the graph (moving in the center of the graph the original URI), closing the single relationship, closing the single entity to allow creating specific examples of instances.

III.D URI Details

URI attributes. A number of attributes/values (literal) may be associated with the URI. These data should be accessible without involving graph representation. To this end, a simple table with a list of values can be provided as in LodLive and LOG.DISIT. **Among** the possible values, the GPS coordinates could be used for positioning the URI on geo-MAP (**Map allocation of URI**).

URL to resources. An URI in the LOD graph is the representation of an RDF entity in the store. On the other hand, the original data can be opened in the browser. Moreover, a URI may have among its attributes some URL to external digital resources. These URL should be accessible for opening the digital resources into the browser or for download. They can be files, such as: images, video, documents, web pages, etc. In these cases, it can be useful to have the possibility to directly **Open play resources**.

TABLE 1: SUMMARY OF COMPARATIVE ANALYSIS

	LOG	LodLive	Gruff
Access and Query			
Access and rendering of LD	Y	Y	N
Access and rendering URI from SPARQL entry point	Y	Y	Y
Managing Entry Points with different URL in URI.	Y	N	Y
Multiple SPARQL entry points	Y(10)	N	N
Making keyword based query	Y	Y	Y
Inspecting entry point for searching classes	Y	Y	Y
Relationships vs entities			
Showing relationships, turning on/off, singularly or globally	Y(3)	Y(2)	Y(2)
Representing relationships (managing complexity)	Y	Y(4)	Y(4)
Discovering inbound/outbound relationships, URI and queries	Y	Y	Y(7)
Discovering /searching single element from 1:N relation , or samples	Y	N	N
Discover paths between URI	N	N	Y
Creating triples/relationships	N	N	Y
Expand all relationships	Y	Y	N
Close all relationships, close single relationship	Y	N	N
Closing / hiding the single entity	Y	N	N
Counting number of elements	Y	Y	Y
"sameAs" management	Y	Y	Y
Blank nodes rendering	Y	Y	Y
General Manipulation			
Undo actions performed, "back"	Y	N	Y
Save and Load LOD graphs	Y	N	(Y)
Share and collaborative LOD graphs	Y	N	N
Export of RDF graph triples	N	N	N
Re-layouting the graph	Y(6)	N	Y
Focusing on an URI	Y	Y	N
Zooming the graph	Y	N	Y(8)
Centering the graph	Y	N	N
Panning the graph with mouse/finger	Y	Y	Y
URI Details			
URI attributes (showing info or an URI)	Y	Y	Y(1)
Map allocation of URI	Y(9)	Y(9)	N
URL to resources	Y	Y	N
Open play resources	Y	Y	Y
Representing entities	Y	Y(5)	Y(5)
Non Functional			
Web based tool	Y	Y	N
Embed in web pages of third party service: ECLAP	Y	N	N
Graph Invoked by URL	Y(7)	N	N

1. Gruff presents literal attributes of URI as graph nodes, while LodLive usies a single aside panel, and LOG multiple frames, thus making simpler the comparison among nodes.
2. In Gruff: single and multiple links can be off at the same time, limited capability in tuning on all links of the same kind in the graph. In LodLive, links can be singularly turned on/off. The complexity is not managed.
3. In LOG, multiple links on/off of the same kind

4. LodLive and Gruff allow opening all or singularly, no middle way or precise control. LodLive presents a limited number of elements in some cases, and does not inform the user about the applied limitation.
5. LodLive and Gruff adopt different colours for representing different type of entities, and not icons.
6. In LOG.DISIT, the positioning of the entities and relationships is dynamically performed on the basis of a force model, in some case, this can be confusing.
7. Gruff provide support to discover inbound/outbound links (as object/as subject) only taking into account the current RDF store. LOG and LodLive perform the query on a range of SPARQL entry points (at their disposal in some database), while others can be added.
8. Gruff has a powerful zoom and large graph management; on the other hand, it is a standalone application in native code.
9. LodLive provide direct support for placing on a Map the URI if they present GPS coordinates. Integration with Map can be performed for LOG since the LOG graphs can be opened and recalled by an REST call / URL. See for example the Http://servicemap.disit.org .
10. LOG allow the loading of multiple SPARQL entry points and the web sharing of LOG graph, by sending emails with the links to reload and manipulate them

Representing entities. In complex LOD graph the fast identification of URI type is very important. Not all the URIs have a relationships with an URI formalizing its type, and it is not rare to see an URI with multiple types. The URI can be represented by using specific icons on the basis of their: (i) type (problems in the case of multiple types), (ii) information and attributes (such as some connected image), (iii) specific icon associated with the URI (e.g., image of the person for *dc:author*), (iv) specific case, for example to represent the *Blank* nodes.

III.E Summary of comparison

Table 1 reports the summary of the performed comparative analysis of Section III.

IV. LOG.DISIT.ORG COMPUTING

As described in Section III, the Linked Open Graph, LOG.Disit.org, allows opening multiple reticular RDF representations starting from different URIs (also called graph *root*) of different SPARQL entry points. All the starting URIs/URLs loaded are also listed on top of the LOG user interface. The listed URL/URI can be clicked to highlight the corresponding root URI.

In LOG graph reported in Figure 5 an algorithmic aspects related to multiple entry points is discussed. In Figure 5a, the 1:N relationships (as R0, R1, ..) are represented with a unique arc exiting from the sourcing node, N0. Among the visual browsers analysed in the state of the art, LOG.disit is the only one managing multiple SPARQL entry points and allowing the web collaboration. Circles, as R0, represents the relationship and manages the multiplicity (for example towards N1 and N2). This approach (adopted in LOG to have only one line exiting from the entity per relationship kind), allows managing the complexity of large data sets. On the other hand, it computation adds an additional complexity in LOG drawing where multiple roots may be present.

