Fusion – Visually Exploring and Eliciting Relationships in Linked Data

Samur Araujo[1], Geert-Jan Houben[1], Daniel Schwabe[2], Jan Hidders[1]

[1]Delft University of Technology, PO Box 5031, 2600 GA Delft, the Netherlands
[2]PUC-Rio, Rua Marques de Sao Vicente, 225, Rio de Janeiro, Brazil

{s.f.cardosodearaujo, g.j.p.m.houben, a.j.h.hidders}@tudelft.nl
dschwabe@inf.puc-rio.br

Abstract. Building applications over Linked Data often requires a mapping between the application model and the ontology underlying the source dataset in the Linked Data cloud. This mapping can be defined in many ways. For instance, by describing the application model as a view over the source dataset, by giving mappings in the form of dependencies between the two datasets, or by inference rules that infer the application model from the source dataset. Explicitly formulating these mappings demands a comprehensive understanding of the underlying schemas (RDF ontologies) of the source and target datasets. This task can be supported by integrating the process of schema exploration into the mapping process and help the application designer with finding the implicit relationships that she wants to map. This paper describes Fusion - a framework for closing the gap between the application model and the underlying ontologies in the Linked Data cloud. Fusion simplifies the definition of mappings by providing a visual user interface that integrates the exploratory process and the mapping process. Its architecture allows the creation of new applications through the extension of existing Linked Data with additional data.

Keywords: semantic web, data interaction, data management, RDF mapping, Linked Data

1 Introduction

Nowadays, the Linked Data[1] cloud provides a new environment for building applications where many datasets are available for consumption. Although data in this cloud is ready to use, applications over the Linked Data cloud have currently an intrinsic characteristic: they consume RDF[2] data "as is", since designers do not have write permission over the data in the cloud which would enable them to change the data in any way. This fact raises an important issue concerning the development of

[1] Linked Data - http://linkeddata.org/
[2] http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

applications over Linked Data: how to fill the gap between the ontology associated with the application model and the ontology used to represent the underlying data from the Linked Data cloud? The main benefit of mapping these two models is that then Linked Data can be accessed through properties defined in the application model, which is more convenient for the designer, consequently simplifying considerably the development and maintenance of the application.

Although a number of techniques can be applied for mapping two RDF models, such as ontology matching, or inference rules, or views over RDF data, they often do not take into account that expressing the mapping rules themselves is a separate challenge, since in most cases Linked Data sources are represented using domain-specific ontologies that do not explicitly offer all common properties in the domain. Take for example DBLP[3] Linked Data, one of the best-known bibliography information sources available as Linked Data. Its ontology does not have an explicit property that connects directly co-authors, a common property in this domain. Although DBLP Linked Data contains paths that represent this relationship, it is not trivial to find them. Indeed, it requires understanding the schema behind the data and how this relationship is implicitly represented in this dataset. Similar examples can be found in any dataset in the Linked Data cloud, where the required information is implicitly encoded in the instance of data.

In this context, two specific and common scenarios often occur. The first is where the designer needs to express a mapping between a property in her application model (e.g. how a City is located in a Country) and a path in the RDF graph of the given dataset (e.g. a City belongs to a Province which belongs to a Country). Another example of this scenario can be given in the domain of government data. Suppose you are building an application over the GovTrack.Us[4] dataset and its application model requires a property *isSenatorOf* that directly connects instances of the class Politician to instances of the class State (e.g: Christopher Bond is a senator from Missouri). However, this relationship is not explicitly represented in the GovTrack.Us ontology. In order to obtain this relationship, the designer has to use the path *Senator -> has Role-> forOffice-> represents -> State* which, in this RDF graph, represents the relationship. Note that the designer needs a clear understanding of the GovTrack.Us schema in order to find the corresponding path to be mapped. The second scenario occurs when the mapping is in fact a computation over the existing data that produces a new explicit data value. For instance, a mapping between a property *screen resolution* from the application model and the concatenation of the properties *screen width* and *screen height* defined in the target dataset.

Note that in those scenarios for defining those mappings special attention should be paid to the exploratory process, especially when it demands from the designer to dive into the instances and the schema of the source dataset in order to find implicit relationships, which is not a straightforward task at all. Some authors have shown that visual exploration [1, 9] can help users to understand an unknown schema used to represent a known domain. Although those mechanisms help users to query an unknown schema, it will be always easier to explore a schema that is closer to the application models, often expressed in a specific application ontology. Although

[3] http://dblp.l3s.de/d2r/

[4] http://www.govtrack.us/

many tools are available for exploring Linked Data and for expressing mapping rules between RDF models, there is still a lack of tools that integrate these processes.

This paper presents Fusion[5], a lightweight framework to support application designers in building applications over Linked Data. It supports designers in mapping the ontology of the used Linked Data sources to their application model by integrating the process of exploration of the target schema with the task of expressing a mapping rule itself. Fusion features a visual user interface that guides the designer in the process of specifying a mapping rule. It uses a standard RDF query language and allows Linked Data to be accessed using properties defined in the application model, consequently simplifying the use of Linked Data in a specific context.

The remainder of this paper is organized as follows. Section 2 presents relevant related work. Section 3 describes how Fusion supports the designer in deriving rules; while Section 4 describes Fusion's architecture. Section 5 presents some examples and shows how Fusion solves the problem of enriching access to Linked Data with application model properties. Finally, Section 6 presents the conclusion of this work.

