future internet

MDPI

Article

Interactive 3D Exploration of RDF Graphs through Semantic Planes

Fabio Viola [1,2,*], Luca Roffia [1,2], Francesco Antoniazzi [1,3], Alfredo D'Elia [2], Cristiano Aguzzi [1,2] and Tullio Salmon Cinotti [1,2]

1 Department of Computer Science and Engineering (DISI), University of Bologna, 40126 Bologna, Italy; luca.roffia@unibo.it (L.R.); francesco.antoniazzi@unibo.it (F.A.); cristiano.aguzzi@unibo.it (C.A.); tullio.salmoncinotti@unibo.it (T.S.C.)

2 Advanced Research Center on Electronic Systems "Ercole De Castro" (ARCES), University of Bologna, 40125 Bologna, Italy; alfredo.delia4@unibo.it

3 Centro Nazionale per la Ricerca e Sviluppo nelle Tecnologie Informatiche e Telematiche (INFN CNAF), 40127 Bologna, Italy

* Correspondence: fabio.viola@unibo.it

Received: 19 July 2018; Accepted: 14 August 2018; Published: 17 August 2018

Abstract: This article presents Tarsier, a tool for the interactive 3D visualization of RDF graphs. Tarsier is mainly intended to support teachers introducing students to Semantic Web data representation formalisms and developers in the debugging of applications based on Semantic Web knowledge bases. The tool proposes the metaphor of semantic planes as a way to visualize an RDF graph. A semantic plane contains all the RDF terms sharing a common concept; it can be created, and further split into several planes, through a set of UI controls or through SPARQL 1.1 queries, with the full support of OWL and RDFS. Thanks to the 3D visualization, links between semantic planes can be highlighted and the user can navigate within the 3D scene to find the better perspective to analyze data. Data can be gathered from generic SPARQL 1.1 protocol services. We believe that Tarsier will enhance the human friendliness of semantic technologies by: (1) helping newcomers assimilate new data representation formats; and (2) increasing the capabilities of inspection to detect relevant situations even in complex RDF graphs.

Keywords: semantic web; linked data; internet of things; visualization; 3D; computer graphics

1. Introduction

The Semantic Web [1] introduced to Information Technology a novel way to intend the Web and its resources. In fact, the Web is evolving from a set of interlinked human-understandable HTML pages, towards a set of interlinked machine-understandable data. The implementation of such innovative vision started with a layered architecture. First, data are modeled, stored and retrieved as a set of RDF (https://www.w3.org/RDF/) triples composed of a subject, a predicate and an object: as the object of a triple can assume the role of the subject of other triples, the underpinning data structure can be represented as a directed labeled graph. Second, RDFS (https://www.w3.org/TR/rdf-schema/) and OWL (https://www.w3.org/OWL/) provide a set of constructs to define an ontology [2] with the concepts and relationships considered relevant for a given application domain. Through the ontology, it is possible to assert meaningful RDF triples. Third, data can be stored into RDF stores by means of SPARQL Updates (https://www.w3.org/TR/sparql11-update/) and retrieved through SPARQL Queries (https://www.w3.org/TR/rdf-sparql-query/). An RDF store can be fully managed by a SPARQL protocol service through the SPARQL 1.1 Protocol (https://www.w3.org/TR/sparql11-protocol/). Gyrard et al. [3] described and compared four ontology repositories for IoT and smart

cities (analysis of May 2018): Ready4SmartCities, OpenSensingCity, LOV and LOV4IoT. LOV4IOT includes 448 ontologies, a high number considering that the repository is limited to Smart Cities or IoT vocabularies. LOV (Linked Open Vocabularies) (https://lov.linkeddata.es/dataset/lov) [4] is an innovative observatory of the semantic vocabularies ecosystem. In this paper, authors proposed an interesting overview of the impressive growth of the repository: in approximately four years (March 2011–June 2015), the number of vocabularies hosted by LOV grew from less than 100 to 511 (66.14% of which developed in English). As of August 2018, the number is 650. This is significant evidence of how Semantic Web technologies are gaining momentum in the last decade, also thanks to the spread of the Internet of Things.

How can the content of an RDF knowledge base (KB) be visualized? The simplest way could be a single table with three columns (i.e., subject, predicate, and object). Unfortunately, this solution scales very poorly. In fact, even with a few dozen triples, dominating the content becomes almost impossible. An alternative is represented by a (possibly very large) set of tables where every property is mapped into a relation. Even though relational tables have been widely used to optimize data storage and retrieval [5], this may not be the best choice for an effective visualization of RDF data. In fact, data visualization should provide an expressive and clear representation to easily grasp complex concepts and identify recurring structures (e.g, subgraphs characterized by a similar topology as in networks of hundreds of sensors semantically mapped in a graph). One of the most natural (and diffused) ways to represent an RDF KB is a graph, as it directly maps with the RDF data model: RDF data can be, as previously mentioned, represented as a directed and labeled graph where the subject and the object of each statement are linked by an edge whose label is the predicate of the statement. As this graph can be very complex (i.e., composed by many nodes and many edges), a tool for its visualization should address a set of issues and requirements that can be summarized as:

***p*0 Pre-Filtering**: When dealing with RDF knowledge bases, the amount of triples contained in the data store may be really high (e.g., DBpedia contains more than 6.6 M triples). Representing in a graphical way such a large amount of data is usually both ineffective (it is hard for the user to retrieve the information he looks for) and inefficient (it is heavy from the point of view of memory occupation and computational resources). Then, a pre-filtering mechanism is required to extract the subgraph that is really relevant for the user from the full knowledge base.

***p*1 Node placement**: Node positioning should be smart enough to avoid overlapping with other graphical elements such as edges or labels. The complexity of this task grows with the size of the KB. If possible, linked resources should be placed close to each other to easily gather as much information information as possible in a glimpse.

***p*2 Incremental approach**: Often, only a small portion of a large KB needs to be inspected. Creating the visualization that fits the user needs may require a series of steps (e.g., to specify the information that must be represented and how it should be done). Thus, the incremental building of the view should be supported.

***p*3 Filtering**: Filtering must be as flexible as possible to focus on the parts considered relevant for a task by hiding/showing information. Providing powerful filtering features in a user-friendly way is often a difficult task. It is, nevertheless, very important because of the close relationship to one of the techniques that are usually used in designing and debugging queries. According to an incremental progressive method, every step consists in applying a new condition to the previous query: in that way, a less error-prone coding technique, while at every step a new filter is applied to results, until the required level of precision is reached in the request.

***p*4 Support for RDFS and OWL**: Both RDFS and OWL must be supported. The user should be able to select and filter the graph content by means of concepts such as class, domain and range of properties, datatype and object properties distinction.

***p*5 Support for a high number of scenarios**: Semantic Web and Linked Data technologies may be applied to totally different and heterogeneous domains even within the same application. The datasets in the Linked Open Data cloud mainly belong to seven domains (cross-domain,

geographic, media, life sciences, government, user-generated content, and publications) [6], while in the Internet of Things (IoT) where SW technologies are often applied, the application domains are more than 50 [7]. Depending on the specific use case, the end user may be interested in completely custom visualization perspectives.

In this article, we propose Tarsier: an interactive and ontology agnostic tool for the 3D visualization of RDF graphs. Tarsier implements all the above-mentioned features and it allows partitioning the KB into semantic planes. A semantic plane can be defined as a set of RDF terms sharing a common meaning and it can be created directly or indirectly through standard SPARQL 1.1 queries. We argue that this approach would help to understand or extract the structure of represented data by following a common mental approach: splitting the KB among planes, each of them related to a specific concept. The user can incrementally build a view by adding and/or removing information according to its actual needs and splitting the information among planes. Relationships among resources in different planes are in such a way emphasized. Furthermore, the user may still maintain a view on the rest of the knowledge base, if needed. To the best of the authors' knowledge, none of the existing tools provides a similar feature and we argue that many application domains may benefit of a multiplanar visualization. Eventually, Tarsier is also suitable to view reified knowledge bases [8], since it allows the distinction among the raw triples, the statements reifying them and the information about the statements (i.e., meta information). Very poor support to reification is currently provided by the existing tools.

The idea of Tarsier is grounded on the authors' experience in teaching Semantic Web technologies in the "Interoperability of Embedded Systems" course held at the School of Engineering and Architecture of the University of Bologna. The course is focused on the application of Semantic Web data representation means to context-aware IoT applications. Examples of these applications are presented in [9,10], respectively, belonging to the areas of Domotics and Electro-mobility with knowledge bases hosting up to one hundred thousand triples. On top of this teaching experience, the authors believe that the learning curve of RDF, RDFS/OWL and SPARQL is often steep, due to the strong linked nature of data. Tarsier should support students learning to deal with RDF knowledge bases by providing an interactive exploration tool for small- or medium-sized knowledge bases (up to a few thousand triples). Through Tarsier, students can play with classes, instances, relationships using filters to toggle their visibility or isolate the interesting entities by moving them on proper semantic planes. The tool would help the students during the so-called "sensemaking" activity that consists in understanding the content and overall structure of an ontology [11]. Furthermore, we believe that Tarsier can also be helpful for developers to inspect and debug applications based on Semantic Web knowledge bases.

The rest of the paper is organized as follows: Section 2 presents the proposed tool, starting from the initial considerations on the motivations which brought to its conception. Then, its internal software architecture is explained and examined in detail. In Section 3, the proposed approach is shown in practice through examples: several use cases are introduced and commented. In Section 4, an evaluation of Tarsier is proposed based on user surveys and performance analysis. Section 5 summarizes the state of the art of the visualization of semantic KBs, with an overview of the existing tools. Finally, in Section 6, conclusions are drawn and future works are outlined.

2. Tarsier: Splitting Data among Semantic Planes

In this section, we present Tarsier and the approach based on the concept of Semantic Plane. The approach is described in Section 2.1. The architecture of the software is presented in Section 2.2, while implementation details are in Section 2.3. The full list of features is detailed in Section 2.4. The mechanism exploited by Tarsier to identify the elements in a graph is shown in Section 2.5, while Section 2.6 describes the User Interface of the tool.

2.1. Semantic Planes

The main contribution of Tarsier is the ability to visualize an RDF graph (or portion of it) subdivided into different layers, built according to the user's needs. We name these layers semantic planes, since every layer is built to group a set of resources (and optionally their attributes) sharing a set of common semantic features. The meaning conveyed by a semantic plane may be very simple (e.g., all resources belonging to the class `foaf:Person`), or the result of a more complex filtering (e.g., the set of resources belonging to the class `foaf:Person` that work on the same project but do not know each other). Adopting such a layered visualization allows:

- focusing on the information considered meaningful by looking at the related plane, while preserving a non-intrusive view on the rest of the knowledge base;
- focusing on the incoming and outgoing edges of a subgraph (i.e., semantic connection between planes).

As detailed in the following sections, semantic planes are the results of filtering and this feature is accessible for both inexperienced and advanced users. In fact, semantic planes may be created by selecting over the list of properties (i.e., datatype or object), classes, instances and blank nodes or through SPARQL queries. Both filtering operations can be iterated and combined to refine the content of semantic planes.

2.2. Tarsier Architecture

Tarsier (https://github.com/desmovalvo/tarsier) was designed following a client-server architecture, depicted in Figure 1. This architectural choice is motivated by the need to pre-process a potentially very large set of data (i.e., to subdivide RDF Terms among classes, instances, datatype and object properties) while still providing light clients that only focus on the drawing task. Server-side, the main components are:

- A config manager, through which the server can be configured.
- A client for SPARQL endpoints, to retrieve data from the desired datasets.
- A Cache Manager. Since Tarsier is intended to be used also with dynamic systems where the KB evolves quickly, the application creates a snapshot of the knowledge base: (1) to avoid changes that would disrupt the user process of analysis; and (2) to have a local cache that speeds up every query to data. The user is able to update the local storage producing a new snapshot.
- A data extractor that performs the above-mentioned identification of RDF terms. The resulting information is then organized in a data structure that helps the client to easily retrieve all the elements needed to draw and apply the filters selected by the user. The data extractor performs its task through a set of SPARQL queries detailed in Section 2.5.
- The HTTP interface through which client and server communicate.

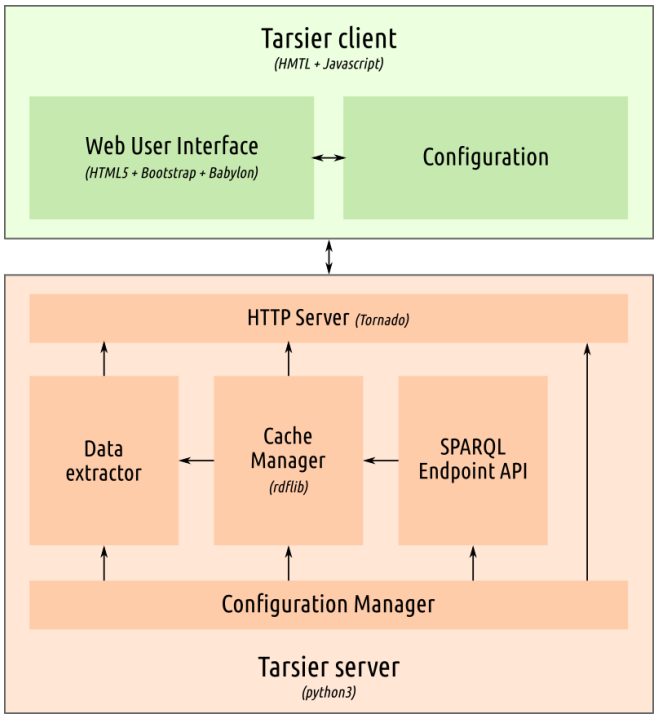


Figure 1. Software architecture of Tarsier. Implementation details are reported with the italic font.