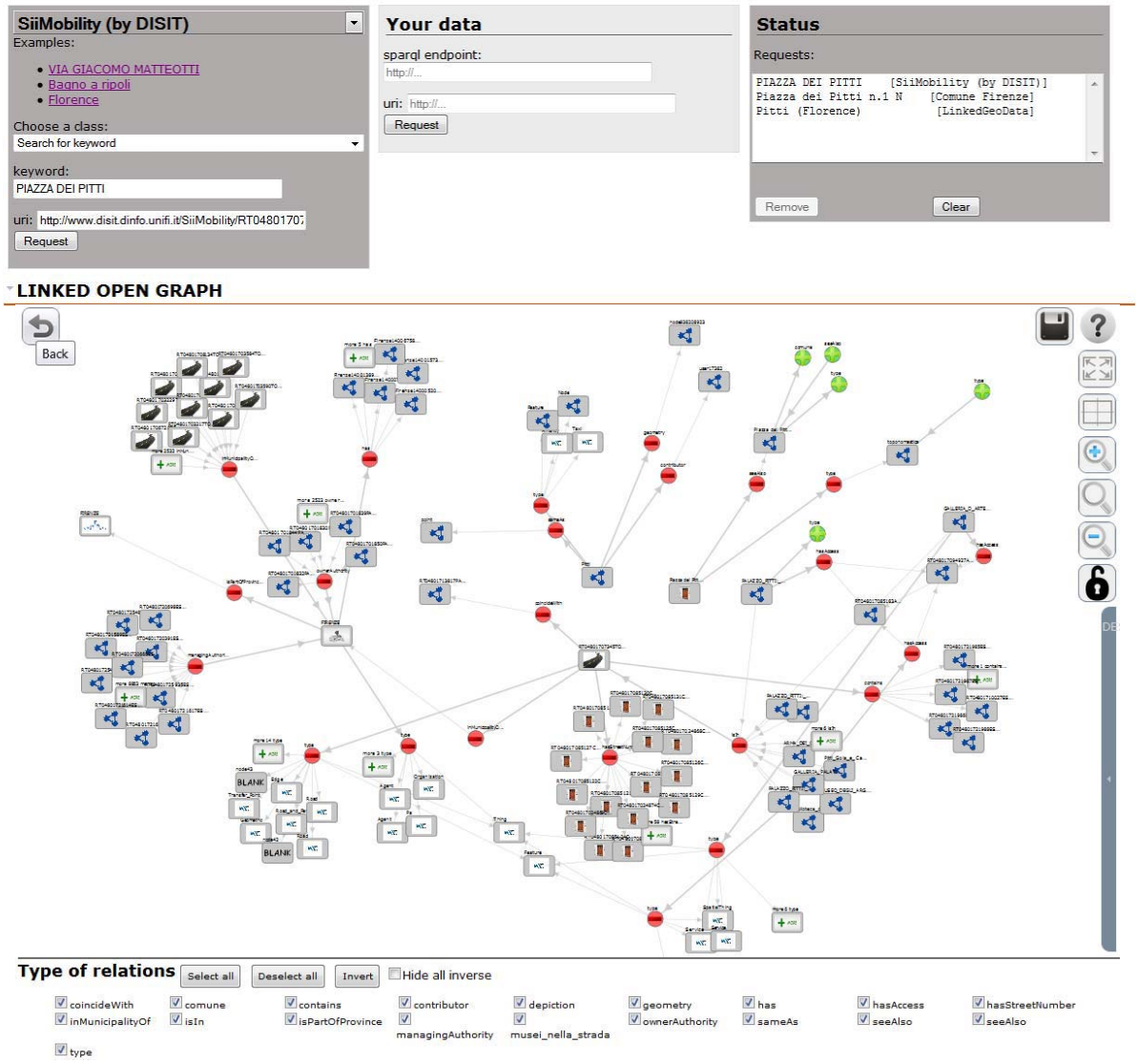

Figure 6: A LOG RDF graph with multiple URI of different entry points, expansion and relationships enabled

In Figure 5a, a LOG case with two roots is presented: N0 and N8; the two roots share node N5 that holds a double multiplicity (belonging to two graphs). When the user closes R0, with a double click: the 2 relationships related arcs dotted in Figure 5a are deleted. According to that action, a graph analysis is needed. The analysis is started by performing a labelling process from both roots N0 and N8. This allows identifying all nodes that are connected from some root (all except N2, N3) in the graph. Thus, the elements which are not connected have to be removed (see Figure 5b), for example: N2, N3, R3 and R2. In addition, shared nodes, such as node N5 lose their multiplicity. Figure 5c represents the final results after the application of the above described "*closure*" algorithm, where it is evident that some elements passed from one root to the other. A complementary operation is performed, when an inbound link of a node is opened (for example by using a query similar to that obtained in Figure 4), for example, N3 request the opening of R3, then a situation similar to Figure 5b can be reached.

In most cases, the removal of elements does not means to delete the elements from the internal graph model, but only its hidden from the graph. This approach allow to pass from less to more details in a very fast manner, but at the expenses of the loading time when data are collected form the remote RDF stores

With the **contextual menu** on the node/URI, the user may perform the analysis of the inbound and outbound relationships, or explore all the relationships (see Figure 4). Thus, in the browsing and construction of an LOG RDF graph, a number of progressive queries are performed. The graph is constructed on the basis of the resulting triples obtained from those queries: (i) some of the resulting relationships and URIs (nodes) could be already present in the graph; (ii) a node may have multiple arcs entering and exiting to/from a node. They do not have to be drawn more than once; the duplications have to be avoided by using an efficient

algorithm on the data model since real time rendering is needed. Thus, the algorithm verifies every new arc to check if it is already included or not; duplicated arcs are removed from the model. Then, nodes without arcs are also removed. The graph cleaning has to be performed every time nodes/URI and relationships/arcs are added or removed.

From the technical point of view, LOG.disit provides a server side application in PHP and exploit on client side: Javascript, JQuery, Ajax, and D3 graphic library [15].

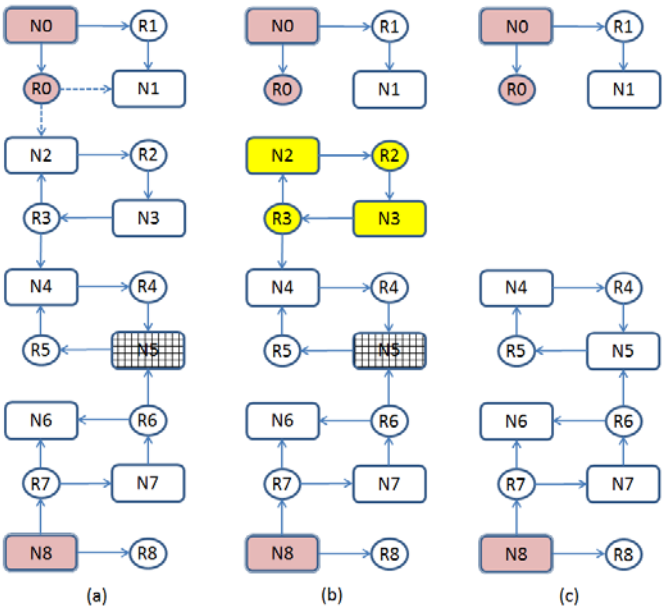

Figure 5: Graph reduction process in LOG

When the graph related to a URI needs to be created the server side script needs to retrieve from the SPARQL endpoint the information to depict the node: the type, the label, when available the *foaf:depiction* image, the predicates that are associated with the node and for all the nodes that are to be represented the type, label and depiction image. To this end, the server-side script performs the following numbered steps:

1. it is requested the *rdf:type*, *rdfs:label* and *foaf:depiction* associated with the URI;

2. it is requested for the URI the number of occurrences of each predicate using a query such as:

```
SELECT ?p (count(*) as ?c) WHERE {
  <URI> ?p ?o.
  FILTER isURI(?o)
} GROUP BY ?p
```

This is performed to have an idea of the complexity of the relations with other nodes, it can happen that a node has thousands of associations with other nodes and in this case a query that gets all the triples where the URI is the subject can be unmanageable.