2 Related Work

2.1 Ontology Mapping

The problem of mapping data models can also be conceived as an ontology-mapping problem, since it encompasses describing existing data in another vocabulary. In [8] a SPARQL extension is proposed to achieve that. Their solution merges SPARQL++ [3] and PSPARQL [9], two extensions of the SPARQL specification. The first extension adds some functions for enabling SPARQL to translate one vocabulary to another one by just using SPARQL *CONSTRUCT*. The second one adds path expressions to SPARQL, allowing a better navigation through the graph. Together they empower the SPARQL language to perform ontology mapping over two or more ontologies. Although the theory is given, the authors do not provide a concrete implementation especially because the proposed primitives have many implications for the performance of the query over the distributed environment of Linked Data.

2.2 SPARQL Construct Queries and their Extensions

Another way to solve the problem of mapping RDF datasets is by specifying a *CONSTRUCT* query in SPARQL [2] that derives the triples in the target data set from the source data set. The resulting graph can then be stored in an arbitrary RDF repository. However, the *CONSTRUCT* query has limited expressive power, since some computation over the original RDF triples cannot be done, such as string manipulation and aggregation. For instance, using this approach it is not possible to

[5] http://www.wis.ewi.tudelft.nl/index.php/fusion

generate the triple that would represent the mapping between the properties *screen width* and *screen height* (shown in Fig. 1) to a property *resolution* (shown in Fig. 2) that is their simple concatenation.

<http://sw.tv.com/id/2660> <http://sw.tv.com/screen_width> "128" .
<http://sw.tv.com/id/2660> <http://sw.tv.com/screen_height> "160" .

Fig.1 – A resource with predicates *screen width* and *screen height*.

<http://sw.tv.com/id/2660> <http://sw.tv.com/resolution> "128x160" .

Fig.2 – A resource with predicate resolution.

Polleres et al. [3] have proposed an extension of *CONSTRUCT* that overcomes such limitations, however this extension is limited to a specific RDF query engine that implements this SPARQL extension. Therefore, at this moment, such a solution is not feasible for the Semantic Web environment, which is very diverse in terms of query engines - the majority of data is stored in repositories that implement variations of the standard SPARQL specification that do not include the extensions discussed here.

2.3 Views over RDF Data

Another way to specify mappings between different representational models is by defining *views* [5, 6, 7]. This concept is well known in the field of database theory, and can be used to aggregate and personalize data. A *view* is a query accessible as a virtual table composed of the result of the query. Although *views* are frequently used in relational databases, building *views* over Linked Data presents many additional challenges. Issues such as view maintenance (including updates) and querying over virtual (non-materialized) views in the distributed environment of Linked Data are still open problems, besides several other performance issues that arise.

Volz et al. [4] have proposed a language based on RQL [5] for specifying views over RDF data. It defines views over RDF classes and views of RDF properties. Although this proposal presents a complex specification of views over RDF, it cannot solve the simple scenario described in Section 2.2, and its solution is based on RQL, which is not the standard RDF query language used nowadays. Magkanaraki et al. [7] have proposed a view specification language also based on RQL. Its processing model is based on materialized views. Chen et al. [6] present a scenario of accessing relational data using RDF views. In their approach a query over a view result in query rewriting that exploits the semantics of RDF primitives, such as, *subPropertyOf* or *subClassOf*. While their approach enriches the access to the relational data, it does not cover the transformations over the data that we are considering here, moreover it is focused on mapping relational schema to an RDF/S ontology.

2.4 SWRL Rules

Hassanpour et. al [12] proposes a tool for supporting the user on creating SWRL[6] rules. Their tool contains a visual interface that guides the user in visualizing, managing and eliciting SWRL specifications. Although this tool can be used to map two models using SWRL rules, it does not integrate the process of specifying the rules with the process of exploring an unknown schema, which is the main aim of Fusion.

2.5 RDF Exploration

RelFinder [1] is a visual tool for finding *n*-ary relationships between RDF resources. It contains a visual interface that allows the user to visualize the relationship in a directed graph layout. Basically, RelFinder issues a set of queries against a specific SPARQL endpoint in order to find relationships between two or more RDF resources. RelFinder aims to be a better mechanism for finding relationships among data than any other exploratory mechanism. Explorator [9] is another tool that aims to facilitate the querying of instances of an unknown RDF schema, consequently allowing the user to discover relations between data instances even without previous knowledge of the domain. These tools re-enforce the idea that accessing RDF data is not a trivial task and demands a complex exploratory model behind it. In spite of the fact that they support users in finding relationships between data, they do not solve the problem of accessing the Linked Data through a schema associated with the application model.

2.6 Interlinking

From an operational point of view the mapping of two RDF models can be perceived as the addition of new triples to the original dataset for any new relationship expressed in the target ontology. Clearly this task requires some sort of automation. For instance, Silk [10] is a linking framework for discovering relationships between data items within different Linked Data sources. By specifying rules, the application designer can define how two distinct sets of resources, possibly belonging to distinct endpoints, can be interlinked, and as a result it produces a graph with all discovered connections. Although Silk automates the process of interlinking resources, Fusion goes one step further, since it supports also the process of specifying the rule. They solve two different problems: Silk interlinks two disconnected RDF graphs while Fusion extends the knowledge for a single endpoint. Although Silk's mapping language can be used for materializing the rules defined in Fusion, it does not support the full process supported by Fusion, which also includes, most notably, the discovery of a path in the schema to be mapped. While Silk allows the user to serialize a rule, it does not support her in finding it and expressing it.