2.3. Implementation

From the implementation point of view, Tarsier server is a Python 3 application that through the framework Tornado (http://www.tornadoweb.org/en/stable/) provides an HTTP interface to receive requests from the clients. The server can be configured through a proper YAML Semantic Application Profile (YSAP) file (a YAML-encoded in the JSAP format: http://wot.arces.unibo.it/TR/jsap.html) containing the port of the server and all the SPARQL queries needed by the Data Extractor. Tarsier server relies on `rdflib` to maintain a local cache.

Client-side, Tarsier is a Javascript application that exploits HTML5, and in particular the canvas element, to build a 3D representation of the knowledge base. While the UI is built using the framework Bootstrap (https://getbootstrap.com/), the drawing part is in charge of Babylon JS (https://www.babylonjs.com/). Babylon JS was selected because of its support to hardware acceleration.

The client is started with a default configuration that can be overwritten by loading a YAML file (that hosts the parameters for a set of SPARQL endpoints and all the settings to customize the drawing) and modified at run-time through the UI. The configuration file may also contain saved SPARQL queries to easily recall the most frequently used ones.

2.4. Features

Tarsier is characterized by a set of features summarized in the following list.

Initial Knowledge Base : Tarsier is a visualization tool for RDF graphs. As such, it allows to view the whole content of a graph through the canvas. However, RDF graphs may be very large, hosting a high number of triples too difficult to represent in an effective way. For this reason, Tarsier supports a pre-filtering of the knowledge base through SPARQL Construct queries (addressing in this way the requirement $p0$) that can be also loaded from a file. In this way, the user may dominate the complexity of the underlying knowledge base focusing only on the information considered relevant for the current task. An example is shown in Figure 2.

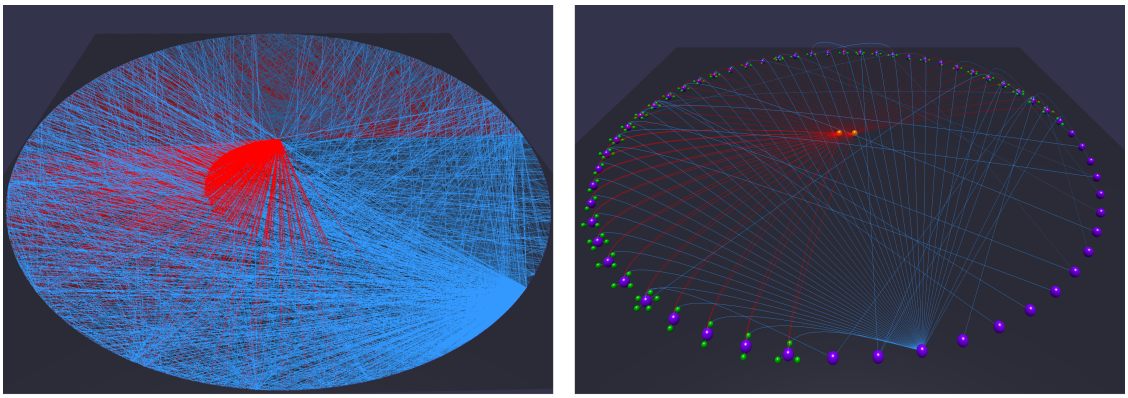


Figure 2. RDF graphs can host a number of triples too high to be effectively and efficiently visualized (**a**), but a prefiltering stage can help to visualize only a subgraph of interest (**b**).

Support for RDFS and OWL : Despite being ontology agnostic (to adapt to different use cases, as suggested by point *p*5), Tarsier, through its data extractor (see Section 2.5), is able to detect classes, datatype properties and object properties through the use of RDFS and OWL constructs (requirement *p*4). Comments and labels are also retrieved with the specific predicates.

Visualization techniques : Being able to quickly distinguish classes from other resources, datatype from object properties and `rdf:type` relationships above all may significantly speed up the analysis process (Figure 3a). Therefore, Tarsier adopts a classification algorithm based (mostly) on a set of SPARQL queries (detailed in Section 2.5) to identify classes, instances of classes, blank nodes, object and datatype properties and `rdf:type` relationships and paint each of them with a different color. Furthermore, a smart placement algorithm allows facing requirement *p*1, representing items through the following scheme: classes and individuals are represented as equidistant spherical meshes on two different circumferences. The circle dedicated to classes is the innermost, since usually the number of concepts is less than the number of instances. Blank nodes lay on a third circumference. Datatype properties of an instance are equidistant spheres placed on a circumference centered on the instance (Figure 3c). Further research will be carried out to design and test different arrangement methods.

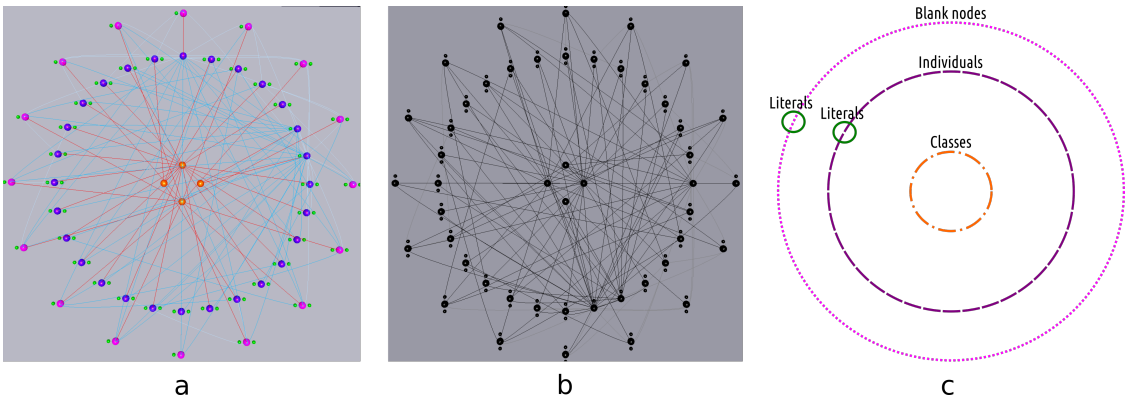


Figure 3. The classification of RDF terms among blank nodes, individuals, classes or literals as well as data and object properties, bound to using colors provide a more intuitive visualization (**a**), if compared to a monochrome one (**b**), the drawing strategy adopted by Tarsier (**c**) .

Filtering : The filtering mechanism (Figure 4) implemented by Tarsier allows selecting items through UI or SPARQL queries and decide what action to perform. Selection can be related to classes, instances, datatype or object properties as well as literals, URIs or blank nodes. The action

consists in showing or hiding selected meshes as well as moving them across layers. Every filter applies to the current visualization, allowing incremental filtering. This mechanism allows iteratively building a visualization that fits the user need even for novice users. Tarsier's filtering mechanism meets the requirements identified in points *p*2 and *p*3.

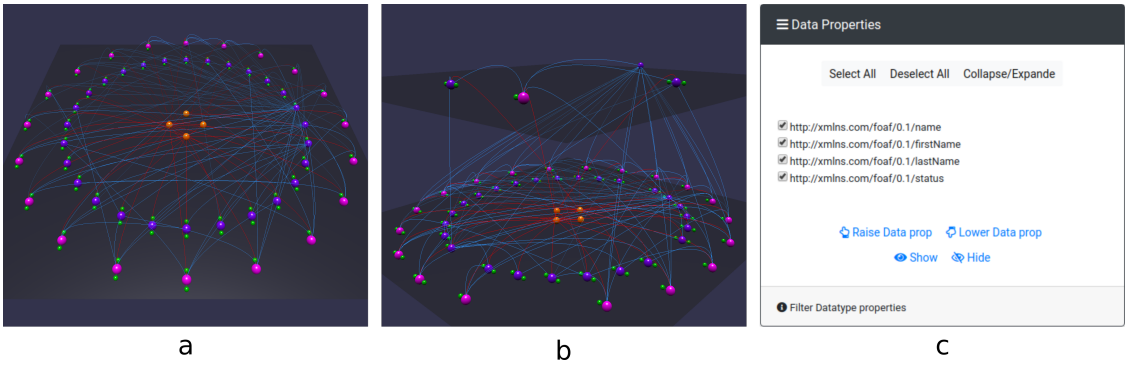


Figure 4. Filtering helps to gradually build the desired visualization of data: an example knowledge base (**a**); the result of filtering (**b**); and one of the UI boxes through which filtering can be applied (**c**).

2.5. Data Extractor

Data extracted from the triple store is classified through a set of SPARQL queries. First, classes (and details, if present) are retrieved with the SPARQL query proposed in Listing 1.

Listing 1: Classes.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
SELECT DISTINCT ?class ?label ?comment
WHERE {
  { ?resource rdf:type ?class .
    OPTIONAL { ?class rdf:label ?label } .
    OPTIONAL { ?class rdf:comment ?comment }
  }
  UNION {
    ?class rdf:type owl:Class .
    OPTIONAL { ?class rdf:label ?label } .
    OPTIONAL { ?class rdf:comment ?comment }
  }
  UNION {
    ?class rdf:type rdfs:Class .
    OPTIONAL { ?class rdf:label ?label } .
    OPTIONAL { ?class rdf:comment ?comment }
  }
}
```

The data extractor also retrieves a list of Datatype properties and Object properties, respectively, with the SPARQL queries of Listings 2 and 3.

Listing 2: Datatype Properties.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
SELECT DISTINCT ?prop ?domain ?range ?label ?comment
WHERE {
  { ?prop rdf:type owl:DatatypeProperty .
    OPTIONAL{ ?prop rdfs:range ?range } .
    OPTIONAL{ ?prop rdfs:domain ?domain } .
    OPTIONAL{ ?prop rdfs:label ?label } .
    OPTIONAL{ ?prop rdfs:comment ?comment }
  }
  UNION {
   ?s ?prop ?o .
   FILTER isLiteral(?o)
  }
}
```