3. For the predicates that are not too numerous it is requested the information of the related nodes and predicates with a query as the following:

```
SELECT ?p ?o ?l ?t ?d WHERE {
        <URI> ?p ?o.
        OPTIONAL { ?o rdfs:label ?l }
        OPTIONAL {?o rdf:type ?t}
        OPTIONAL {?o foaf:depiction ?d}
        FILTER !(?p IN (<...>,<...>)) }
```

4. for each predicate <P-URI> that is too numerous a specific query is performed such as the following:

```
SELECT ?o ?l ?t ?d WHERE {
        <URI> <P-URI> ?o.
        OPTIONAL { ?o rdfs:label ?l }
        OPTIONAL {?o rdf:type ?t}
        OPTIONAL {?o foaf:depiction ?d} }
```

to retrieve information about related nodes.

5. the same operations of steps 2, 3 and 4 are performed using the <URI> as object and not as subject of the predicate.

A special case is the one related to *blank* nodes, generally identifiers used to refer to them are valid only for the specific document that contains them and thus these identifiers cannot be used in later queries to get information about the specific blank node. Moreover, if a blank node is used in a SPARQL query it is treated as a variable matching nodes. Some RDF store solve this problem with specific extensions that are not standard and thus are difficult to be used in this context. To partially solve this problem we decided to retrieve for blank nodes also all the relations of the blank node with other nodes and send all this information to the client that needs to manage its access. This operation is limited since in case the blank node refers to another blank node this one cannot be explored. This problem may be solved in a future version using information from linked data that should contain all the blank nodes used to represent a resource.

For this reason the query used in the third step is changed to:

```
SELECT ?p ?bnode ?p2 ?o ?l ?t ?d WHERE {
        { <URI> ?p ?o.  FILTER isURI(?o)
        } UNION {
          <URI> ?p ?bnode.
          ?bnode ?p2 ?o.
          FILTER isBlank(?bnode) && isURI(?o)
        }
        OPTIONAL { ?o rdfs:label ?l }
        OPTIONAL {?o rdf:type ?t}
        OPTIONAL {?o foaf:depiction ?d}
        FILTER !(?p IN (<…>,<…>, …))
      }
```

that makes a union of the results where the URI is associated with another URI and when the URI is associated through a blank node.

IV.1 LOG usage and examples

Technically, not all ontologies and RDF models and stores have been developed by using the same methods since they have been developed by different teams, using different styles, in different periods, and exploiting different vocabularies. This implies that different approaches to model the same entities and patterns may be possible, as well as different usage of "*sameAs*", "*equivalent class*", *blank* nodes, reuse of vocabulary and concepts, etc. The LOG can be very useful to understand these differences interactively studying the RDF store from remote, to learn and to explore the possibility of reusing and connecting them each other. The LOG.disit tool, with its additional features with respect to the state of the art browsing tools, can be a very useful tool for: analyzing RDF stores and models, comparing and discovering connections and relationships among RDF stores and models, discovering eventual problems in accessible knowledge base for their future reuse and connection.

In Figure 6, an example is presented. The upper part of the screen reports the controls and the list of roots URIs included and loaded in the graph. They have been obtained from: LOD of Florence, Sii-Mobility and LinkedGeoData The resulting LOG graph reported in the Figure 6 can be accessed (in read only mode) by using http://log.disit.org/service?graph=3dfae71db76642b6ba23ce7dccb12bcf, while the URL for modifying the LOG graph has been sent to the email of the LOG graph creator only, that could decide to share. On the bottom part of the screen, the list of active relationships is reported. They can be turned on/off and the whole section inverted. After to have loaded the first URI (Pitti from Linked Geo data) the user discovered relationships (similarly to Figure 4) then decided to open the first URI related to Pitti, and worked a bit on some aspects to browse relationships. Then the decision of searching for Pitti in different SPARQL entry points (Sii-Mobility and Comune di Firenze) provoked the load of the corresponding nodes. Then a number of other nodes have been browsed with the aim of comparing the three different representations of the same entity discovering other similarities (sadly of unconnected entities) as Florence, and related streets. This process helped the user to conquer a global and integrated comparison of the aspects associated to the same topic in multiple RDF stores.

Concrete examples have to be contextualized with respect to the RDF store directly suggested in the LOG interface as follows. In the following, other examples of LOG.disit usage with connected and specialized graphs are reported.

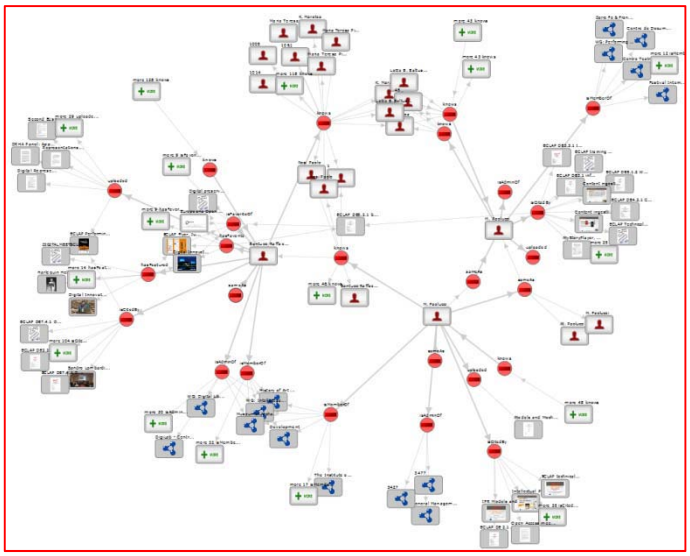

Figure 7: A LOG RDF, indirect relationshipos of two different users on ECLAP.

ECLAP RDF store contains information about content and users of the ECLAP social network (http://www.eclap.eu). In this case, the LOG could be used to (i) compare the user profile graphs of different users, (ii) discover direct and indirect relationships among users by searching and calling their entry points, (iii) explore relationships of a single user among its several connections with other social network actions and elements. The analysis can be focused on producing new metrics, new suggestions, and identifying new cause – effect relationships. The ECLAP model, via users and content are also connected to dbPedia and Geoname. In Figure 7, a study about the indirect relationships among two different users is reported. Some of the possible relationships have been disabled to focus on common favorite content and friendships.

OSIM RDF Store contains a model and data related to the University of Florence knowledge, including all structures, research lab, researchers, their publications, relationships among them, related competences of people and structures and thus a taxonomy of concepts and competences. In this case, the LOG can be useful to browse and analyze the network of experts that are working on a given topic, their relationships, the places in which they have published, the projects in which they worked, and who worked on what. The browsing of the store allows extracting more information than the simple semantic query on the user interface. There are some connections among users of ECLAP and the OSIM store since some of the users are also modeled in the OSIM store. The usage of multiple RDF stores allows understanding how these to stores could be used to create new knowledge and services. For example, learning preferences on ECLAP and providing suggestions on OSIM or viceversa. In Figure 8, a LOG graph analysing connection and structures of the same user on ECLAP and OSIM RDF stores is presented.