[6] http://www.w3.org/Submission/SWRL/

3 Discovering and Deriving RDF Relationships

The main aim of Fusion is to help the designer in discovering relationships in RDF graphs that exist in the Linked Data cloud and specifying rules for the derivation of new properties for these relationships. We refer to this process as *relationship derivation*. The result of the *relationship derivation* process is a set of rules such that each produces RDF triples based on queries over an existing RDF graph. The evaluation of a rule results in a set of triples, each of which contains either a new *object property*[7] or a new *datatype property*. In the cases where it results in a new *object property*, the triples produced connect existing resources, while in the case where it derives a new *datatype property* the triples produced connect existing resources with values computed by a function over the RDF graph being queried. In the remainder of this section we describe how Fusion supports the designer in specifying these derivation rules.

3.1 Deriving Object Property Relationships

The main issue regarding the derivation of new *object property relationship* is to specify the correspondence between resources. For example, if a user wants to create a new object property *locatedIn* that directly connects cities to their respective countries, she needs to specify the relationship between cities and countries in the existing data, i.e. which cities are located in which country. Such a correspondence can be obtained by following a certain path between two resources in the RDF graph of the source dataset. For example, Fig. 3 shows a sub-graph of Geonames Linked Data[8] that represents a path between the resource for the city of Delft and that of the country The Netherlands. By using the predicates in this example path and generalizing the intermediate nodes, it is possible to generalize such an example correspondence to apply to all cities and respective countries in this dataset, and thus bring the correspondence to the class level. By exploiting this resulting path, Fusion can map all cities to their corresponding country and thereby add a new object property to the original graph. Note that this process maps a newly added relationship that is defined in the application model, onto an implicit relationship that exists in the original graph. In the example above, the object property *locatedIn* defined in an application model could be mapped to the generalization of the path between Delft and the Netherlands *geo:parentFeature* — *Province of Zuid-Holland* — *geo:parentFeature*.

[7] http://www.w3.org/TR/owl-ref/#ObjectProperty-def

[8] http://www.geonames.org/ontology/

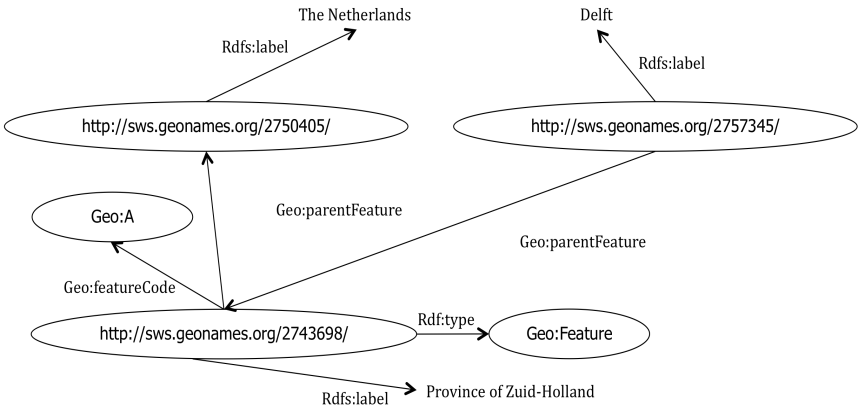


Fig. 3 – A path between the resources Delft and The Netherlands in Geonames Linked Data

3.1.1 A Path Discovery Algorithm

The first step in the mapping process is to find the paths that could potentially be used in the mapping. Fusion automates this step by eliciting all possible paths in an RDF graph that connect two example RDF resources (e.g. Delft and The Netherlands) that have a specified maximum length. Thus, finding the relationship between two RDF resources becomes finding a path in the RDF graph that would allow navigating from the one resource to the other one. This process can be implemented as a modified version of the *breadth-first search algorithm (BFS)* with a maximum limit on the depth of the search and without the restriction that it should stop when the goal node is reached. Consequently, it can be used to retrieve all paths in the graph within a maximum length from the source to the target node. This algorithm is applied to the RDF graph by interpreting each triple as an undirected edge between its subject and object. Since this algorithm is a small variation on the standard BFS and retrieves all possible paths from a to b of a maximum length d, its complexity is $O(c^d)$, where c is the maximum branching factor in the graph. This asymptotic complexity is in this case the theoretical optimum since it describes the size of the output.

3.1.2 Implementing the Path Discovery Algorithm over a SPARQL Endpoint

Considering that Fusion searches for paths in a Linked Data dataset, the path discovery algorithm needs to be implemented as a set of SPARQL queries, since the most direct way to search in an RDF graph in the Linked Data cloud is by issuing SPARQL queries over its remote SPARQL endpoint. In order to generate these queries we consider the RDF graph as an *undirected graph* as previously described. Thus, all paths with length n from node a to node b in this graph can be obtained with a set of SPARQL queries containing 2^n queries. Since we want to ignore the direction in the graph, we issue a distinct graph pattern for all possible choices of direction for each triple pattern in the path. Each query in this set contains n connected triple patterns, one for each edge in the path. For example, to obtain all paths between *a* and *b* with length 3, 8 (= 2^3) graph patterns, each containing 3 triple patterns, are generated. Fig. 5 shows all these 8 patterns.

```
1 (:a,:p1,:a2),(:a2,:p2,:a3),(:a3,:p3,:b)
2 (:a2,:p1,:a),(:a2,:p2,:a3),(:a3,:p3,:b)
3 (:a,:p1,:a2),(:a3,:p2,:a2),(:a3,:p3,:b)
4 (:a2,:p1,:a),(:a3,:p2,:a2),(:a3,:p3,:b)
5 (:a,:p1,:a2),(:a2,:p2,:a3),(:b,:p3,:a3)
6 (:a2,:p1,:a),(:a2,:p2,:a3),(:b,:p3,:a3)
7 (:a,:p1,:a2),(:a3,:p2,:a2),(:b,:p3,:a3)
8 (:a2,:p1,:a),(:a3,:p2,:a2),(:b,:p3,:a3)
```

Fig. 5 – Graph patterns generated for path length 3.