Listing 3: Object Properties.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
SELECT DISTINCT ?prop ?domain ?range ?label ?comment
WHERE {
  { ?prop rdf:type owl:ObjectProperty .
    OPTIONAL { ?prop rdfs:range ?range } .
    OPTIONAL { ?prop rdfs:domain ?domain } .
    OPTIONAL { ?prop rdfs:label ?label} .
    OPTIONAL { ?prop rdfs:comment ?comment }
  }
  UNION
  { ?s ?prop ?o .
    FILTER (isIRI(?o) || isBlank(?o))
  }
}
```

As detailed in Section 2.6, the user interface also presents a list of the Resources, Literals and Blank Nodes. Since this information is already available through the underlying Python RDFlib after the pre-filtering query, no further queries are used.

2.6. User Interface

In this section, the user interface of Tarsier is presented. A screenshot of the UI is presented in Figure 5. The top left hand side panel allows the user to load the above mentioned configuration file and shows all the parameters read from it. Among these parameters, it is worth mentioning the colors used to draw meshes in the 3D canvas and the level of detail (LOD) that allows setting the quality of the representation (i.e., to find the best trade-off between resource usage and appearance). On the right hand side, there is the canvas managed by Babylon JS. Below the canvas, a text box shows information about the clicked elements.

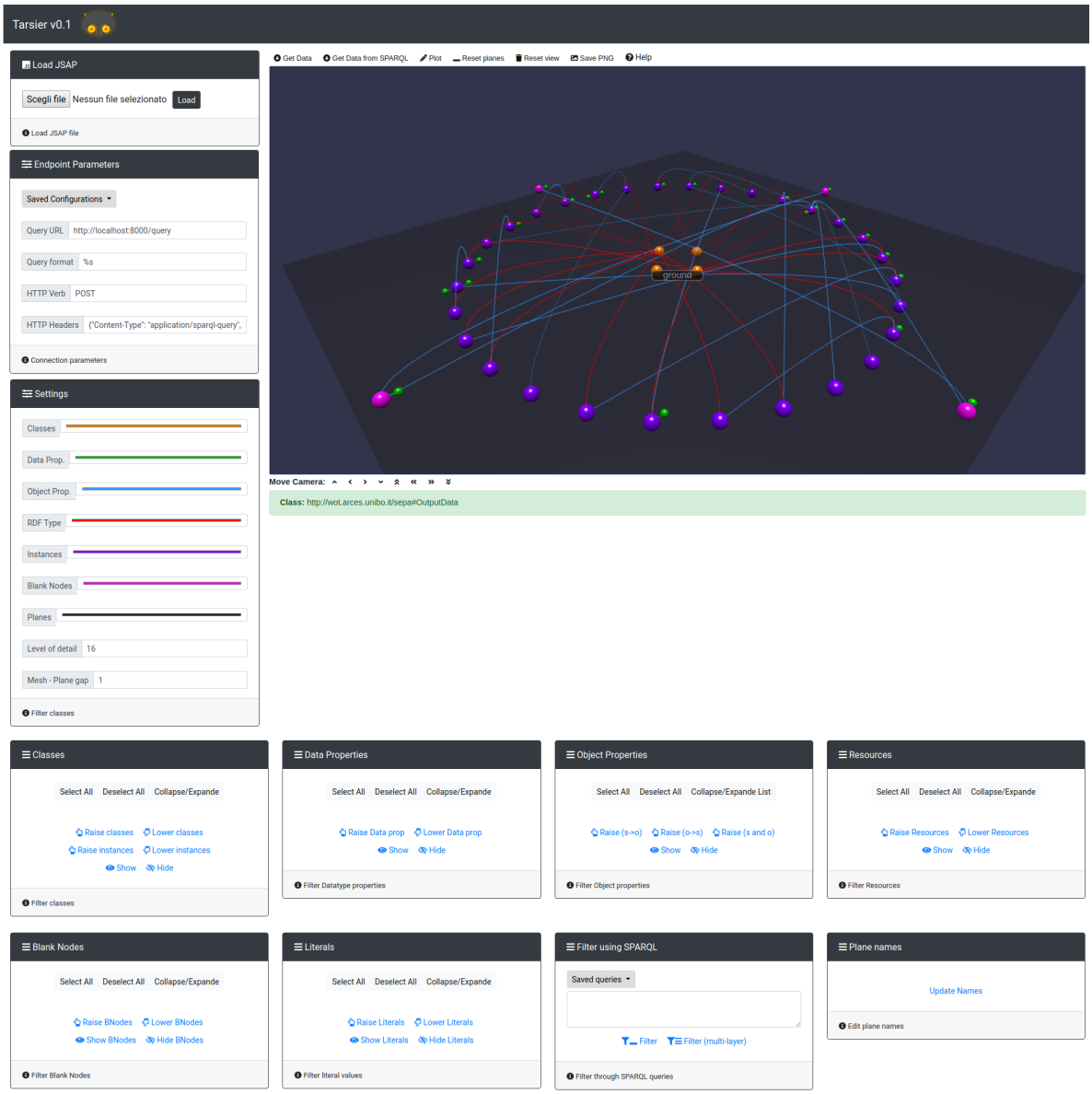


Figure 5. UI of Tarsier.

The bottom part of Figure 5 shows the control panel available to manipulate the view. This panel presents the following eight cards:

- **Classes**: Presents a lists of the classes identified by the data extractor using the query in Listing 1. A checkbox is drawn next to each item to select and deselect the related class. On the selected items, the user may act to toggle visibility or move them across layers. Through this box, it is also possible to show/hide and move resources belonging to the selected classes.
- **Resources**: Contains a list of referents (i.e., IRI resources). As for the classes, the user is allowed to select/deselect items and modify visibility and the layer they belong to.
- **Blank Nodes**: This panel presents the list of blank nodes found in the knowledge base, along with the buttons to change visibility and layer.
- **Object Properties**: This box contains the list of object properties detected by the data extractor (with the SPARQL Query of Listing 3). This box allows to select properties and show/hide or move to other layers the subject and/or of the triples with that predicate.
- **Data Properties**: In this card, all the data properties are shown and the same functionalities of the previous boxes are provided.

- **Literals**: Through this box, it is possible to see all the literals (i.e., values of the datatype properties) found in the knowledge base and move them or toggle their visibility.
- **Filter Using SPARQL**: While the previous boxes allow modifying the view without any knowledge of the SPARQL query language, more complex analysis are possible through this card. The results of the SPARQL query are then shown on a new semantic plane or on a set of semantic planes (i.e., one for each variable in the variable list of the query). The text area in this box is also used to input a SPARQL construct query for the initial extraction of the knowledge base.
- **Plane names**: Moving object from one plane to another causes the creation of semantic planes. The user, who knows the real meaning of a plane, can set and update the name through this box.

3. Use Cases

This Section proposes three use cases to see Tarsier in action. The first use case is a didactic scenario based on the FOAF ontology, often proposed by the authors to student of the course "Interoperability of Embedded Systems" that inspired the tool. In the second one, Tarsier is used to visualize data extracted from DBpedia, while in the third we propose a different use case based on the reification pattern.

3.1. Use Case #1: Teaching through FOAF

From the didactic point of view, Tarsier may help to face the learning curve of Semantic Web technologies. Tarsier allows visualizing an RDF graph, being it an ontology or the content of a store and isolate the concepts of interest, while still maintaining a view to the rest of the data. It is not intended to build or modify RDF stores, but rather to explore and debug. It can then be considered as part of a student or developer toolkit, together with ontology editors, dashboards and APIs.

The default color scheme, also visible in the following examples, is based on the one proposed by the well-known ontology editor Protégé (https://protege.stanford.edu/). Classes are depicted with the orange color, datatype and object properties, respectively, with green and blue, while the color dark purple is used to draw individuals. A little modification consists in the adoption of the color red to mark the property `rdf:type`, which is, in our opinion, very important to quickly identify the relationship between a class and its instances. Lastly, blank nodes are represented with light purple.

Based on the FOAF ontology (http://xmlns.com/foaf/spec/) (one of the first met by students approaching Semantic Web technologies), we propose a simple knowledge base representing people, projects and relations among them. For the sake of clarity, the size of the knowledge base will be kept small. This is not limiting, since the UI proposes intuitive filtering functions to hide unwanted items. Displaying only a small-sized graph, the potentiality of Tarsier may emerge even through static images. Using Tarsier, we want to visually answer to the following questions:

1. Is there a person without friends?
2. Is there any un-assigned project?
3. Do Person1 and Person2 share any projects?

Before answering the questions, it is important to introduce the knowledge base adopted in these examples. To do so, Table 1 briefly summarizes the content of the knowledge base. The graphical representation of the full graph is instead proposed by Figure 6.

Table 1. Use Case #1: Summary of the knowledge base.

OWL Ontology T-Box Content	
Classes (`Person`, `Project` ∈ `foaf`)	2
Object Properties (`knows`, `currentProject` ∈ `foaf`)	2
Datatype Properties (`name`, `surname`, `status` ∈ `foaf`)	2
OWL Ontology A-Box Content	
Persons	25
Projects	5
Links among persons (i.e., `foaf:knows`)	250
Links persons-projects (i.e., `foaf:currentProject`)	125

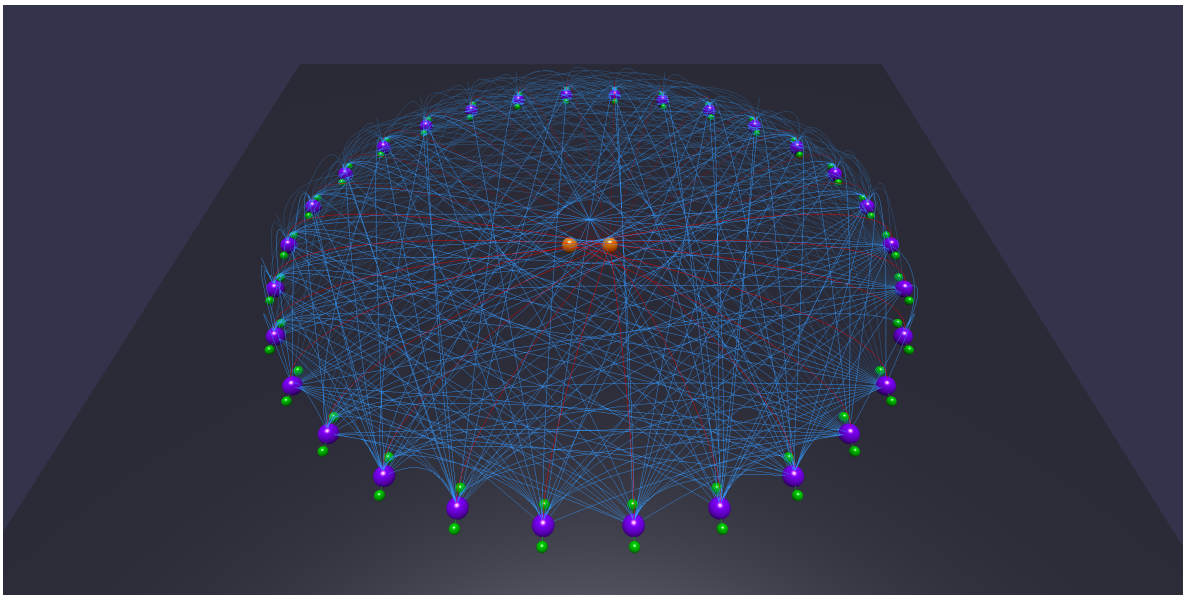


Figure 6. Full knowledge base of the Use Case #1.

Question 1: *Is there a person without friends?* This question can be answered in multiple ways. For example, we may issue a SPARQL query such as the following one:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?p1
WHERE {
  ?p1 rdf:type foaf:Person .
  ?p2 rdf:type foaf:Person .
  FILTER NOT EXISTS { ?p1 foaf:knows ?p2 } .
  FILTER NOT EXISTS { ?p2 foaf:knows ?p1 } .
  FILTER(?p1 != ?p2)
}
```

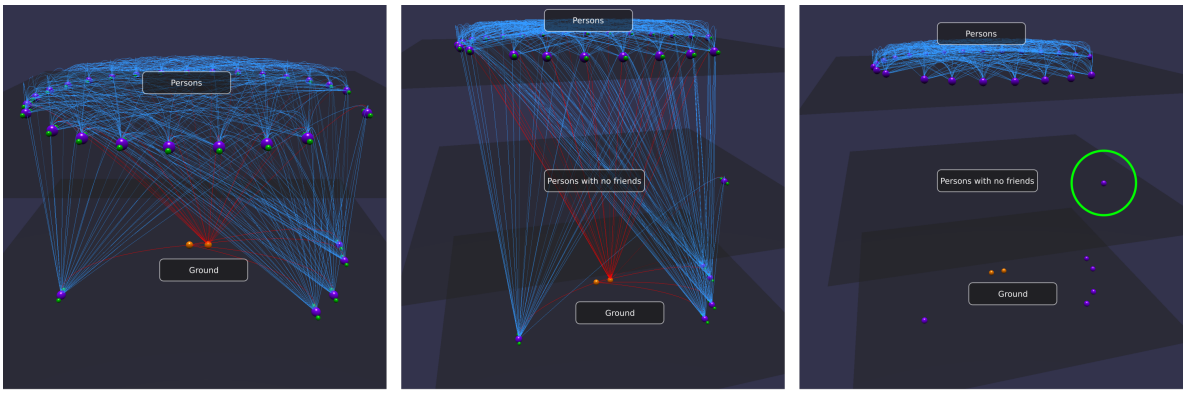
However, Tarsier provides an even simpler way to achieve this scope through the creation of the following semantic planes. In order, the user should:

(a) Create a first semantic plane containing all the instances of the class `foaf:Person`. This causes the instances and their datatype properties to be moved above the rest of the knowledge base.
(b) Create a second semantic plane containing all the instances of the class `foaf:Person` that are involved in a friendship relationship (i.e., being either the subject or the object of a `foaf:knows`

triple). In this way, all persons without friends, if any, remain on the previously created plane. This can be done by selecting the object property `foaf:knows` and clicking on *Raise (S and O)*.

(c) Finally, just to have a better view, it is possible to hide unwanted information (e.g., all the datatype properties and all the object properties except `foaf:knows`).

The previous steps are shown in Figure 7. Through the semantic planes, the existence of an instance of the class `foaf:Person` not linked to the others that stand on the mid-plane renamed as "Persons with no friends" is then immediately noticed. To enhance the readability, datatype properties and object properties other than `foaf:knows` were hidden through the UI commands.



a b c

Figure 7. Use Case #1, Question 1: From the full knowledge base, a new semantic plane containing all the instances of the class `foaf:Person` is created (**a**). Then, all the instances involved in a friendship are moved to a second plane (**b**). All the unnecessary edges are hidden to notice one instance of the class `foaf:Person` with no incoming or outgoing `foaf:knows` edges (**c**).