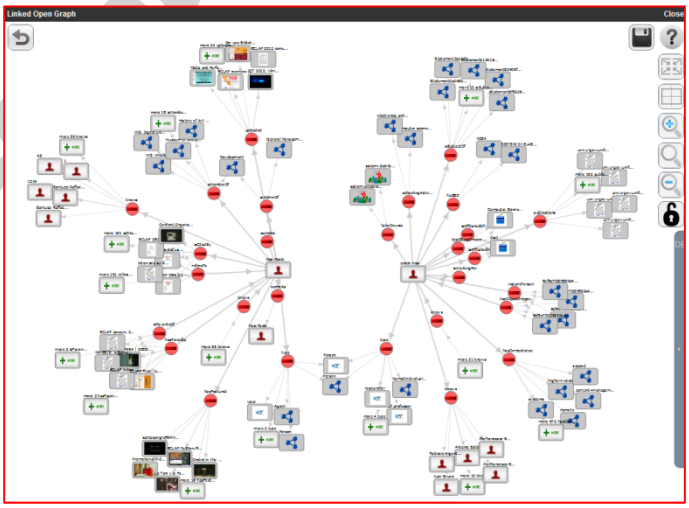

Figure 8: A LOG RDF graph analysing connection and structures of the same user on ECLAP and OSIM RDF stores.

Senato and **Camera** RDF stores contains the information related to laws and political decisions by the Italian govern, and thus also the involvements of the politicians. The two stores are not physically connected while relationships are evident in terms of laws, politicians, approvals of documents and their passages and demands from one camera to the other (the famous disputed Italian perfect bicameralism). In this case, the activation of the one URI in a store may really link to information in the other, and the complete view can be obtained only by a tool as LOG.disit.org, that allows you to join them together. Another interesting analysis can be performed to see the votes of politicians during their political life and the support they gave to different political groups and laws.

Sii-Mobility RDF Store models a large repository of geolocalized data regarding Smart City concepts and data connected to Tuscany: topographic information, administration, services, statistics, time line of busses, parking status, weather forecast. In this case, the LOG tools is very useful in the hands of potential SME interested in developing mobile applications during *Hackathon* for the definition of innovative Smart City services. For example, to (i) discover and understand the model and the information associated to a given service in the city, (ii) discover connections and similarities among different open data set of public administration, (iii) study the integration of open data with geographic information. In this particular case, Sii-Mobility provides a user interface to perform geographic queries and from the results the LOG graph can be open (http://servicemap.disit.org).

V. Conclusions

The navigation on internet accessible RDF stores is becoming every day more relevant. They are frequently based on local and commonly accepted ontologies and vocabularies to set up large knowledge base to solve specific problems of modelling and reasoning. The growing needs of such structures increased the need of having flexible and accessible tools for RDF store browsing taking into account multiple SPARQL entry points to create and analyse reticular structure and scenarios of remote stores. The LOG.disit tool presented in this article provides innovative features solving a number of problems related to graph computation to cope with high complexity of large LOD graphs with a web based tool. The complexity is mainly managed by providing tools for (i) progressive browsing of the graphs, (ii) allowing graph composition, (iii) providing support to pose specific and local queries, (iv) allowing the progressive discovering/selection of instances, (v) editing the LOD graph to create your secifica case studies and link discovering. A comparative analysis with reference solutions at the state of the art has been also provided, showing that LOG tool presents a number of innovative and very useful features for RDF store analysis and development. In general, RDF stores have been developed by using different methods, by different teams, using different styles, in different periods, and exploiting different vocabularies. The Linked Open Graph, LOG, is a web based tool for collaborative analysis, browsing and navigation on multiple SPARQL entry points. The LOG.disit tool, with its additional features with respect to the state of the art browsing tools, can be very useful to understand these differences interactively studying the RDF store from remote, to learn and to explore the possibility of reusing and connecting them each other.

The LOG tool is used in multiple projects as ECLAP for cultural heritage (http://www.eclap.eu), Sii-Mobility for smart city [16] and ICARO for smart cloud ontology analysis. It has been validated using multiple public accessible RDF stores such as: dbPedia, Europeana, Getty Vocabulary, Camera and Senato, GeoLocation, etc., putting in evidence the different cases and usage of LOG tools in the different scenarios, with a specific stress on the analysis of multiple RDF stores on the same graph. The recent improvement of the LOG allowed the user to embed the linked open graph in third party web pages and tools.

Bibliography

[1] T. Berners-Lee, “Linked Data”, http://www.w3.org/DesignIssues/LinkedData.html, 2006.

[2] C. Bizer, T. Heath and T. Berners-Lee (2009) Linked Data - the story so far. Int. Journal on Semantic Web and Information Systems, 5, (3), 1-22.

[3] G. Klyne, J. Carroll, “Resource Description Framework (RDF): Concepts and Abstract Syntax - W3C Recommendation”, 2004

[4] FOAF, http://www.foaf-project.org/

[5] G. Tummarello, R. Delbru, and E. Oren. 2007. Sindice.com: weaving the open linked data. In Proc. of ISWC'07/ASWC'07, Springer, Berlin, Heidelberg, pp.552-565.

[6] O. Hartig, C. Bizer, J.-C. Freytag. 2009. Executing SPARQL Queries over the Web of Linked Data. In Proc. of ISWC '09, Springer, pp.293-309.

[7] S. Ramakrishnan and A. Vijayan. 2014. A study on development of cognitive support features in recent ontology visualization tools. Artif. Intell. Rev. 41, 4 (April 2014), pp.595-623.

[8] Protégé http://protege.stanford.edu/

[9] iSPARQL, http://oat.openlinksw.com/isparql/index.html,

[10] O. Ambrus, K. Moller, S. Handschuh, “Konduit VQB: a Visual Query Builder for SPARQL on the Social Semantic Desktop”, proc of VISSW2010, IUI2010, 2010, Hong Kong, China.

[11] A. Russell, P.R. Smart, D. Braines, Dave, N.R. Shadbolt, “NITELIGHT: A Graphical Tool for Semantic Query Construction”, In, *SWUI 2008,* Florence, Italy,

[12] Gfacet, http://www.visualdataweb.org/gfacet.php

[13] D. V. Camarda, S. Mazzini, A. Antonuccio. 2012. LodLive, exploring the web of data. In *Proc. of the* I-SEMANTICS '12, ACM, pp.197-200. http://lodlive.it

[14] P. Bellini, P. Nesi, "Modeling Performing Arts Metadata and Relationships in Content Service for Institutions", Multimedia Systems Journal, Springer, 2014. http://www.eclap.eu

[15] D3, Data-Driven Documents, http://d3js.org/

[16] P. Bellini, P. Nesi, N. Rauch, "Smart City data via LOD/LOG Service", Workshop Linked Open Data: where are we?, LOD2014, org. by W3C.