Each of these patterns will be transformed into a single SPARQL query, as shown in Fig. 6 for pattern 1 from Fig. 5, where *a* and b were specified as the resource Christopher Bond (http://www.rdfabout.com/rdf/usgov/congress/people/B000611) and the resource Missouri (http://www.rdfabout.com/rdf/usgov/geo/us/mo.), respectively. In this example, the path connects the US politician Christopher Bond with the state (Missouri) that he represents.

```
PREFIX Geo: <http://www.rdfabout.com/rdf/usgov/geo/us/>
PREFIX Gov:
<http://www.rdfabout.com/rdf/usgov/congress/people>
SELECT DISTINCT ?p1 ?a2 ?p2 ?a3 ?p3
WHERE {
    Gov:B000611 ?p1 ?a2 .
    ?a2 ?p2 ?a3 .
    ?a3 ?p3 Geo:mo .
}
```

Fig. 6 – Query performed for graph pattern 1 from Fig. 5.

Thus, when all graph patterns are executed, all paths of length *n* are retrieved. Fig. 7 shows a sample path found as result of the query from Fig. 6 being issued over the GovTrack.Us endpoint.

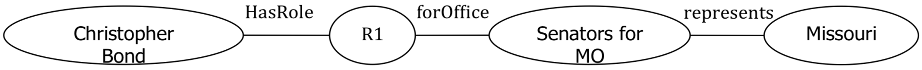


Fig. 7– A sample path found between the resources Christopher Bond (a senator) and Missouri (a US state) in the GovTrack.Us dataset.

This algorithm is similar to RelFinder's, however RelFinder's algorithm considers the RDF graph as a *directed graph* and it searches *only* for 4 graph patterns, which does not cover all possible paths between *a* and *b*.

3.1.3 Derivation Process

The algorithm described previously is used in the first stage of the derivation process. The complete derivation process ends with the designer choosing one of the paths found in the first stage and generalizing it to find correspondences between two broader (more general) sets of resources. As a result of this procedure, a derivation rule is produced.

In order to apply the chosen path to a broader set of resource pairs the designer needs to generalize all nodes in the path. For instance, the path in Fig. 8 shows a generalization of the path in Fig. 7 that considers all sources and targets (their generalization is indicated by the *?*) that are connected through the path *hasRole* — *R1 (blank node)* — *forOffice* — *Senators for MO* — *represents,* i.e., it will find the correspondence between all senators and the state of Missouri.

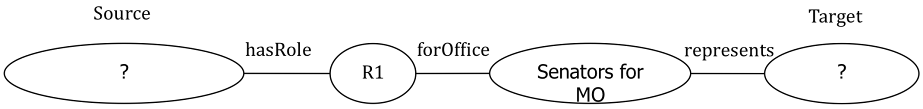


Fig. 8 – A possible generalization between senator resources and the Missouri resource.

Fig. 9 shows another (yet more general) generalization that can find the correspondence between all senators and the respective state that they represent, since now *all* intermediate nodes are variables.

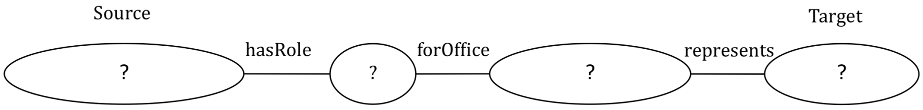


Fig. 9 – A possible generalization between senator resources and the respective US state that they represent.

It should be noticed that the predicates in the path are not generalized and remain fixed. These generalizations define derivation rules, which select the resources that will be interconnected.

For the designer to control this generalization process, we provide a graphical user interface that will be shown later.

3.2 Deriving New Datatype Properties

Fusion also supports application designers to extend the original dataset with *datatype properties*. As Fusion's goal is to allow application designers to map a property in their application model to an existing Linked Data dataset, the values of the new *datatype properties* are computed over the existing values in the original dataset.

Formally, a derivation rule that produces *datatype properties* is defined by a tuple (q, p, f) with a query q, a predicate name p, and a function f. The query q defines a function that maps an RDF graph to a set of URIs in that graph, which defines the set of resources for which the new *datatype* property is defined. The predicate name p defines the predicate name of the new property. Finally the function f maps an RDF graph and a particular URI within that graph to an RDF value. The result of applying such a rule to an RDF graph G is the addition of all tuples (s, p, o) such that $s \in q(G)$ and $o = f(G, s)$.

4 Architecture Overview

Fusion's implementation architecture provides a complete environment to specify and execute a derivation rule. An overview of this architecture is shown in Fig. 10. The specification of the rules is supported in Fusion's user interface that will be explained further in the section 5. Fusion's server engine is responsible for executing the derivation rule itself. During the process of executing of a rule, it queries a source endpoint in the Linked Data, processes the result, and produces a set of new triples that will be added to the Fusion repository. Any RDF data store can be used as Fusion's repository. Currently, Fusion implements adapters for Sesame[9] and Virtuoso[10] data stores, although other adapters can be easily added to its architecture. All derived triples in Fusion contain as subject a resource that belongs to the queried dataset, so the derived data is intrinsically interlinked with the Linked Data cloud. For this reason, a query over a federation of endpoints, that includes the Fusion repository endpoint, will allow the designer to have a view over the Linked Data that also includes the properties defined in her application model.