Question 2: *Is there any unassigned project?* Finding unassigned projects is the second task. One could answer this question through the following query:

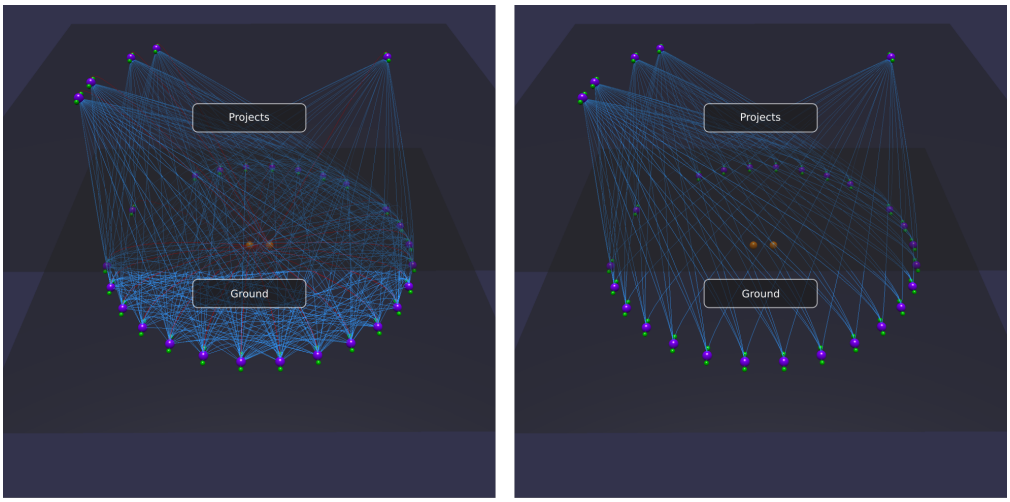
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?p
WHERE {
  ?p rdf:type foaf:Project .
  ?person rdf:type foaf:Person .
  FILTER NOT EXISTS { ?person foaf:currentProject ?p }
}
```

However, Tarsier allows retrieving and showing the same information without having to know the SPARQL query language. Again, the user may draw upon semantic planes by (in order):

(a) Creating a semantic plane containing all the projects (i.e., selecting the class `foaf:Project` and clicking on *Raise instances*); and

(b) Hiding all the data properties and all the arcs related to `foaf:knows` and `rdf:type`.

The result of these actions is shown in Figure 8. The user may immediately notice that all the existing projects are assigned to instances of the class `foaf:Person`.



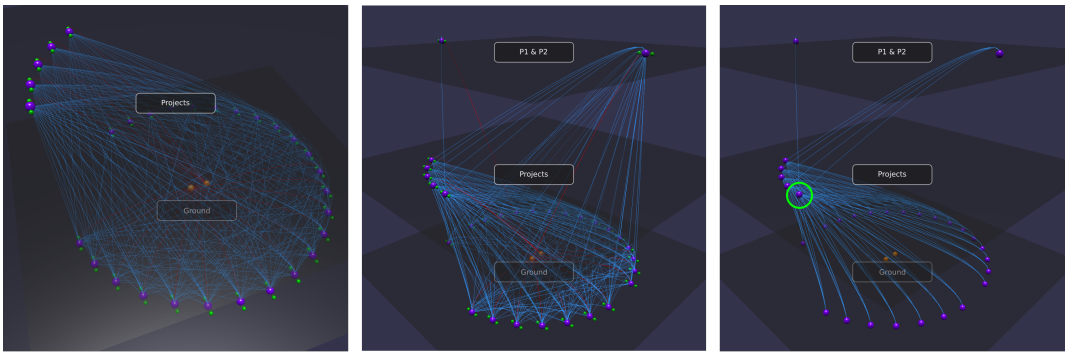
a b

Figure 8. Use Case #1, Question 2: The semantic plane of the projects clearly highlight that all projects are bound to at least one person (**a**). This is more evident when unnecessary edges are hidden (**b**).

Question 3: *Do Person1 and Person2 share any projects?* The third question can, once again, be answered through a SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
ASK {
  foaf:Person1 foaf:currentProject ?p .
  foaf:Person2 foaf:currentProject ?p
}
```

Once again, the user may avoid typing a SPARQL Query creating a semantic plane for the projects (Step a) and a semantic plane hosting only the resources `foaf:Person1` and `foaf:Person2` (Step b). Hiding object properties different from `foaf:currentProject` and all the datatype properties (Step c), it is easy to notice that one of the projects (hosted by the middle semantic plane) presents two incoming edges from the topmost plane (the one related to the selected persons). Thus, it is possible to verify that the previous question has an affirmative answer and, if needed, a click on the project reveals further information. These steps and the results are visualized in Figure 9.



a b c

Figure 9. Use Case #1, Question 3: Do `foaf:Person1` and `foaf:Person2` work on at least one common project? To answer this question we start creating a semantic plane hosting all the projects (**a**). Then, a second plane hosting `foaf:Person1` and `foaf:Person2` is created (**b**). Hiding unnecessary edges, it is easy to identify a project where both work together (**c**).

3.2. Use Case #2: Exploring DBpedia

DBpedia (http://dbpedia.org) is a Public Data Infrastructure for a Large, Multilingual, Semantic Knowledge Graph. As stated in the Introduction, a tool for the visualization of RDF graphs should provide functionalities to declare the portion of the knowledge base that the user intends to inspect. This is particularly true with DBpedia, since visualizing a graph containing 6.6 M entities (as of the last official release dated 2017) can be both heavy and slow to compute and ineffective from the point of view of the results. Thus, the first step when using Tarsier with DBpedia should be the definition of the subgraph of interest through a proper SPARQL CONSTRUCT. In this section, we focus on a specific use case, i.e. retrieving from DBpedia and visualizing the following.

All the artists born in Bologna between 1000 AD and 2000 AD and people who inspired them.

The desired information can be retrieved from DBpedia with a SPARQL query such as the following:

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?artist ?artBirthD ?artDeathD ?artBirthP ?artName ?ins ?insName ?insBirthD
?insDeathD ?insBirthP ?insDeathP
WHERE {
  ?artist rdf:type dbo:Artist ;
          rdf:type foaf:Person ;
          foaf:name ?artName ;
          dbo:birthDate ?artBirthD ;
          dbo:birthPlace :Bologna .
  OPTIONAL {
    ?artist dbo:deathDate ?artDeathD } .
  OPTIONAL {
    ?artist dbo:deathPlace ?artDeathP } .
  OPTIONAL {
    ?artist dbo:influencedBy ?ins .
    ?ins rdf:type foaf:Person ;
         dbo:birthPlace ?insBirthP ;
         dbo:birthDate ?insBirthD .
    OPTIONAL {
      ?ins dbo:deathPlace ?insDeathP ;
           dbo:deathDate ?insDeathD }} .
  FILTER (?artBirthD > "1000-01-01"^^xsd:date).
  FILTER (?artBirthD < "2000-01-01"^^xsd:date)
}
```

From the original SPARQL SELECT query, it is possible to design a specific SPARQL CONSTRUCT query to define a graph with the same meaning:

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>
CONSTRUCT {
  ?artist rdf:type dbo:Artist ;
          rdf:type foaf:Person ;
          foaf:name ?artName ;
          dbo:birthDate ?artBirthD ;
          dbo:birthPlace :Bologna ;
          dbo:deathDate ?artDeathD ;
          dbo:deathPlace ?artDeathP ;
          dbo:influencedBy ?ins .
  ?artDeathP rdf:type dbo:Place .
  :Bologna rdf:type dbo:Place .
  ?ins rdf:type foaf:Person ;
       dbo:birthPlace ?insBirthP ;
       dbo:birthDate ?insBirthD ;
       dbo:deathPlace ?insDeathP ;
       dbo:deathDate ?insDeathD .
  ?insDeathP rdf:type dbo:Place .
  ?insBirthP rdf:type dbo:Place .
}
WHERE {
  ?artist rdf:type dbo:Artist ;
          rdf:type foaf:Person ;
          foaf:name ?artName ;
          dbo:birthDate ?artBirthD ;
          dbo:birthPlace :Bologna .
  OPTIONAL {
    ?artist dbo:deathDate ?artDeathD } .
  OPTIONAL {
    ?artist dbo:deathPlace ?artDeathP } .
  OPTIONAL {
    ?artist dbo:influencedBy ?ins .
    ?ins rdf:type foaf:Person ;
         dbo:birthPlace ?insBirthP ;
         dbo:birthDate ?insBirthD .
    OPTIONAL {
      ?ins dbo:deathPlace ?insDeathP ;
           dbo:deathDate ?insDeathD }} .
  FILTER (?artBirthD > "1000-01-01"^^xsd:date).
  FILTER (?artBirthD < "2000-01-01"^^xsd:date)
}
```

Using Tarsier, the user is then able to explore the subgraph, as shown in Figure 10.

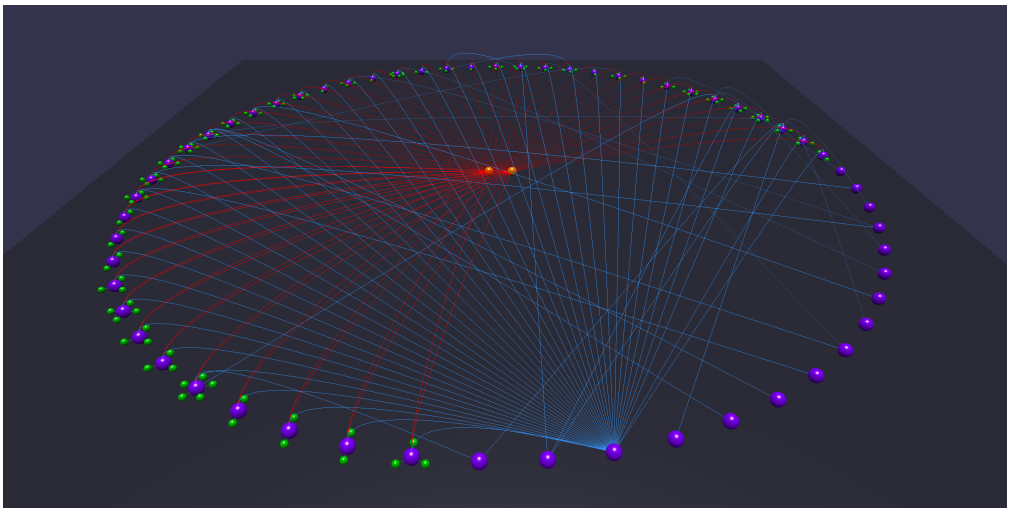
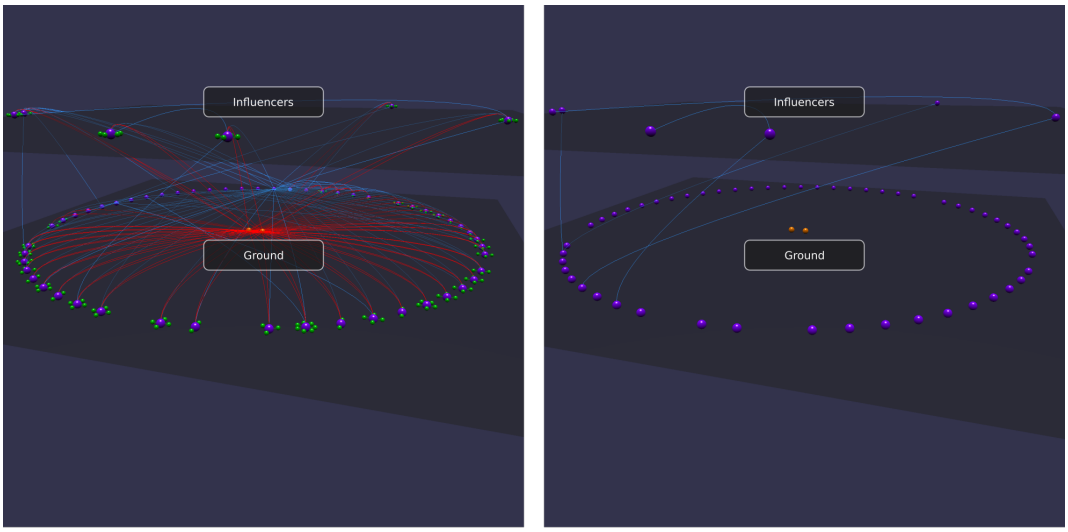


Figure 10. Visualization of the graph extracted from DBpedia (Use Case #2), containing all the artists born in Bologna between 1000 AD and 2000 AD and people who inspired them.

The subgraph extracted with the CONSTRUCT can be browsed by using the mouse and clicking on spheres and edges to see the relative data. However, where Tarsier comes in help is answering questions through the use of semantic planes. Based on the knowledge base just retrieved, we propose three questions:

1. Are there any relations among influencers?
2. Are there any connections between living artists and influencers?
3. Are there any living artists?

Question 1: *Are there any relations among influencers?* Answering this question is easy through the use of semantic planes. In fact, it is sufficient to create a semantic plane hosting influencers (Step a) and hide unwanted information (Step b), to notice the presence of two links among influencers (Figure 11). Clicking on these links, it is possible to discover that Carlo Cignani was influenced by Francesco Albani and Ludovico Carracci was influenced by Annibale Carracci. Note that, in Figure 11, all other object properties and all data properties were hidden through the proper controls.



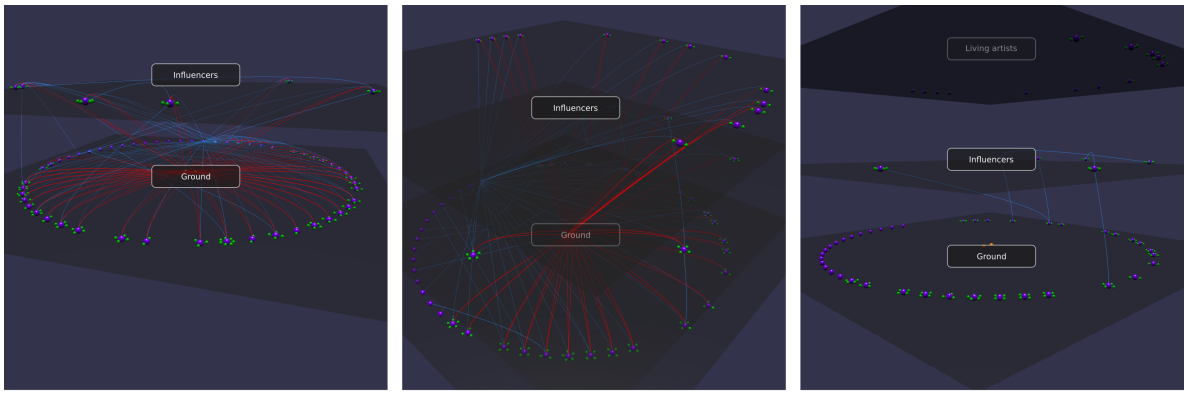
a b

Figure 11. Use Case #2, Question 1: A semantic plane showing the influencers, placed above the semantic plane with the rest of the KB (**a**). Hiding unnecessary edges produces a more effective visualization (**b**).