[17] Prud'hommeaux, E., Seaborne, A., SPARQL Query Language for RDF, http://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/

[18] OTN, Ontology of Transportation Networks, Deliverable A1-D4, Project REWERSE, 2005 http://rewerse.net/deliverables/m18/a1-d4.pdf

[19] http://dublincore.org, http://dublincore.org/documents/dcmi-terms/

[20] VCARD, http://www.w3.org/TR/vcard-rdf/

[21] wgs84, http://www.w3.org/2003/01/geo/wgs84_pos

[22] dbPedia, http://dbpedia.org/resource/

- LOG tool provides a large number of new features for managing LDs on web
- Understanding RDF storesi s becoming a fundamental skill
- Exploiting linked data without copying them locally is becoming a must
- Managing complexity of Linked Data on web is needed

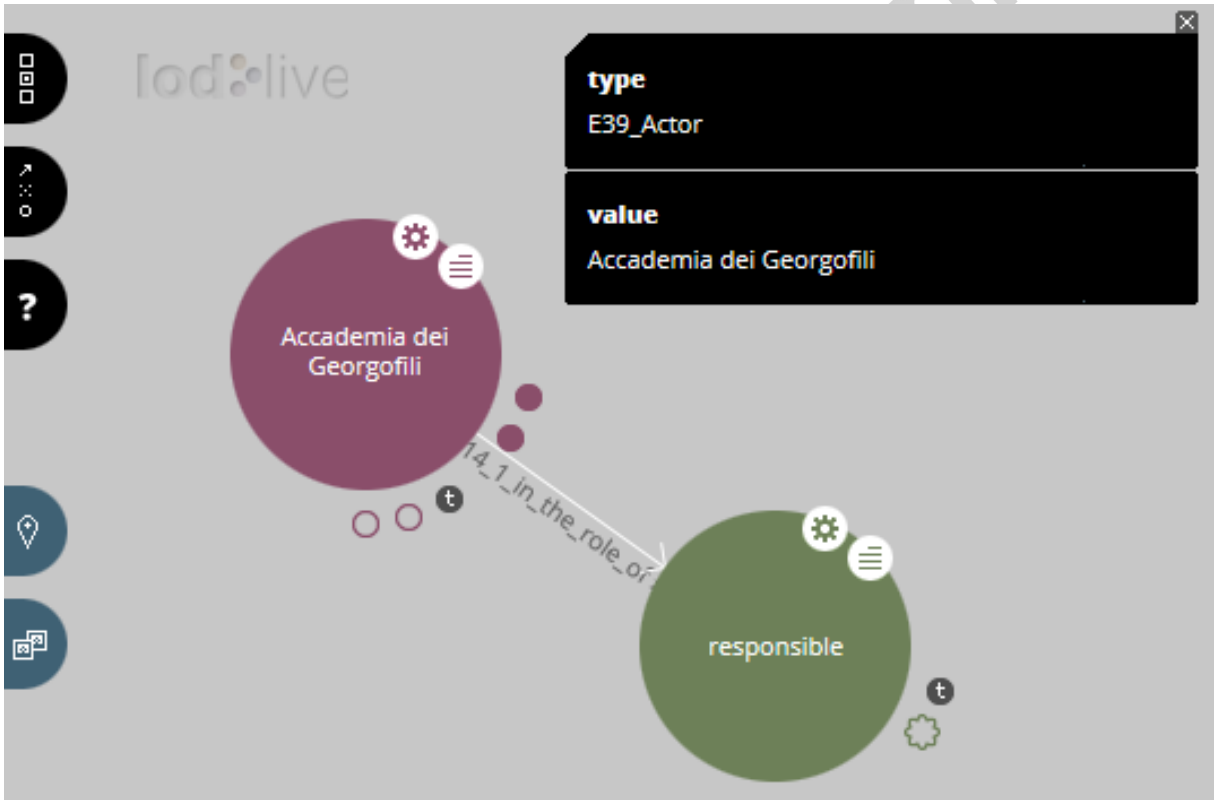

Figure 1: URI rendering with LodLive

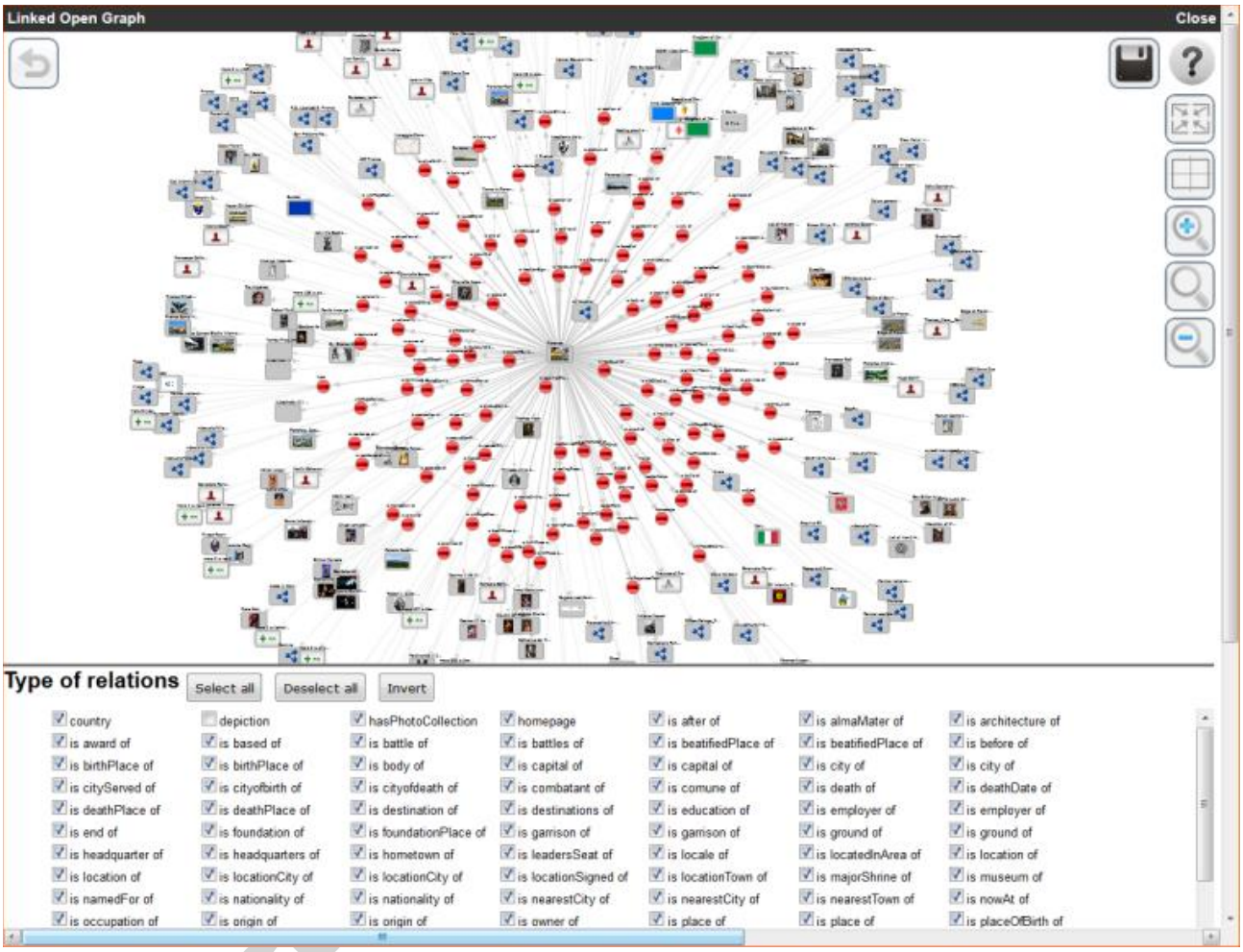

Figure 2: Florence URI on dbPedia, via LOG.DISIT, providing 364 elements: 237 entities and 127 multiple relatioships (red circles).

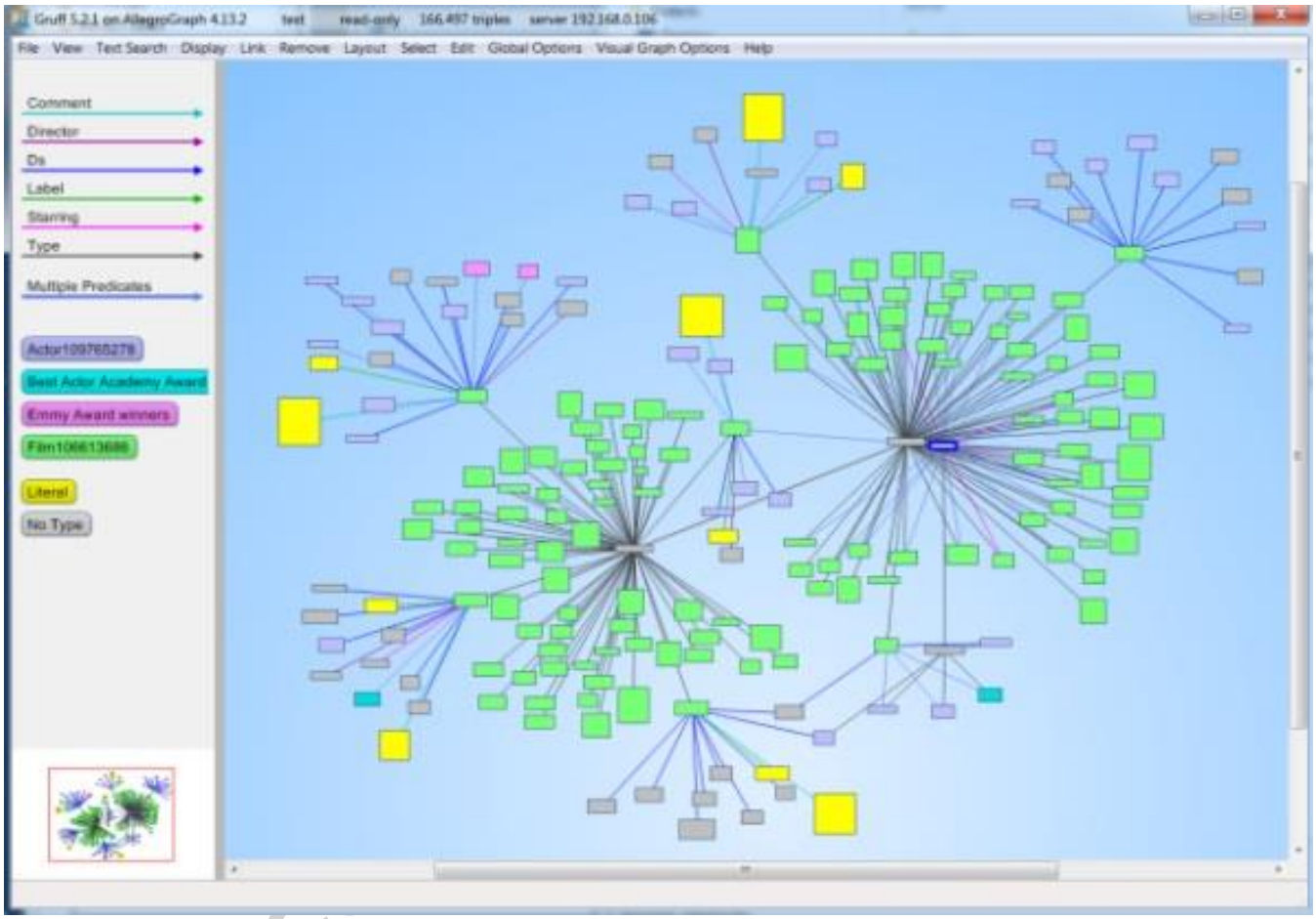

Figure 3: URI on a dbPedia segment, via Gruff.

Figure 4: Results of the relationships analysis for http://www.w3.org/2002/07/owl#Thing URI LOG.DISIT.org

.