There are other approaches to how to store the derived triples. For example, it is possible to use a user-defined namespace for the subjects of the derived triples, and add an owl:sameAs statement linking it to the original URIs, as opposed to using the original URIs directly as subject. The shortcoming of this alternative is that others who want to find out about the new derived properties would not look for them in Fusion's local repositories, but in the original URI, which doesn't know about these new derived properties. On the other hand, with the current approach, if the VoID description of Fusion's local repository is updated to reflect the inclusion of information about the original URI, then others would still find its SPARQL endpoint when looking for endpoints containing information about that URI, thus having access to all derived triples in the Fusion's repository.

So it is really a modeling trade-off, with no clear advantage to either side, and our solution requires less involvement from third parties (e.g., for owl:sameAs processing).

Although Fusion does not serialize a rule before executing it, any serialization mechanism could be used in its architecture. For example, its rules could be serialized

[9] http://www.openrdf.org/

[10] http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/

as inference rules, by using Spin Inference Notation[11], which contains a collection of RDF vocabularies enabling the use of SPARQL to define constraints and inference rules on Semantic Web models. Although this configuration is theoretically possible, executing inference rules, or even instantiating a *virtual view* over the Linked Data is still an open problem, since it raises many performance issues. Such issues do not occur in the current Fusion architecture because it materializes the result of the rules as new triples in the Fusion repository. The performance of querying data that is already materialized is always faster than querying data that needs to be processed at runtime. The main drawback with materializing the result of the rules is that once the original source is updated, the rules have to be executed again. Furthermore, detecting these changes in the Linked Data is not trivial. Such synchronizing or updating issues are actually the realm of research (and practice) in (database) view definitions and updates. The performance trade-offs in each case are well known, and are addressed by researchers working in that area, which while relevant, is not the research focus for Fusion at this state. Fusion is be able to benefit from whatever techniques are available on this topic.

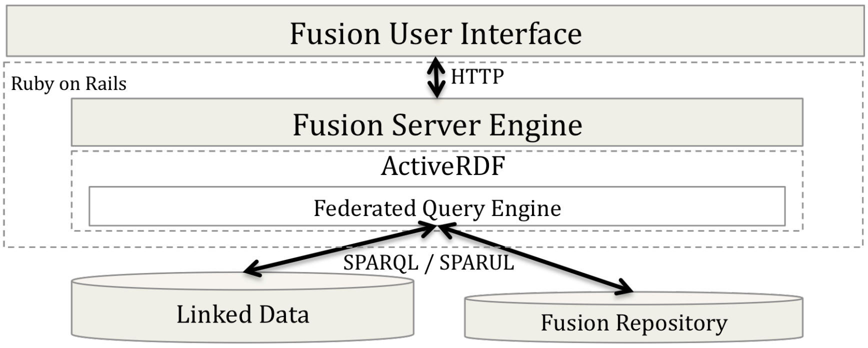


Fig. 10 – Fusion's architecture overview.

Fusion is implemented in Ruby on Rails[12] as a web application. It uses the ActiveRDF API [11] that allows an RDF graph to be accessed in the *object-oriented* paradigm. By using this API the properties of an RDF *resource* can be accessed as an attribute of its corresponding Ruby[13] object. For instance, the predicate http://www.geonames.org/ontology#population can be accessed as *<resource>.population*. This architecture allows the designer to write complex functions for computing a new *datatype* property value using the full power of the Ruby language, which cannot be achieved simply by using the SPARQL language.

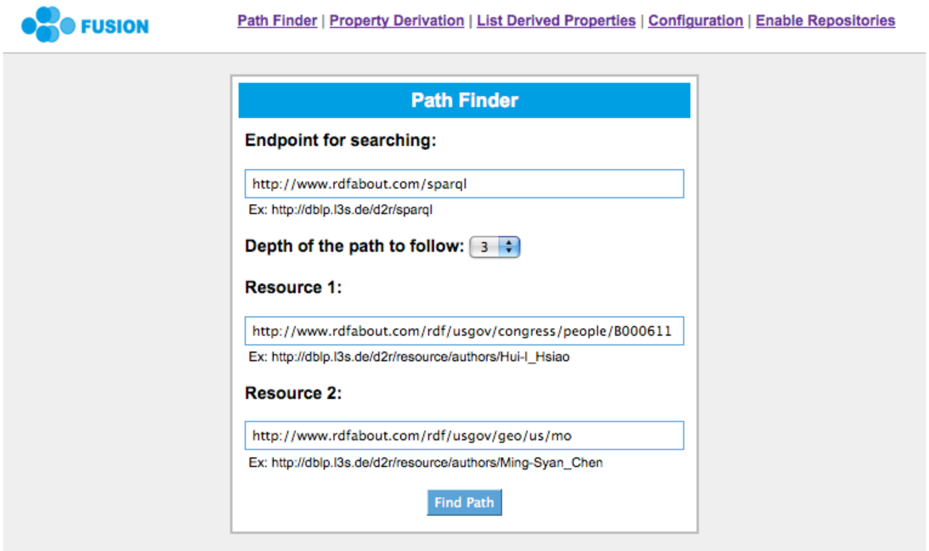
[11] http://www.spinrdf.org/
[12] http://rubyonrails.org/
[13] http://www.ruby-lang.org/en/

5 Examples of Use

This section describes two concrete scenarios that illustrate the use of Fusion to create an application by extending Linked Data sources with additional properties.

5.1 Scenario 1 – Adding the *isSenatorOf* object property to GovTrack.Us

In this example, we suppose that the designer wants to establish the relationship between US senators and the US state that they represent. Therefore she needs to construct a derivation rule that will find and define such a correspondence between politicians and states in GovTrack.Us's Linked Data repository. In the first step in the process, the designer provides an example of two resources in GovTrack.Us that she knows in advance that are actually related, for instance, politician Christopher Bond and the state of Missouri. Also, she needs to declare the GovTrack.Us endpoint to be queried and the maximum depth of the path. This step is shown in Fig. 11.