Question 2: *Are there any connections between living artists and influencers?* An answer to this question can be easily found with two iterative steps that corresponds to the creation of two semantic planes: the first dedicated to influencers, and the second to living artists. This second plane can be created with a simple SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?art
WHERE {
  ?art rdf:type dbo:Artist .
  FILTER NOT EXISTS { ?art dbo:deathPlace ?dp }.
  FILTER NOT EXISTS { ?art dbo:deathDate ?dd }
}
```

A third optional step is to hide unwanted information. These steps and the resulting visualization are depicted in Figure 12. This picture clearly highlights the absence of connections between the two new semantic planes. Thus, at least according to DBpedia, none of the living artists born in Bologna is influenced by another artist.

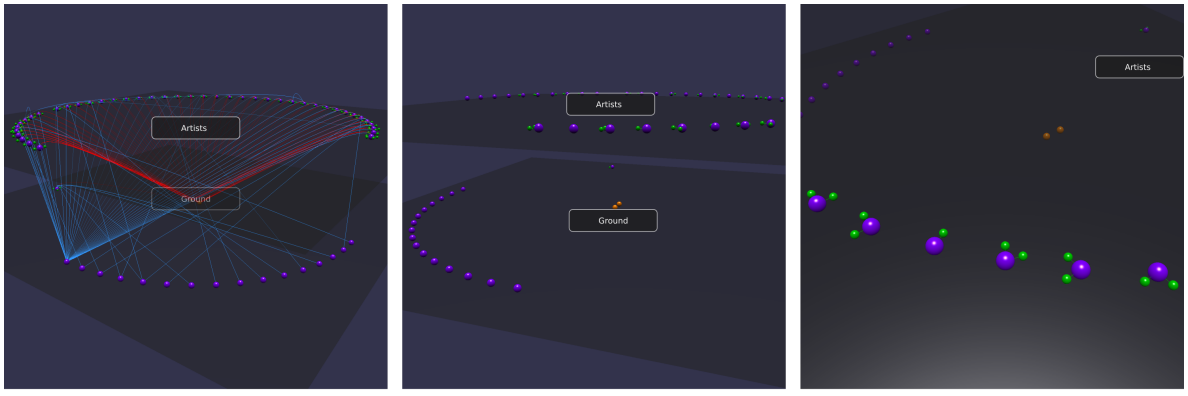


a b c

Figure 12. Use Case #2, Question 2: A semantic plane hosting influencers is created on top of the ground plane (**a**). A second semantic plane containing the living artists stands above the semantic plane of the influencers (**b**). No links between these two planes exists (**c**). At the bottom, the rest of the knowledge base.

Question 3: *Are there any living artists?* To answer this question, it is sufficient to move all the artists to a dedicated semantic plane and hide all the data properties except `dbo:deathDate`. These two steps are depicted in Figure 13a,b. All the resources not connected to a green sphere are living artists. The task is then achieved. Figure 13c shows a close-up on the second plane where resources without visible data properties can be considered as living artists. This closer look helps in noticing something strange in the knowledge base. Through Tarsier, in fact, it is easy to notice that many resources have more than one visible green ball, i.e., they died more than once. This is the case, for example, of Alessandro Tiarini, for which the death date is reported as "`1668-02-08`" and "`"1668-2-8`" (clearly the same date with different formatting) or the case of Domenichino which instead has two death dates that differ not only for the formatting, but also for the value of the year ("`1641-4-6`" and "`1648-04-06`").

Therefore, Tarsier allowed identifying the wrong use of a functional property.



a b c

Figure 13. Use Case #2, Question 3: A semantic plane hosting artists is created on top of the ground plane (**a**). Unnecessary edges are hidden and the remaining green spheres refer to the `dbo:deathDate` property (**b**). Having more than one of these spheres means that the related artist has more than one death date (**c**).

3.3. Reificated KBs

The use case presented in this section still involves FOAF with persons and projects. The knowledge base adopted in this use case is a very simple one showing relationships among people and projects. Organizations (yet a concept borrowed from FOAF) confirms the relationships

by also adding a start date and the envisioned end date (if any). The confirmation of a relationship between a person and a project, in this sample knowledge base, is expressed through the reification pattern. In this specific case, the standard reification [12] (involving then the `rdf:Statement` class) will be take as a reference point. A summary of the knowledge base is proposed in Table 2.

Table 2. Use Case #3: Summary of the Knowledge Base.

OWL Ontology T-Box Content	
Classes (`Person`, `Project`, `Organization` ∈ `foaf`, `Statement` ∈ `rdf`)	4
Object Properties (`currentProject` ∈ `foaf`, `ackBy`, `startDate`, `endDate` ∈ `ns`, `subject`, `predicate`, `object` ∈ `rdf`)	6
Datatype Properties (`name`, `surname`, `status` ∈ `foaf`, `object` ∈ `rdf`)	5
OWL Ontology A-Box Content	
Persons	20
Projects	5
Organizations	2
Links persons-projects (i.e., `foaf:currentProject`)	20
Links organizations-persons (i.e., `foaf:member`)	20
Acknowledged statements (i.e., `ns:ackBy`)	20
Statements time-stamped with `ns:startDate`	20
Statements time-stamped with `ns:endDate`	10

Before presenting the example in detail, a brief introduction to standard reification is proposed. Through RDF, information can be easily represented as a set of triples (subject, predicate and object). Here is an example:

```
foaf:Person1 foaf:currentProject foaf:ProjA
```

Sometimes, this it is not enough and there is the need to state something more about a given triple. This can be achieved through the reification pattern, where a triple $t = (s, p, o)$ is broke down into four triples. The first is used to declare a statement, and the others to express its components. Thus, the previous triple can be decomposed as follows:

```
ns:St1 rdf:type rdf:Statement
ns:St1 rdf:subject foaf:Person1
ns:St1 rdf:predicate foaf:currentProject
ns:St1 rdf:object foaf:ProjA
```

(where `ns` is a custom namespace). In this way, information related to the previous triple can be expressed by simply referring to the statement as follows:

```
ns:St1 ns:ackBy foaf:Organization1
ns:St1 ns:startDate ‘‘...’’
ns:St1 ns:endDate ‘‘...’’
```

The unfiltered content of the knowledge base is presented in Figure 14, while Figure 15 shows one of the present statements on a dedicated plane. This visualization allows viewing the subject, predicate and object composing the triple (again, a click on these items allows seeing all the related information). In the described scenario, one instance of the class rdf:Statement is used to link a person to its current project. Organizations may acknowledge the triple and append information to each statement as the start and end date of the collaboration. Link outgoing from the statement (i.e., the pink sphere, since in our case is a blank node) represent the information appended to it by the Organization: in this case, only a data property is bound to the statement, so no end date is envisioned for the collaboration of

the person with that project. The presence of the underlying semantic plane (i.e., the ground) name allows maintaining a view on the rest of the knowledge base, even when looking at a single statement. It is then possible to notice, in the specific example, that the predicate is linked to the ground by a high number of links, so many other statements may have this predicate; the project only has three links with the ground (a click on them reveals that are links of type `foaf:currentProject`).

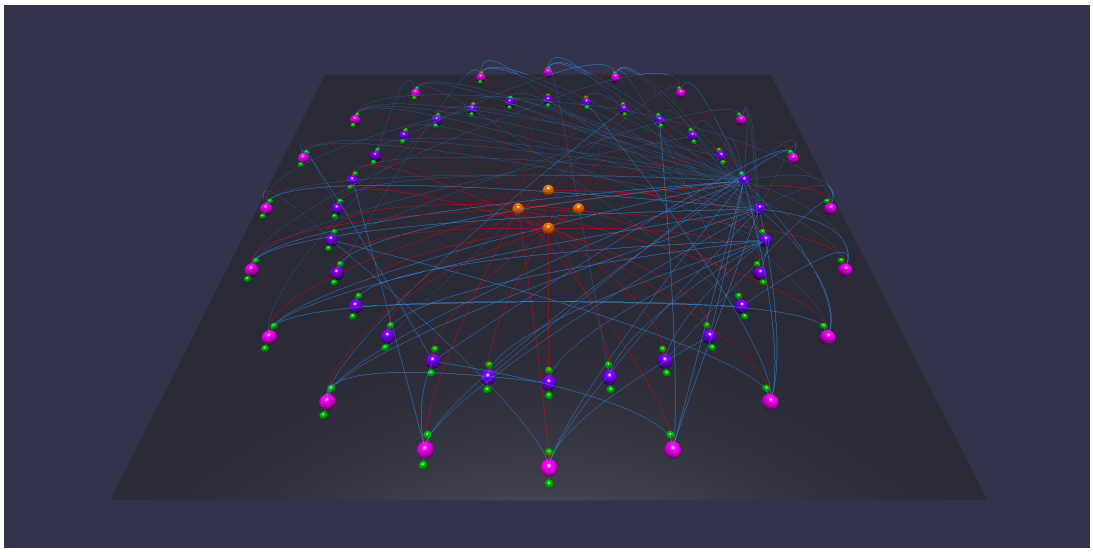


Figure 14. Use Case #3: The unfiltered knowledge base of the reification use case.

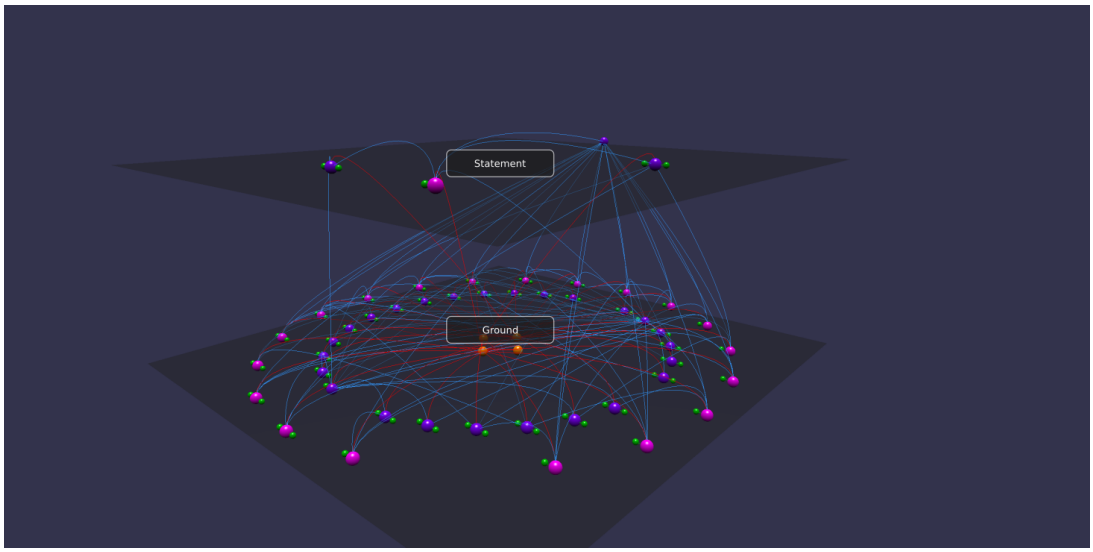


Figure 15. Use Case #3: A statement has been moved to a dedicated plane.

In Figure 16, another example based on the same knowledge base is proposed. All the statements have been moved to a proper semantic plane (Step a), while one of the organizations (i.e., instances of `foaf:Organization`) was moved to the topmost plane (Step b). Then, all the `rdf:type` edges have been hidden for the sake of readability. This allows visually identifying the relationship among the selected organization and all the instances of `rdf:Statement`. It is then easy to notice how this organization took the burden of acknowledging all of the present statements.

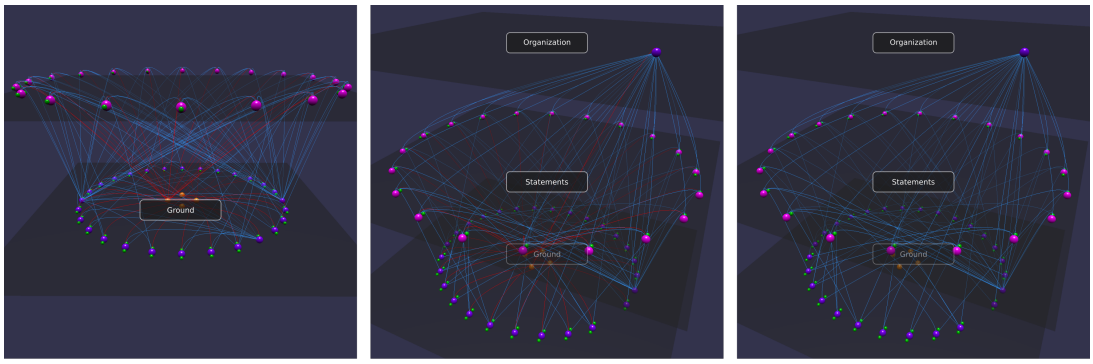


Figure 16. Use Case #3: A first semantic plane hosts instances of the class `rdf:Statement` (**a**). A multi-planar view with a topmost semantic plane dedicated to `foaf:Organization1` shows the link among the organization and the statements (**b**). Then, unnecessary edges are hidden (**c**).

4. Evaluation

In this section, the preliminary results of the evaluation of the user experience are presented (Section 4.1), followed by a computational assessment of the performance of the tool (Section 4.2).

4.1. User Evaluation

Since Tarsier was mainly born as a tool to support students dealing with Semantic Web technologies, an evaluation of the user experience is important to assess the validity of the approach and identify possible improvements. Sixteen participants were selected among the students attending the course "*Interoperability of Embedded Systems*" held at the Computer Engineering faculty of the University of Bologna. Six of them have had previous minor experience on Semantic technologies. For the others, this university course has been the first contact with RDF knowledge bases. User evaluation methods based on a set of tasks to assess efficiency and effectiveness were employed. The evaluation was preceded by a short presentation of the tool (10 min) where the basics of the UI and the aim of the tool were discussed. Then, the experimenters were free to explore the tool and perform a feature walkthrough (10 min). Through the concurrent think-aloud protocol (CTA), we gathered the insights of users' cognitive processes while both free experimenting and performing five assigned tasks characterized by increasing complexity. In the first three tests, they were asked to interact with a local SPARQL Endpoint based on SEPA [13], the same adopted by students for their final assessment project. The remaining tests were about the visualization of data contained in DBpedia. Participants were free to allocate the desired amount of time to carry out the assigned tasks for a total time of one hour.

At the end of the test, students completed a survey with two set of questions: the first task-specific, intended to understand how the user carried out each task, while the second aimed at the overall evaluation of the tool. Students were also allowed to write a short sentence after each question to express their opinion and highlight strong and weak points of the tool and its approach based on semantic planes. The test was intended to assess the level of usability and the learning curve, the overall feedback and the perceived level of utility and novelty. This was achieved through ten five-point Likert items (selected scale: *Strongly disagree*, *Disagree*, *Neutral*, *Agree*, and *Strongly Agree*).

The results of the final users questionnaires are shown in Figure 17. The preliminary results suggest that the tool is useful for newcomers, as well as effective to understand data. The idea of semantic planes and the filtering mechanism were judged positively by all the participants. Among the suggestions received from the students, the most common was related to "adding more visual tips and feedbacks". This request was promptly accepted and new messages to confirm the action performed by the user were added. Moreover, a help screen was added to further guide a new user to the tool and three introductory

videoclips (available on the GitHub page: https://github.com/desmovalvo/tarsier) showcase it through three examples based on the three use cases proposed in this paper.

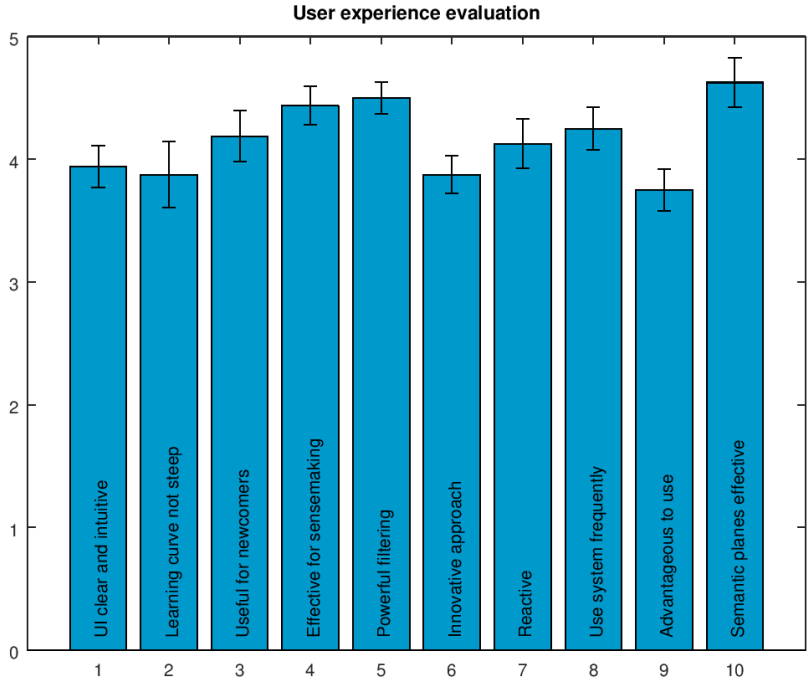


Figure 17. Mean and standard error of the mean (SEM) of the results of the questionnaire items.

4.2. Performance Evaluation

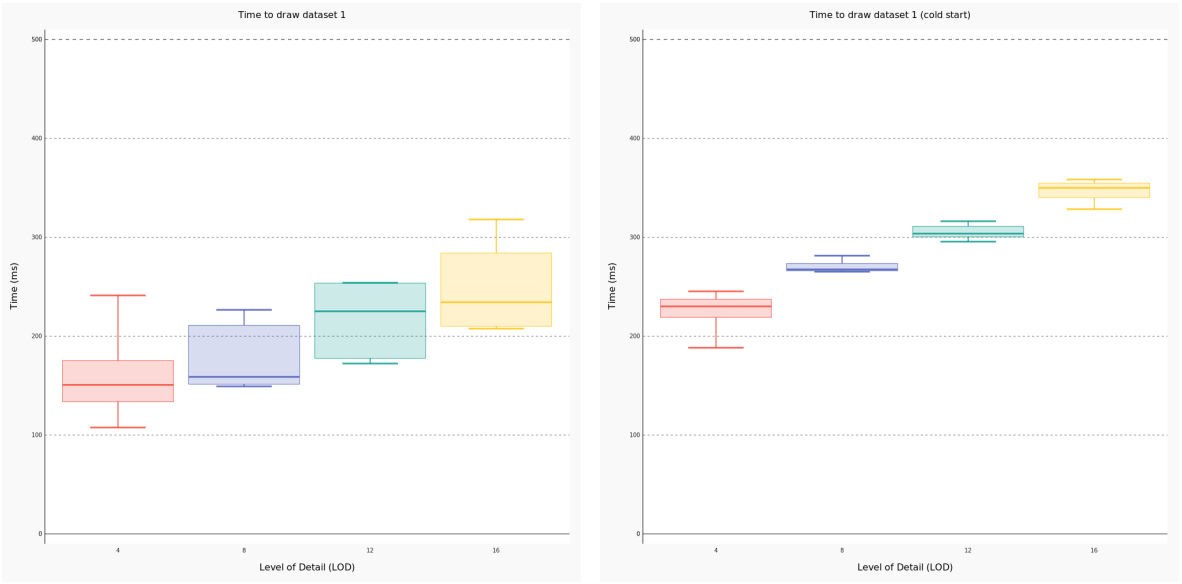
Even though performance is not a primary aspect of inspection tools including Tarsier, this section presents the results of preliminary evaluation tests. Intuitively, the time needed to draw the graph depends on three main factors:

- the amount of data to be traced (i.e., the number of meshes);
- the requested level of detail that is one of the parameters configurable by the user; and
- wether the 3D scene has been initialized (i.e., the latter condition is defined *cold start*).

A generic dataset proposed with five different sizes was utilized. Datasets are identified by labels DS#i with $i = 1, \ldots, 5$. Dataset DS#i contains $200 \cdot i$ triples, and the full representation requires drawing $200 \cdot i + 1$ spherical meshes and $200 \cdot i$ bezier curves (adopted for datatype and object properties). For the purpose of this test, the specific content of the knowledge base does not influence the evaluation.

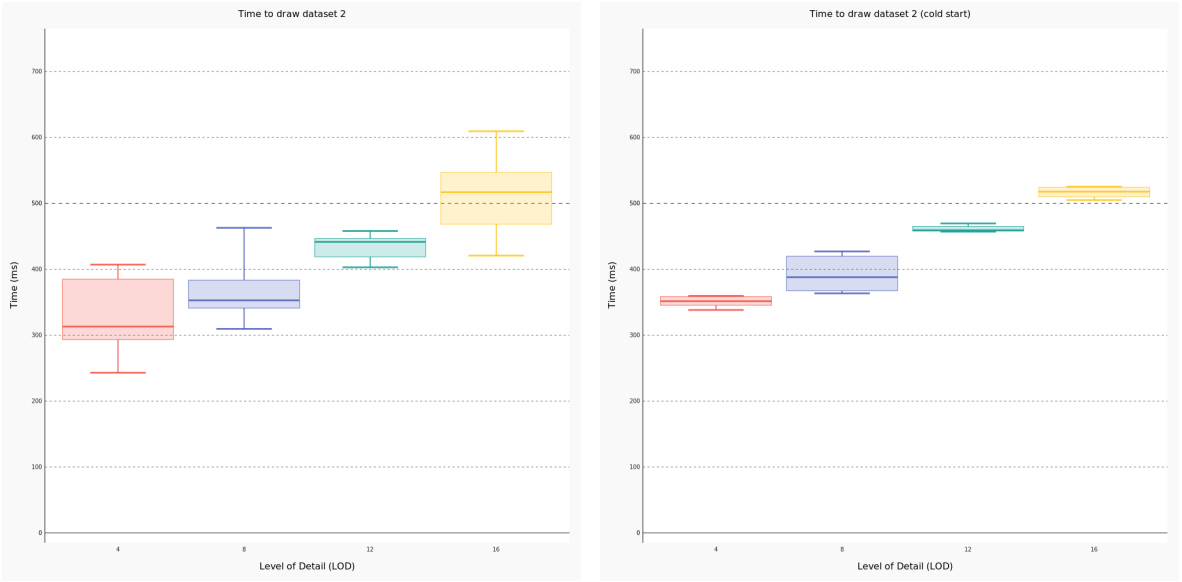
Every dataset was tested with both a cold start condition and with an already initialized scene. Four LOD values were tested (4, 8, 12 and 16). All tests were performed on a Dell Alienware with 8-core Intel(R) Core(TM) i7-4720HQ CPU at 2.60GHz and 8 GB RAM. Both server and client were running on the same machine using Google Chrome 64.0.

Figures 18a, 19a, 20a, 21a and 22a report the results of cold start test, while Figures 18b, 19b, 20b, 21b and 22b report the results of tests executed on an instance of Tarsier where the scene was already initialized. The charts confirm the expected behavior of the application: time needed to draw the graph grows with the size of data and with the requested level of detail. Furthermore, a cold start of Tarsier (i.e., where the scene is not yet initialized) requires a higher number of milliseconds to complete the drawing. Lastly, Figure 23 compares the time needed to identify the role of each RDF term in the knowledge base for the five different datasets.



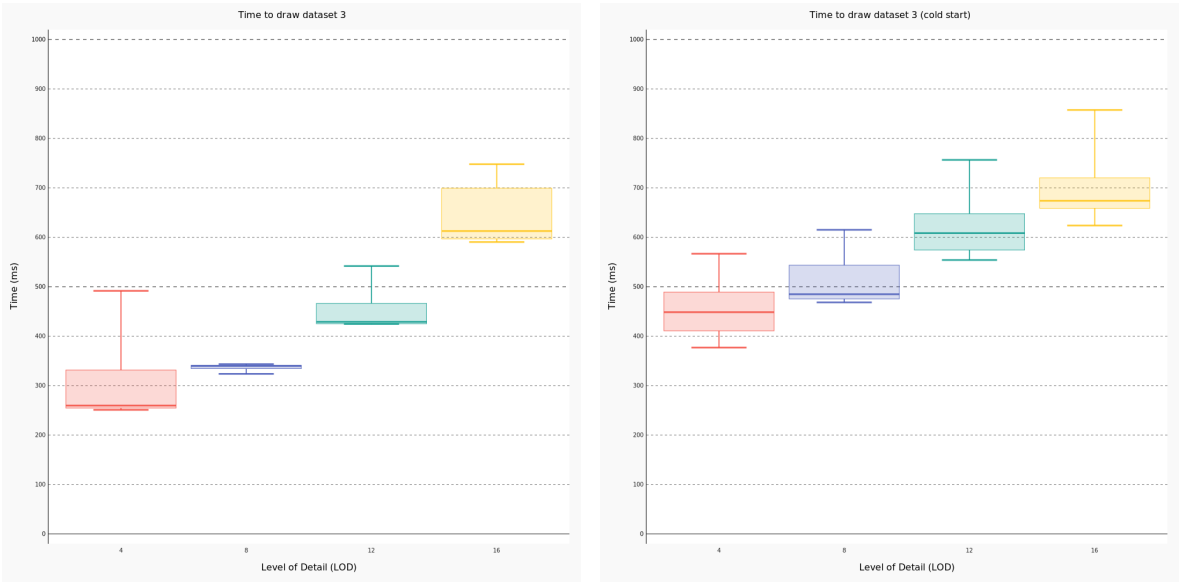
(**a**) DS#1 (Cold Start) (**b**) DS#1

Figure 18. Time to represent DS#1.



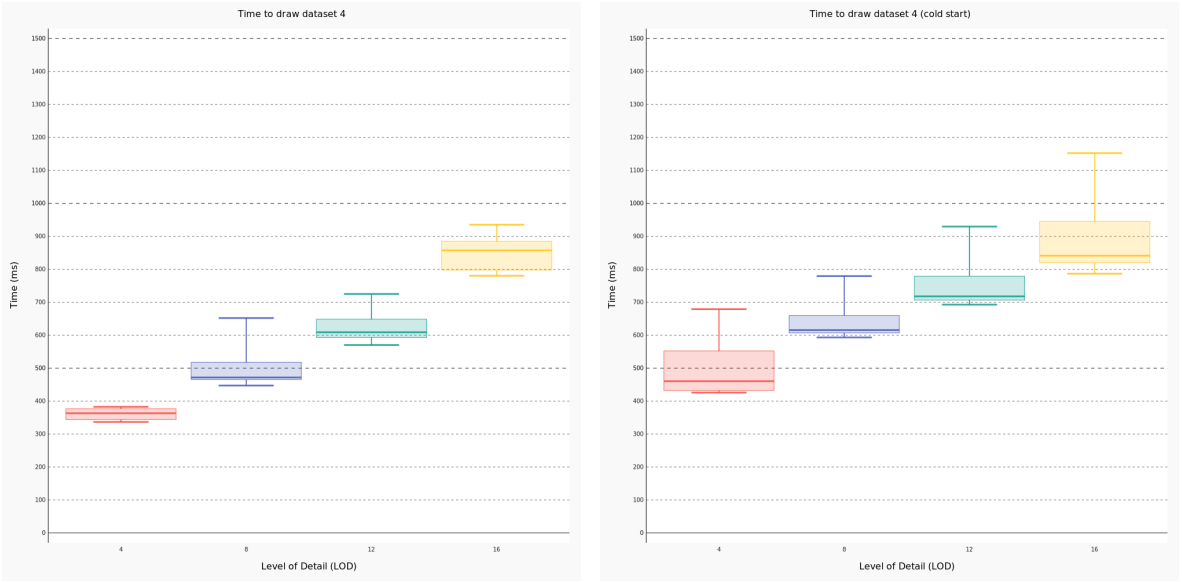
(**a**) DS#2 (Cold Start) (**b**) DS#2

Figure 19. Time to represent DS#2.



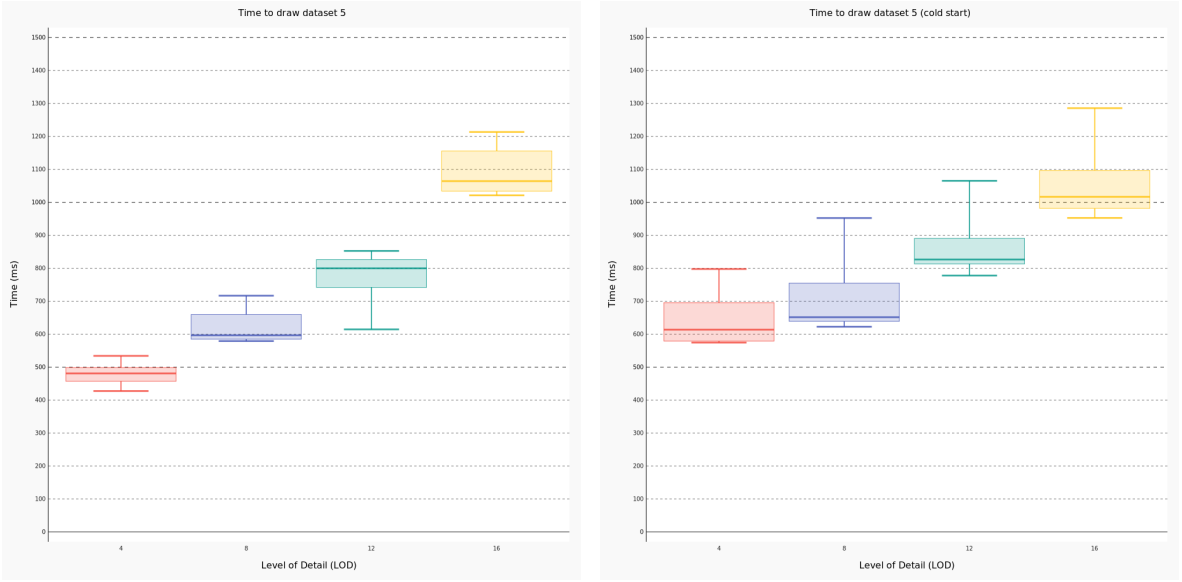
(**a**) DS#3 (Cold Start) (**b**) DS#3

Figure 20. Time to represent DS#3.



(**a**) DS#4 (Cold Start) (**b**) DS#4

Figure 21. Time to represent DS#4.



(**a**) DS#5 (Cold Start) (**b**) DS#5

Figure 22. Time to represent DS#5.

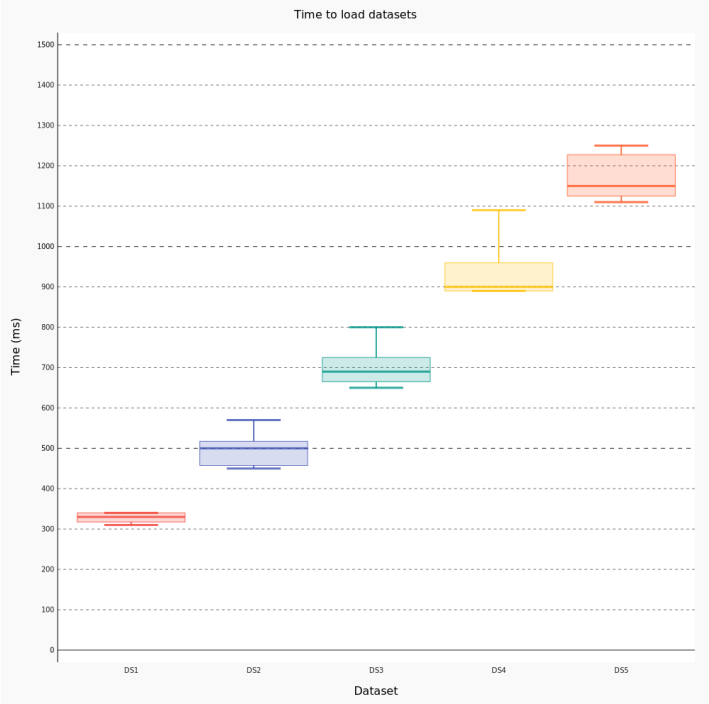


Figure 23. Time employed by the data extractor to analyze data depending on the dataset.

5. Related Work

Graph theory [14] and the use of graph data structures have found their application in several domains, from chemistry and biology, to social sciences, networks and, of course, the Semantic Web [1]. In particular, ontologies and Linked Data take the form of oriented and labeled graphs and so the tools for their interactive visualization could be classified as: (1) tools for the visualization and exploration of all kinds of graphs with the support of plugins/extensions to import semantic data; and (2) tools