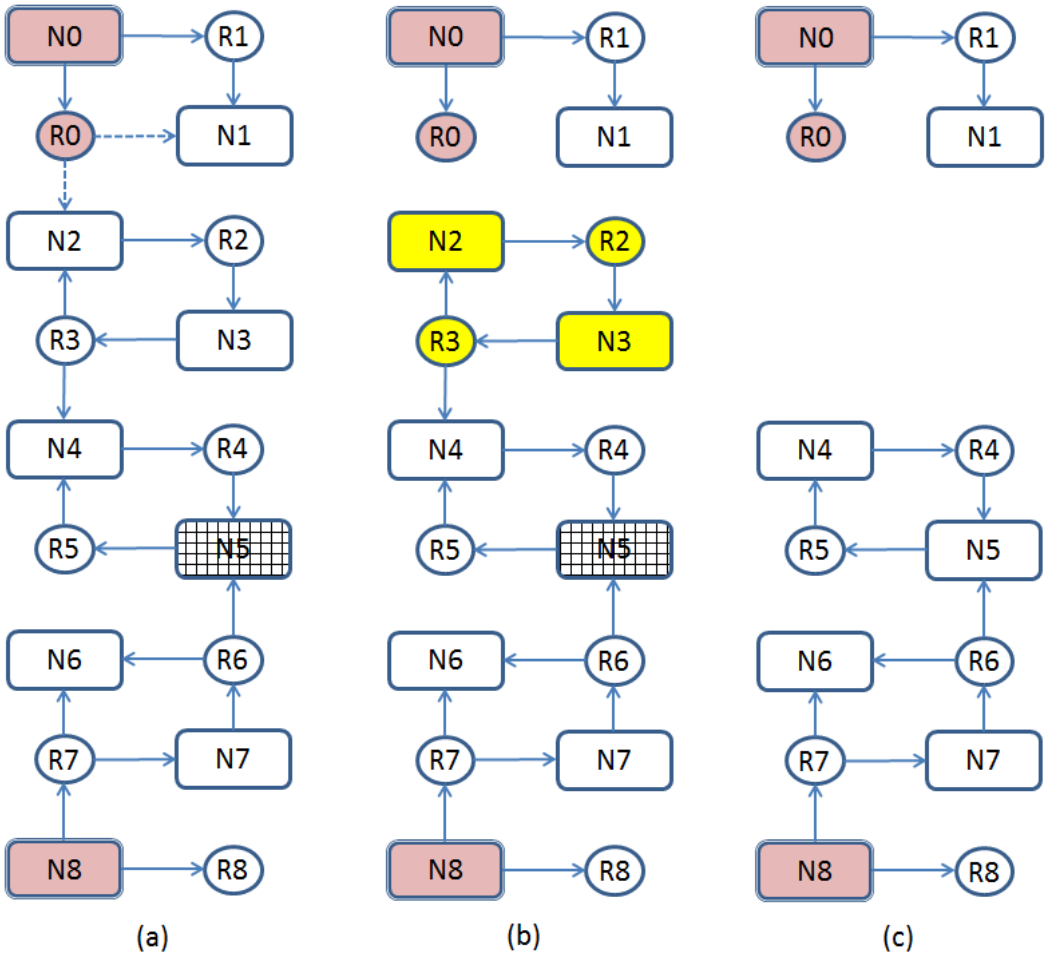

Figure 5: Graph reduction process in LOG

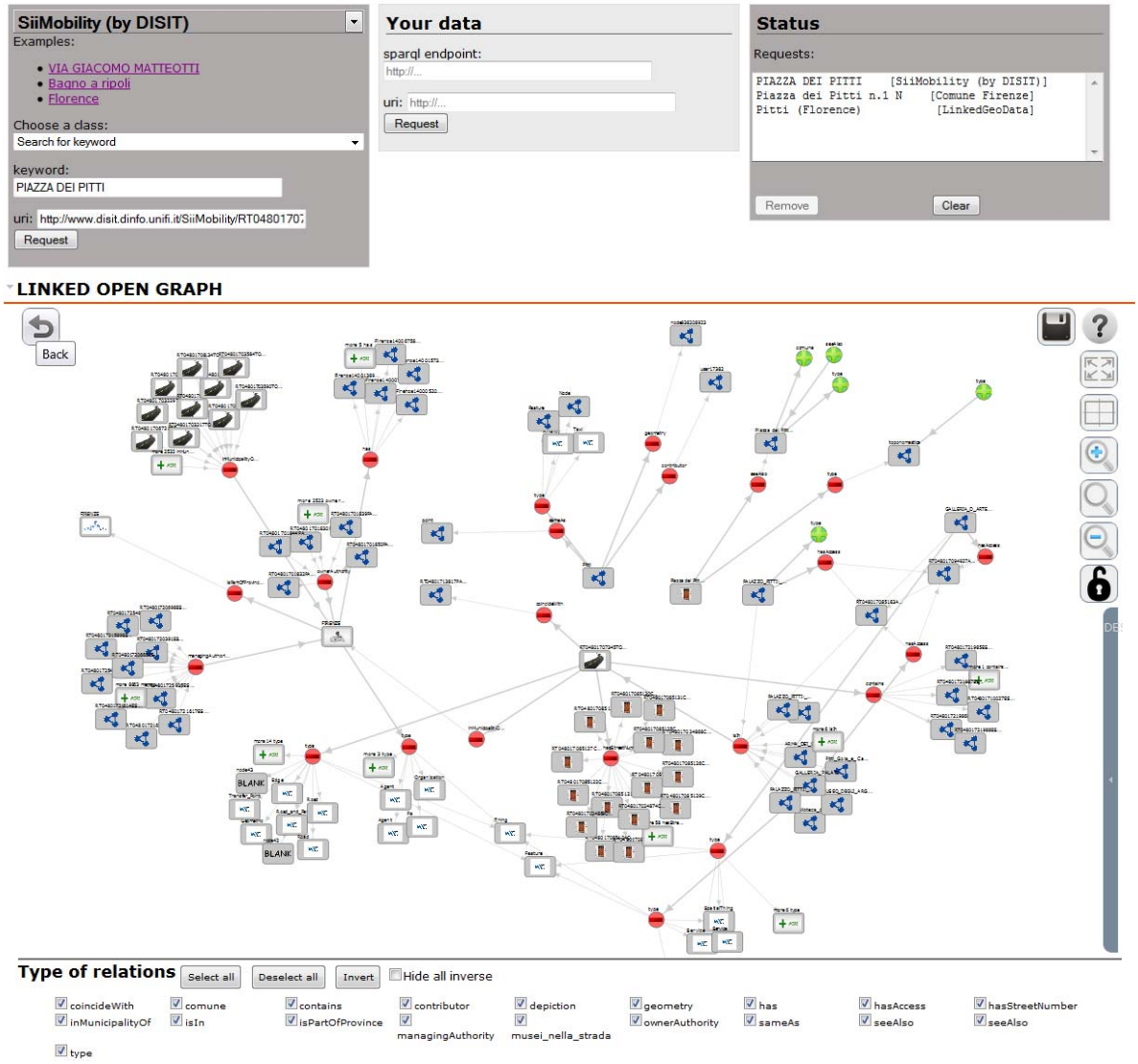

Figure 6: A LOG RDF graph with multiple URI of different entry points, expansion and relationships enabled