FUSION
Path Finder | Property Derivation | List Derived Properties | Configuration | Enable Repositories
Path Finder
Find Path

Endpoint for searching:

http://www.rdfabout.com/sparql

Ex: http://dblp.l3s.de/d2r/sparql

Depth of the path to follow: 3

Resource 1:

http://www.rdfabout.com/rdf/usgov/congress/people/B000611

Ex: http://dblp.l3s.de/d2r/resource/authors/Hui-I_Hsiao

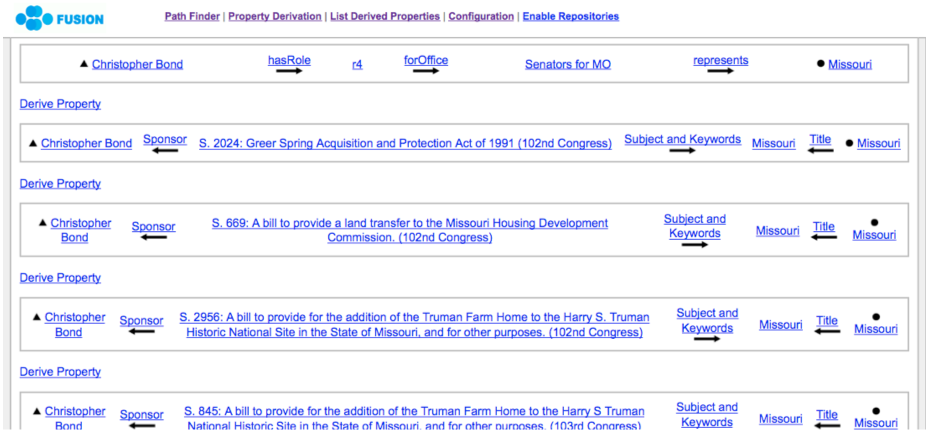
Resource 2:

http://www.rdfabout.com/rdf/usgov/geo/us/mo

Ex: http://dblp.l3s.de/d2r/resource/authors/Ming-Syan_Chen

Fig. 11 – Fusion's interface for finding a path between two known resources

As the result of this first step, Fusion shows all the paths that connect these two example resources satisfying the maximum path length. This result is shown in Fig. 12. In this example, the paths found have a maximum length of 3.


FUSION
Path Finder | Property Derivation | List Derived Properties | Configuration | Enable Repositories
Derive Property

▲ Christopher Bond —hasRole→ r4 —forOffice→ Senators for MO —represents→ ● Missouri

▲ Christopher Bond ←Sponsor— S. 2024: Greer Spring Acquisition and Protection Act of 1991 (102nd Congress) —Subject and Keywords→ Missouri ←Title— ● Missouri

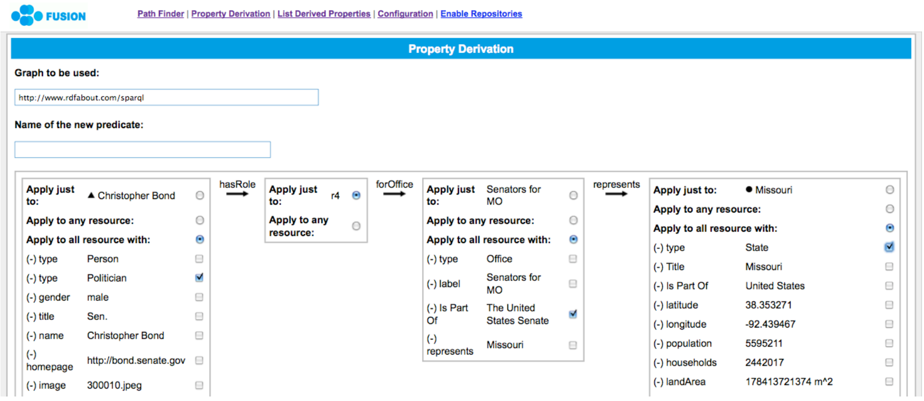
▲ Christopher Bond ←Sponsor— S. 669: A bill to provide a land transfer to the Missouri Housing Development Commission. (102nd Congress) —Subject and Keywords→ Missouri ←Title— ● Missouri

▲ Christopher Bond ←Sponsor— S. 2956: A bill to provide for the addition of the Truman Farm Home to the Harry S. Truman Historic National Site in the State of Missouri, and for other purposes. (102nd Congress) —Subject and Keywords→ Missouri ←Title— ● Missouri

▲ Christopher Bond ←Sponsor— S. 845: A bill to provide for the addition of the Truman Farm Home to the Harry S Truman National Historic Site in the State of Missouri, and for other purposes. (103rd Congress) —Subject and Keywords→ Missouri ←Title— ● Missouri

Fig. 12 – Fusion’s interface showing the discovered paths.

In this view, the designer can now look for the path that has the intended semantics. Note that with this view the tool assists the designer in this discovery process, since she does not need to query the schema manually in order to find these paths. The first path shown in Fig. 12 indicates that the politician Christopher Bond has a role as senator representing the state Missouri, and in our example case the designer can now infer that this is an instance of the path that she is looking for. After this conclusion, the designer chooses that instance to be the template for the rule.