specifically designed for the Semantic Web. In general, the former offer sophisticated analysis and exploration features for graphs expert users, while the latter implement a reduced set of functions but are more suitable for Semantic Web users. While in the rest of this section we focus on the use of graphs, we would like to mention that this is not the only approach to RDF data visualization, as demonstrated by Gallego et al. [15] who proposed for example a method to visualize RDF data based on a 3D adjacency matrix. In the literature, many algorithms to layout graphs and interact with them have been proposed. Among all the algorithms we have been able to discover, here we provide the ones that, from our point of view, are more related to our scenario. Gansner et al. [16] proposed a method for drawing directed graphs, and later on Gansner and North [17] presented a graph visualization software along its application in several fields. In 2007 Gansner and Koren [18] focused on algorithms for positioning nodes and routing edges to maximize the readability of circular layouts. An algorithm for drawing labeled nodes removing overlapping nodes and minimizing, at the same time, the drawing area is presented by Gansner and Hu [19]. Binucci et al. [20] focused on the problem of drawing arrows in directed graphs. Some problems related to 3D representation are presented by Brandenburg et al. [21]. An algorithm and the related tool for grouping nodes in not overlapping regions based on node attributes that allow a user to interactively filter the results is presented by Shneiderman and Aris [22]. The main algorithms presented by Gansner et al. [23] are implemented by the GraphViz open source graph visualization software [24], while, focusing on object relationships, Gansner et al. [25] presented a tool for visualizing relational data with geographic-like maps. Stolper et al. [26] presented a new idea of graph-level operations to provide a new model of graph exploration along with the potential of discovering new network visualization techniques. Wu et al. [27] made an interesting list of considerations about the methods used on average to build visualization systems. For instance, the limited flexibility of some tools is highlighted, regarding their specific context of use, or their lack of extensibility towards interactive use. A novel language is presented, called DeVIL, that is able to correlate user interactions with the database views in a variety of ways. The DeVIL program is then translated into a workflow that creates the interface and listens to user's direct and indirect requests. The main difference between those workflows and Tarsier is that in the cited work the matter of discussion is how to visualize predefined views of a database in an effective and if possible interactive way. That is, DeVIL requires users to know what is in the database. Tarsier, on the other side, aims to allow exploration of the data stored and to offer a standard way to interact with every semantic graph.

From our point of view and with respect to our aims, Gephi [28] represents the most interesting tool in the graphs visualization and interaction category. The support to RDF can be provided by two plugins: VirtuosoImporter (https://marketplace.gephi.org/plugin/virtuoso-importer/) and SemanticWebImport (https://marketplace.gephi.org/plugin/semanticwebimport/). The latter offers also the possibility of simple filtering using SPARQL. Another tool for network data integration, analysis and visualization is Cytoscape (http://www.cytoscape.org/). In this case, the RDF support is implemented by a set of extensions such as General SPARQL (http://apps.cytoscape.org/apps/generalsparql) or OWLPlugin (http://apps.cytoscape.org/apps/owlplugin). We were not able to load the datasets we would like to test but only the ones conforming with the BioPAX format (http://www.biopax.org/).

Moving to the research works more related to ontologies and Linked Data visualization, among all the papers we have been able to go through, citetSayers2004 first attempted to visualize and interact with an RDF graph using a web browser: the aim was to visualize only a portion of interest through a research over literals. An incremental graph navigation method is presented by Dokulil and Katreniakov [30] that proposed what they call "node merging" as a way to include into a vertex, not only its label, but also a list of incoming and outgoing edges. In addition, Deligiannidis et al. [31] proposed a tool implementing an incremental algorithm to layout the explored sub-graphs. Focusing on not ontology-expert users, Lohmann et al. [32] presented a visual language based on a set of graphical primitives and a color scheme (VOWL 2). Instances are not considered in this tool, although their

number can be shown for the selected class. An example of a tool implementing VOWL 2 is provided by WebOWL (http://vowl.visualdataweb.org/webvowl.html). Another tool in this context is tFacet (http://www.visualdataweb.org/tfacet.php) that has been presented by Brunk and Heim [33] and is based on the concept of faceted exploration [34]. Facets can be extracted through SPARQL queries to SPARQL endpoints and, also in this case the exploration starts with the limitation of the initial search space to avoid displaying very large amount of data. Focusing on relationships discovery between two or more nodes, Heim et al. [35] presented RelFinder (http://www.visualdataweb.org/relfinder.php), a tool that implements a process consisting in four steps: object mapping, relationship search, visualization, and interactive exploration. A last example of a tool designed for not expert users is presented by Heim and Lohmann [36]. This tool aims at combining scatter plots (i.e., to support the visual identification of linear correlations, clusters, patterns, and extreme values) with the interaction metaphor of magic lenses [37]. For a general discussion on a model designed to describe and thus better understand the human–computer interaction in the Semantic Web, we invite the reader to refer to the paper by Heim et al. [38]. To the best of our knowledge, Dadzie and Pietriga [39] and Nuzzolese et al. [40] published two of the most recent papers on semantic data visualization. In the former, the authors investigated on different approaches for exploratory discovery and analysis of Linked Data provided by a set of tools, while the latter authors focused on presenting a tool based on a novel approach to Linked Data exploration named Encyclopedic Knowledge Patterns (EKPs).

We conclude this section by presenting a set of tools that we think can be representative of the ones specifically designed for ontology and Linked Data visualization. The first is the query service offered by WikiData (https://query.wikidata.org/): the user can formulate a SPARQL query and the results are shown in many different ways, including a graph. Using the graph representation, the user can navigate the graph by clicking on each node. Every click opens the direct relationships with other nodes (i.e., first by presenting the number of resources). As stated at the beginning of this section, the WikiData query service is an example of the different ways of presenting the results of a SPARQL query (e.g., charts, tables, map, timeline, etc.). Another tool is RDF Gravity (http://semweb.salzburgresearch.at/apps/rdf-gravity/). In a 2011 work, Bremer et al. (http://www.ebremer.com/nexus/2011-05-15) investigated whether it would be possible to use semantics to enhance research collaborations within scientists sharing interests in their field. Their software, which the authors were unable to find and try, however, seems achievable with a wise use of Tarsier and might take advantage of the concept of semantic plane to get better visualization of the knowledge retrieved.

LOD Live (http://en.lodlive.it/) offers a service to browse RDF resources through the interaction with SPARQL endpoints (e.g., DBpedia). One can incrementally navigate the Linked Data starting from a selected resource. Around a resource, a set of symbols represents different kinds of relationship, such as direct relations, group of direct relations, inverse relations and group of inverse relations. A specific symbol is assigned to the `owl:sameAs` property (i.e., a very relevant aspect when dealing with heterogeneous data sources such as in the Linked Data domain). OWL visualization is also crucial with reference to ontology design. One of the most used tool in this sense is Protégé (http://protege.stanford.edu/). The Protégé ontology editor can be extended with plugins to visualize the ontology as a graph [41,42]. This feature can be provided by plugins such as OntoGraf (http://protegewiki.stanford.edu/wiki/OntoGraf), OWLViz (http://protegewiki.stanford.edu/wiki/OWLViz) and Jambalaya (http://protegewiki.stanford.edu/wiki/Jambalaya) (not supported by the last stable version of Protégé). Ontograf allows to select a starting point in the ontology and then expand it to show subclass relations, instances, datatype and object properties. Different automated positioning methods are provided, but no repositioning and filtering mechanisms (e.g., based on SPARQL). OWLViz visualizes only classes and "is-a" relationships among them. Jambalaya [43] is designed to show classes and instances, as well as relationships. Jambalaya also provides support for grouping nodes based on relationships (e.g., subclasses are represented as nested node of the superclass). Lomov and Shishaev [44] proposed a novel approach to the visualization of ontologies called cognitive frames. Through cognitive frames, the knowledge of a target concept related to the visualized fragment of ontology can be conveyed to

the user. This approach is more focused on terminological data, rather than assertional data which are the main focus of our work.

6. Conclusions

This paper presents Tarsier, an interactive tool for the visualization of RDF knowledge bases with support for RDFS and OWL and its metaphore of Semantic planes that allows grouping all the RDF terms sharing common concepts. Semantic planes can be created, modified and split through a set of UI controls or through SPARQL 1.1 queries. The purpose of the presented tool is mainly to support: (1) students learning how to deal with Semantic Web data representation formats; and (2) software developers during the debugging of applications with RDF stores. In fact, through the pre-filtering phase, it is possible to extract a subgraph from a very large dataset. Preliminary user evaluation tests suggest that the three-dimensional visualization of the small- and medium-sized knowledge bases, combined with the approach of semantic planes and a powerful filtering mechanism, is useful for newcomers to understand the nature of data and its structure. Future, relevant enhancements of Tarsier will consist in: (1) support for the whole set of SPARQL constructs (e.g., to support aggregation functions); (2) alternative arrangement methods for meshes in the 3D space; and (3) support for real-time visualization of data through semantic event processing architectures (e.g., [13,45,46]). This would help to monitor the evolution of RDF resources through specific SPARQL subscriptions.