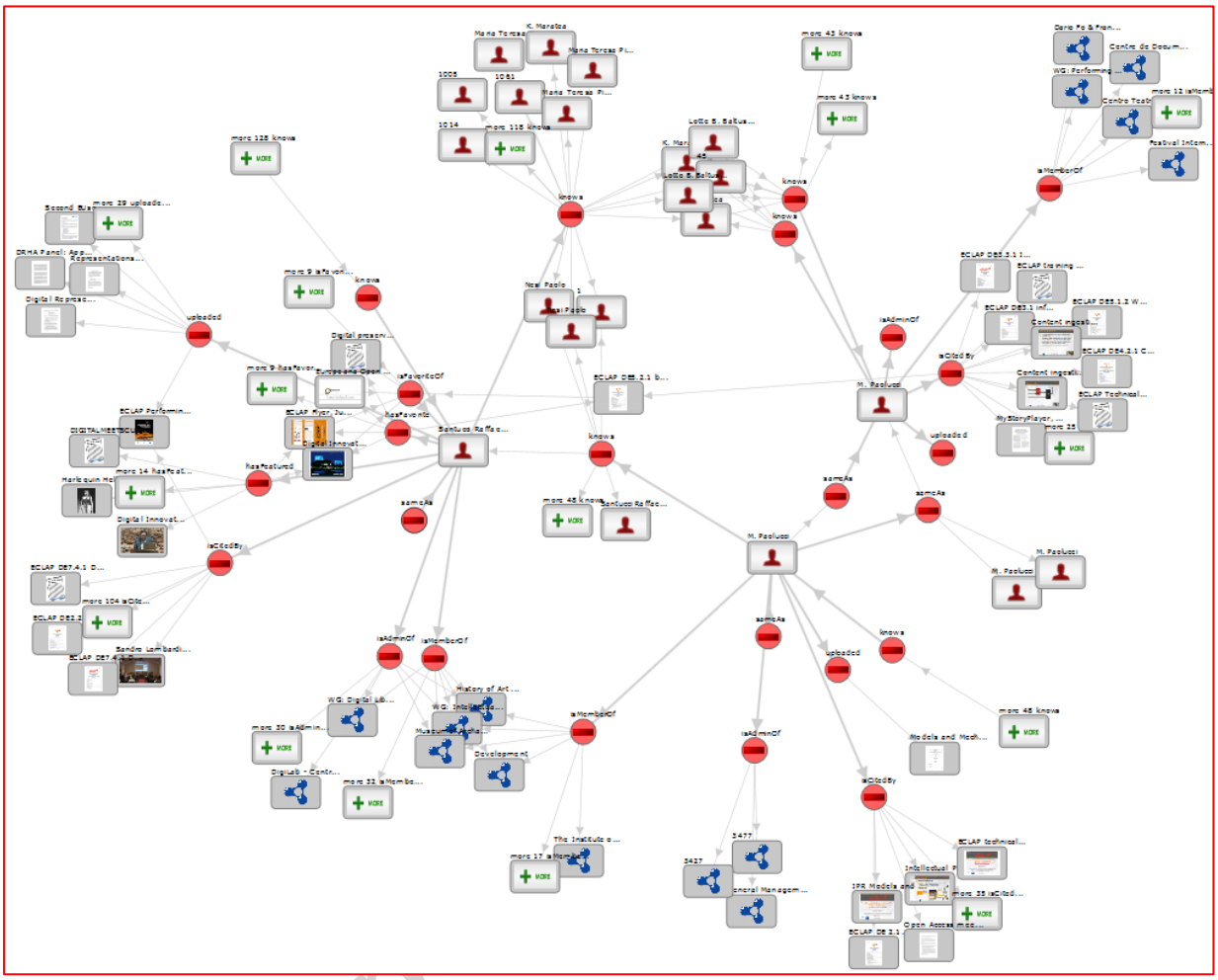

Figure 7: A LOG RDF, indirect relationshipos of two different users on ECLAP.

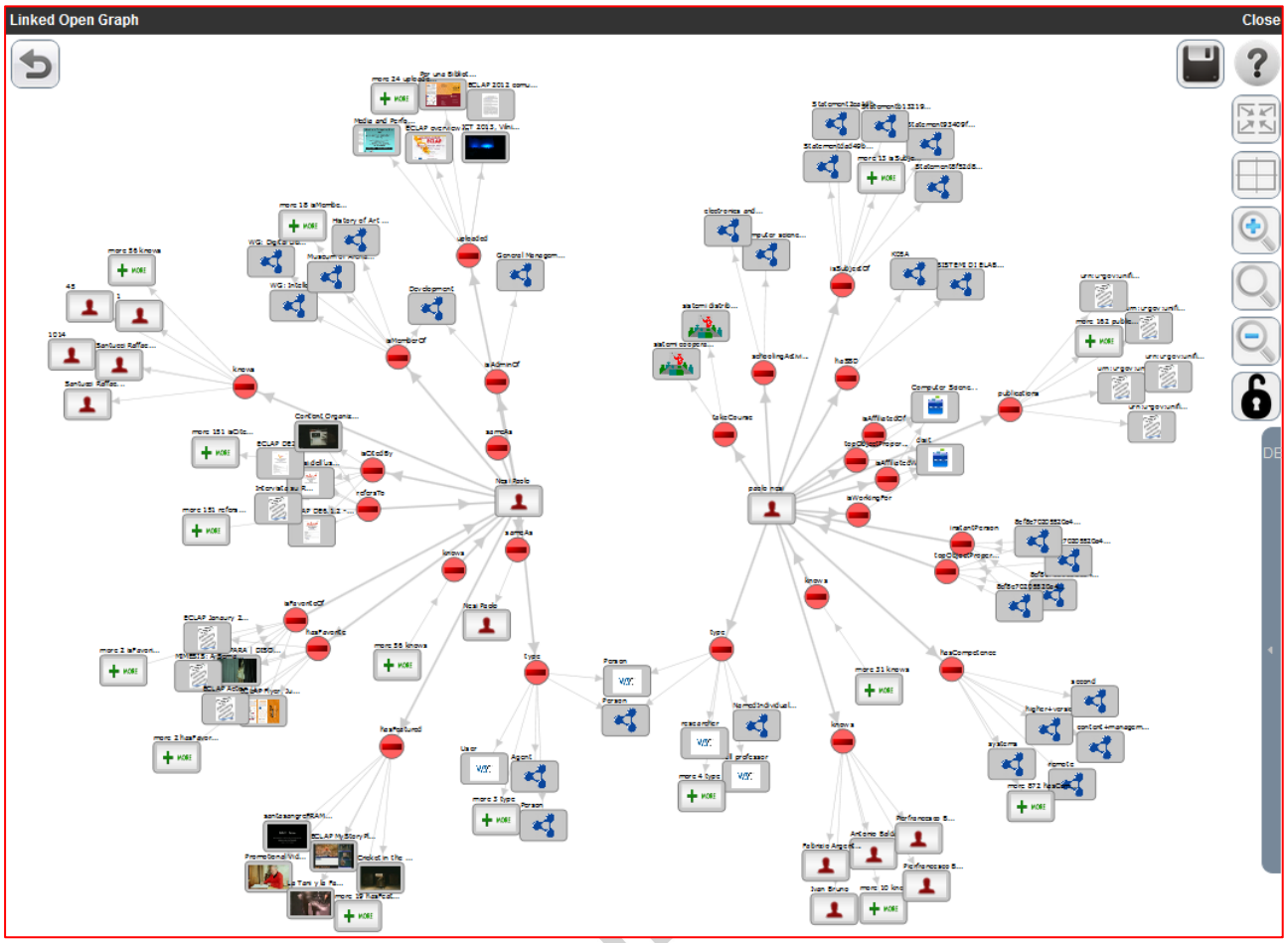

Figure 8: A LOG RDF graph analysing connection and structures of the same user on ECLAP and OSIM RDF stores.