FUSION
Path Finder | Property Derivation | List Derived Properties | Configuration | Enable Repositories
hasRole
forOffice
represents

Property Derivation

Graph to be used:

http://www.rdfabout.com/sparql

Name of the new predicate:

Apply just to:	▲ Christopher Bond	○
Apply to any resource:		○
Apply to all resource with:		●
(-) type	Person	☐
(-) type	Politician	☑
(-) gender	male	☐
(-) title	Sen.	☐
(-) name	Christopher Bond	☐
(-) homepage	http://bond.senate.gov	☐
(-) image	300010.jpeg	☐

Apply just to:	r4	●
Apply to any resource:		○

Apply just to:	Senators for MO	○
Apply to any resource:		○
Apply to all resource with:		●
(-) type	Office	☐
(-) label	Senators for MO	☐
(-) Is Part Of	The United States Senate	☑
(-) represents	Missouri	☐

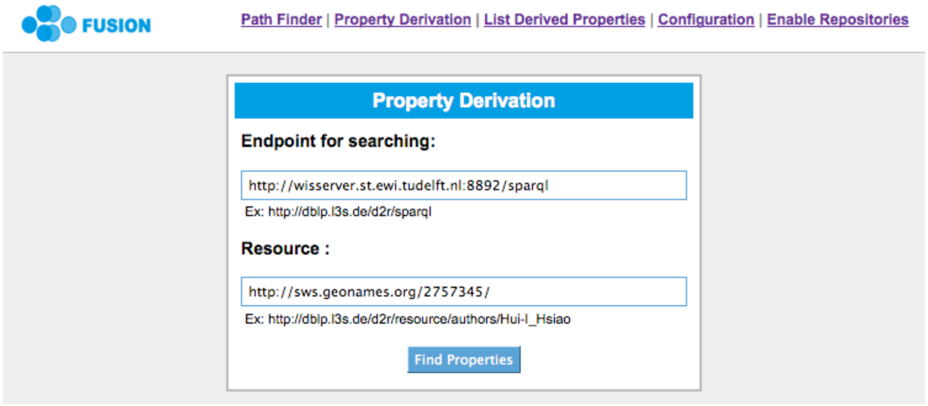
Apply just to:	● Missouri	○
Apply to any resource:		○
Apply to all resource with:		●
(-) type	State	☑
(-) Title	Missouri	☐
(-) Is Part Of	United States	☐
(-) latitude	38.353271	☐
(-) longitude	-92.439467	☐
(-) population	5595211	☐
(-) households	2442017	☐
(-) landArea	178413721374 m^2	☐

Fig. 13 – Generalizing the path for the property isSenatorOf in GovTrack.Us.

In the next step, shown in Fig. 13, the designer will define the derivation rule itself, which means that she visually formulates a query, which generalizes the selected path from the first step into a query that selects the elements to be connected through the property *isSenatorOf*. To complete this operation she also needs to define the graph where the derived triples will be stored and a specific URI to be used as the predicate of the new triples, which in this example will be *http://example.org/isSenatorOf*. Note that in this example 3 nodes were generalized such that only paths between resources of the RDF type *Politician* and RDF type *State* that contain an intermediate node that *is part of* the United States Senate will be considered during the derivation process. Consequently, Fusion will derive the new property *isSenatorOf* for all instances of the class Politician that are connected to an instance of the class State through the specified path. The whole process concludes with Fusion adding new triples to Fusion repository.

5.2 Scenario 2 – Adding a Datatype Property *citySize* to Geonames

In this example, we suppose that the designer wants to derive a new property *citySize* for cities in Geonames to distinguish small and large cities. Therefore she needs to create a derivation rule that will compute the appropriate values for this new property. She will want this property to have the value 'small' for cities with less than 1.000.000 inhabitants, and 'large' otherwise. As the first step in the process, she needs to provide the Geonames Linked Data endpoint to be queried and a resource (a city) in Geonames as an example. In this case she supplies the URI http://sws.geonames.org/2757345/, which represents the city of Delft. This step is shown in Fig. 14.


FUSION
Path Finder | Property Derivation | List Derived Properties | Configuration | Enable Repositories
Property Derivation
Find Properties

Endpoint for searching:

http://wisserver.st.ewi.tudelft.nl:8892/sparql

Ex: http://dblp.l3s.de/d2r/sparql

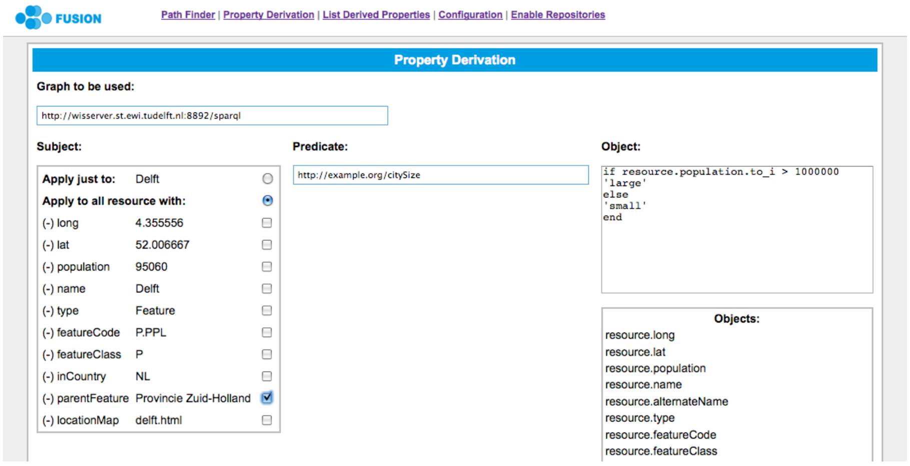
Resource :

http://sws.geonames.org/2757345/

Ex: http://dblp.l3s.de/d2r/resource/authors/Hui-I_Hsiao

Fig. 14 – Fusion's interface for deriving a data type property

In the next step, Fusion uses the city URI to construct the visual interface where the designer will express the derivation rule *R=(q,p,f)*. In this interface, she visually defines the query *q*, a URI for the new property *p,* and a Ruby[14] expression that will be used as the function *f*. In Fig. 15 we show Fusion's datatype property derivation view.