Author Contributions: Conceptualization, F.V., A.D.; Investigation, F.V., L.R. and F.A.; Funding Acquisition, T.S.C.; Resources, T.S.C.; Software, F.V.; Visualization, F.V.; Writing-Original Draft, F.V., L.R., F.A. and A.D.; Writing-Review & Editing, F.V., L.R., F.A. and C.A.

Funding: The work presented in this paper is funded by the Advanced Research Center on Electronic Systems "Ercole De Castro" (ARCES), University of Bologna, 40125 Bologna, Italy.

Acknowledgments: The work presented in this paper is being developed at the Advanced Research Center on Electronic Systems "Ercole De Castro" (ARCES), University of Bologna, 40125 Bologna, Italy.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Berners-Lee, T.; Hendler, J.; Lassila, O. The semantic web. *Sci. Am.* **2001**, *284*, 28–37. [CrossRef]
2. Gruber, T.R. Ontology. In *Encyclopedia of Database Systems*; Liu, L., Özsu, M.T., Eds.; Springer: Berlin/Heidelberg, Germany, 2009.
3. Gyrard, A.; Zimmermann, A.; Sheth, A. Building IoT based applications for Smart Cities: How can ontology catalogs help? *IEEE Internet Things J.* **2018**, 1. [CrossRef]
4. Vandenbussche, P.Y.; Atemezing, G.A.; Poveda-Villalón, M.; Vatant, B. Linked Open Vocabularies (LOV): A gateway to reusable semantic vocabularies on the Web. *Semant. Web* **2017**, *8*, 437–452. [CrossRef]
5. Chebotko, A.; Lu, S.; Jamil, H.M.; Fotouhi, F. *Semantics Preserving SPARQL-to-SQL Query Translation for Optional Graph Patterns*; Tech. Rep. TR-DB-052006-CLJF; Wayne State University: Detroit, MI, USA, 2006.
6. Zhao, L.; Ichise, R. Ontology integration for linked data. *J. Data Semant.* **2014**, *3*, 237–254. [CrossRef]
7. Asin, A.; Gascon, D. 50 sensor applications for a smarter world. In *Libelium Comunicaciones Distribuidas*; Tech. Rep.: Zaragoza, Spain, 2012.
8. Nguyen, V.; Bodenreider, O.; Sheth, A. Don't like RDF reification?: Making statements about statements using singleton property. In Proceedings of the ACM 23rd International Conference on World Wide Web, Seoul, Korea, 7–11 April 2014; pp. 759–770.
9. D'Elia, A.; Perilli, L.; Viola, F.; Roffia, L.; Antoniazzi, F.; Canegallo, R.; Salmon Cinotti, T. A self-powered WSAN for energy efficient heat distribution. In Proceedings of the 2016 IEEE Sensors Applications Symposium (SAS), Catania, Italy, 20–22 April 2016; pp. 1–6.
10. D'Elia, A.; Viola, F.; Montori, F.; Felice, M.D.; Bedogni, L.; Bononi, L.; Borghetti, A.; Azzoni, P.; Bellavista, P.; Tarchi, D.; et al. Impact of Interdisciplinary Research on Planning, Running, and Managing Electromobility as a Smart Grid Extension. *IEEE Access* **2015**, *3*, 2281–2305. [CrossRef]

11. Motta, E.; Mulholland, P.; Peroni, S.; d'Aquin, M.; Gomez-Perez, J.M.; Mendez, V.; Zablith, F. A novel approach to visualizing and navigating ontologies. In *International Semantic Web Conference*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 470–486.
12. Hernández, D.; Hogan, A.; Krötzsch, M. *Reifying RDF: What Works Well with Wikidata*? In Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2015), Bethlehem, PA, USA, 11 October 2015; pp. 32–47.
13. Roffia, L.; Azzoni, P.; Aguzzi, C.; Viola, F.; Antoniazzi, F.; Salmon Cinotti, T. Dynamic Linked Data: A SPARQL Event Processing Architecture. *Future Internet* **2018**, *10*, 36. [CrossRef]
14. Harary, F. *Graph Theory*; Addison-Wesley Series in Mathematics; Addison-Wesley Pub. Co.: Boston, MA, USA, 1969.
15. Gallego, M.A.; Fernández, J.D.; Martínez-prieto, M.A.; Fuente, P.D. RDF Visualization Using a Three-Dimensional Adjacency Matrix. In Proceedings of the 4th International Semantic Search Workshop (SEMSEARCH2011), Hyderabad, India, 29 March 2011.
16. Gansner, E.R.; Koutsofios, E.; North, S.C.; Vo, K.P.a.V.K.P. A technique for drawing directed graphs\nA technique for drawing directed graphs. *IEEE Trans. Softw. Eng.* **1993**, *19*, 214–230. [CrossRef]
17. Gansner, E.R.; North, S.C. An open graph visualization system and its applications to software engineering. *Softw. Pract. Exp.* **2000**, *30*, 1203–1233. [CrossRef]
18. Gansner, E.; Koren, Y. Improved circular layouts. In *Graph Drawing*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 386–398.
19. Gansner, E.R.; Hu, Y. Efficient, Proximity-Preserving Node Overlap Removal. *J. Graph Algorithms Appl.* **2010**, *14*, 53–74. [CrossRef]
20. Binucci, C.; Chimani, M.; Didimo, W.; Liotta, G.; Montecchiani, F. *Placing Arrows in Directed Graph Drawings*; Springer: Cham, Switzerland, 2016; pp. 1–19.
21. Brandenburg, F.J.; Eppstein, D.; Goodrich, M.T.; Kobourov, S.G.; Liotta, G.; Mutzel, P. Selected Open Problems in Graph Drawing. In *Graph Drawing*; Liotta, G., Ed.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2912, pp. 515–539.
22. Shneiderman, B.; Aris, A. Network visualization by semantic substrates. *IEEE Trans. Vis. Comput. Graph.* **2006**, *12*, 733–740. [CrossRef] [PubMed]
23. Gansner, E.R.; Koren, Y.; North, S. Graph Drawing by Stress Majorization. In Proceedings of the 12th International Symposium on Graph Drawing (GD 2004), New York, NY, USA, 29 September–2 October 2004; Volume LNCS 3383, pp. 239–250.
24. Ellson, J.; Gansner, E.R.; Koutsofios, E.; North, S.C.; Woodhull, G. Graphviz and Dynagraph—Static and Dynamic Graph Drawing Tools. In *Graph Drawing Software*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 127–148.
25. Gansner, E.R.; Hu, Y.; Kobourov, S.G. *GMap: Drawing Graphs as Maps*; Springer: Berlin/Heidelberg, Germany, 2009.
26. Stolper, C.D.; Kahng, M.; Lin, Z.; Foerster, F.; Goel, A.; Stasko, J.; Chau, D.H. GLO-STIX: Graph-Level Operations for Specifying Techniques and Interactive eXploration. *IEEE Trans. Vis. Comput. Graph.* **2014**, *20*, 2320–2328. [CrossRef] [PubMed]
27. Wu, E.; Psallidas, F.; Miao, Z.; Zhang, H.; Rettig, L.; Wu, Y.; Sellam, T. Combining Design and Performance in a Data Visualization Management System. In Proceedings of the 8th Biennial Conference on Innovative Data Systems Research (CIDR '17), Chaminade, California, 8–11 January 2017.
28. Bastian, M.; Heymann, S.; Jacomy, M. Gephi: An Open Source Software for Exploring and Manipulating Networks. *Third Int. AAAI Conf. Weblogs Soc. Media* **2009**, *8*, 361–362.
29. Sayers, C. *Node-Centric Rdf Graph Visualization*; Mobile and Media Systems Laboratory, HP Labs: Palo Alto, CA, USA, 2004.
30. Dokulil, J.; Katreniakov, J. Visualization of Large Schemaless RDF Data. In Proceedings of the International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM'07), Papeete, France, 4–9 November 2007; pp. 243–248.
31. Deligiannidis, L.; Kochut, K.J.; Sheth, A.P. RDF Data Exploration and Visualization. In Proceedings of the ACM First Workshop on CyberInfrastructure: Information Management in eScience, Lisbon, Portugal, 9 November 2007; pp. 39–46. [CrossRef]

32. Lohmann, S.; Negru, S.; Haag, F.; Ertl, T. VOWL2: User-Oriented Visualization of Ontologies. In Proceedings of the Knowledge Engineering and Knowledge Management: 19th International Conference, EKAW 2014, Linköping, Sweden, 24–28 November 2014.
33. Brunk, S.; Heim, P. Tfacet: Hierarchical faceted exploration of semantic data using well-known interaction concepts. *CEUR Workshop Proc.* **2011**, *817*, 31–36.
34. Yee, K.P.; Swearingen, K.; Li, K.; Hearst, M. Faceted metadata for image search and browsing. In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, Ft. Lauderdale, FL, USA, 5–10 April 2003; pp. 401–408.
35. Heim, P.; Lohmann, S.; Stegemann, T. Interactive relationship discovery via the semantic web. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6088 LNCS, pp. 303–317.
36. Heim, P.; Lohmann, S. Semlens: Visual analysis of semantic data with scatter plots and semantic lenses. In Proceedings of the 7th International Conference on Semantic Systems—I-Semantics '11, Graz, Austria, 7–9 September 2011; pp. 175–178.
37. Bier, E.A.; Stone, M.C.; Pier, K.; Buxton, W.; DeRose, T.D. Toolglass and magic lenses: The see-through interface. In Proceedings of the ACM 20th Annual Conference on Computer Graphics and Interactive Techniques, Anaheim, CA, USA, 2–6 August 1993; pp. 73–80.
38. Heim, P.; Schlegel, T.; Ertl, T. A Model for Human-Computer Interaction in the Semantic Web Categories and Subject Descriptors. In Proceedings of the 7th International Conference on Semantic Systems, Graz, Austria, 7–9 September 2011; pp. 150–158.
39. Dadzie, A.S.; Pietriga, E. Visualisation of linked data—Reprise. *Semant. Web* **2017**, *8*, 1–21. [CrossRef]
40. Nuzzolese, A.G.; Presutti, V.; Gangemi, A.; Peroni, S.; Ciancarini, P. Aemoo: Linked data exploration based on knowledge patterns. *Semant. Web* **2017**, *8*, 87–112. [CrossRef]
41. Storey, M.A.; Lintern, R.; Ernst, N.; Perrin, D. Visualization and protege. In Proceedings of the 7th International Protégé Conference, Bethesda, Maryland, 6–9 July 2004.
42. Sivakumar, R.; Arivoli, P. Ontology visualization PROTÉGÉ tools—A review. *Int. J. Adv. Inf. Technol.* **2011**, *1*, 1–11. [CrossRef]
43. Storey, M.; Musen, M.; Silva, J.; Best, C.; Ernst, N.; Fergerson, R.; Noy, N. Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protégé. In Proceedings of the Workshop on Interactive Tools for Knowledge Capture (K-CAP-2001), Victoria, BC, Canada, 20 October 2001; pp. 1–9. [CrossRef]
44. Lomov, P.; Shishaev, M. Creating Cognitive Frames Based on Ontology Design Patterns for Ontology Visualization. In *Knowledge Engineering and the Semantic Web*; Klinov, P., Mouromtsev, D., Eds.; Springer International Publishing: Cham, Switzerland, 2014; pp. 90–104.
45. Roffia, L.; Morandi, F.; Kiljander, J.; D'Elia, A.; Vergari, F.; Viola, F.; Bononi, L.; Salmon Cinotti, T. A semantic publish-subscribe architecture for the Internet of Things. *IEEE Internet Things J.* **2016**, *3*, 1274–1296. [CrossRef]
46. Rinne, M.; Nuutila, E. Constructing Event Processing Systems of Layered and Heterogeneous Events with SPARQL. *J. Data Semant.* **2017**, *6*, 57–69. [CrossRef]