FUSION
Path Finder | Property Derivation | List Derived Properties | Configuration | Enable Repositories
Property Derivation

Graph to be used:

http://wisserver.st.ewi.tudelft.nl:8892/sparql

Subject:

Apply just to:	Delft	○
Apply to all resource with:		●
(-) long	4.355556	□
(-) lat	52.006667	□
(-) population	95060	□
(-) name	Delft	□
(-) type	Feature	□
(-) featureCode	P.PPL	□
(-) featureClass	P	□
(-) inCountry	NL	□
(-) parentFeature	Provincie Zuid-Holland	☑
(-) locationMap	delft.html	□

Predicate:

http://example.org/citySize

Object:

```
if resource.population.to_i > 1000000
'large'
else
'small'
end
```

Objects:

resource.long
resource.lat
resource.population
resource.name
resource.alternateName
resource.type
resource.featureCode
resource.featureClass

Fig. 15– Fusion’s datatype property derivation interface.

In this view the designer specifies that the new property is to be defined for all resources for which the predicate *parentFeature* equals *Province Zuid-Holland*, i.e., it will compute it for all cities in the province of Zuid-Holland. Also she defines that the URI of the new property will be http://example.org/citySize. As Fusion was developed in Ruby, using the ActiveRDF API, the function *f* can be defined as a Ruby expression that for this example is shown in Fig. 16.

```
resource.population.to.i > 1.000.000 ? 'large':'small'
```

Fig. 16– Sample Ruby expression for computing the *citySize* value.

This process ends with Fusion adding new triples to the Fusion repository.

6 Conclusion and Future Work

Linked Data is a cloud of distributed datasets that can be used “as is” for building applications. However, its data is often expressed in a low-level ontology that does not reflect the ontology associated with the application model. In order to fill the gap between these two representational models it is necessary to somehow map them. Although there exist approaches for solving this problem, such as ontology matching techniques, views over RDF and inference rules, they do not consider this task as a process that also involves Linked Data schema exploration. In others words, whatever strategy is used, it will demand from the designer to identify in both models exactly what to map and how to map it, which is not trivial, since it also demands a clear understanding of the underlying schema in the used Linked Data. Fusion integrates the exploratory task into the process of mapping, thereby helping the designer with identifying the relationships between her application model and the Linked Data schema, and also in providing a full architecture for expressing the mapping and extending the used Linked Data such that it implements the application model.

Fusion works by querying Linked Data and extending it by adding new data into the cloud without directly altering the original dataset. Fusion also provides a visual interface that allows the user to explore Linked Data, express the rules and derive new data, which in the end covers the whole process of mapping and extending. As Fusion materializes the result of the mapping as new triples in an extra endpoint in the cloud, it consequently allows the separation of the processes of building the application and managing the mapping between models.

References

1. Heim, P.; Lohmann, S.; Stegemann, T.: Interactive Relationship Discovery via the Semantic Web. In Proceedings of the 7th Extended Semantic Web Conference (ESWC2010), Heraklion, Greece. 30 May - 03 Jun, 2010.
2. SPARQL Specification - http://www.w3.org/TR/rdf-SPARQL-query/
3. Polleres A., Scharffe F., Schindlauer R. SPARQL++ for mapping between RDF vocabularies. In Proceedings of the 6th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE2007), Vilamoura, Algarve, Portugal. Nov 27-29, 2007.
4. Volz R., Oberle D., Studer R. Views for light-weight Web ontologies. In Proceedings of the 2003 ACM symposium on Applied computing (SAC2003). Melbourne, Florida. March 09-12, 2003.
5. Karvounarakis G., Alexaki S., Christophides V., Plexousakis D., Scholl M. RQL: a declarative query language for RDF. In Proceedings of the 11th international conference on World Wide Web.. Honolulu, Hawaii, USA. May 07-11, 2002
6. Chen C., Wu Z., Wang H., Mao Y. RDF/RDFS-based Relational Database Integration. In Proceedings of the 22nd International Conference on Data Engineering (ICDE2006), April 03-07. 2006
7. A. Magkanaraki, V. Tannen, V. Christophides, and D. Plexousakis. Viewing the semantic web through RVL lenses. Web Semant. 1, 4 (Oct. 2004), 359-375.
8. J. Euzenat, A. Polleres, and F. Scharffe. Processing ontology alignments with SPARQL. In Proceedings of Proceedings of the 2008 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS2008). Washington, DC, USA, March 04-07, 2008.
9. Araujo S., Schwabe D. Explorator: a tool for exploring RDF data through direct manipulation. In Proceedings of Linked Data on the Web (LODW2009) workshop at WWW2009. Madrid, Spain. April 20, 2009.
10. Volz, J., Bizer C., Gaedke, M., and Kobilarov, G. Silk – A Link Discovery Framework for the Web of Data. In Proceedings of Linked Data on the Web (LODW2009) workshop at WWW2009. Madrid, Spain. April 20, 2009.
11. Oren E.,Delbru R., Gerke S., Haller A., Decker S. ActiveRDF: ObjectOriented Semantic Web Programming. Digital Enterprise Research Institute National University of Ireland, Galway Galway, Ireland. 2007
12. Hassanpour, S., O'Connor, M.J., Das, A.K. A Software Tool for Visualizing, Managing and Eliciting SWRL Rules. In Proceedings of the 7th Extended Semantic Web Conference (ESWC10). Heraklion, Greece. May 30 - Jun 03, 2010.