Discovering and Analysing Ontological Models from Big RDF Data

Carlos R. Rivero[1], Inma Hernández[2], David Ruiz[3], and Rafael Corchuelo[3]

[1] University of Idaho, USA
crivero@uidaho.edu
[2] Universidad Autónoma de Chile, Chile
ichernandez@uautonoma.cl
[3] University of Sevilla, Spain
{druiz, corchu}@us.es

Abstract. The Web of Data, which comprises web sources that provide their data in RDF, is gaining popularity day after day. Ontological models over RDF data are shared and developed with the consensus of one or more communities. In this context, there usually exist more than one ontological model to understand RDF data; therefore, there might be a gap between the models and the data, which is not negligible in practice. In this article, we present a technique to automatically discover ontological models from raw RDF data. It relies on a set of SPARQL 1.1 structural queries that are generic and independent from the RDF data. The output of our technique is a model that is derived from these data and includes the types and properties, subtypes, domains and ranges of properties and subproperties. Our experiments with millions of triples prove that our technique is suitable to deal with Big RDF Data. As far as we know, this is the first technique to discover such ontological models in the context of RDF data and the Web of Data.

Keywords: Ontological models, Web of Data, RDF, SPARQL 1.1

1 Introduction

The goal of the Semantic Web is to endow the current Web with metadata, i.e., to evolve it into a Web of Data [24, 31]. Currently, there is an increasing popularity of the Web of Data, chiefly in the context of Linked Open Data, which is a successful initiative that consists of a number of principles to publish, connect, and query data in the Web [4]. Sources that belong to the Web of Data focus on several domains, such as government, life sciences, geography, media, libraries, or scholarly publications [16]. These sources offer their data using the RDF language, and they can be queried using the SPARQL query language [1].

The goal of the Web of Data is to use the Web as a large database to answer structured queries from users [24]. One of the most important research challenges is to cope with scalability, i.e., processing data at Web scale, which is usually referred to as Big Data [6]. Additionally, sources in the Web of Data are growing steadily, e.g., in the context of Linked Open Data, there were roughly 12 such sources in 2007 and, as of the time of writing this article, there exist 226 sources [20]. Therefore, the problem of Big Data increases due to this large amount of sources.

Ontological models are used to model RDF data, and they comprise types, data properties, and object properties, each of which is identified by a URI [1]. These models are shared and developed with the consensus of one or more communities [29], which define a number of inherent constraints over the models, such as subtypes, the domains and/or ranges of a property, or subproperties.

In traditional information systems, developers first need to create a data model according to the user requirements, which is later populated. Contrarily, in information systems in the Web of Data, data can exist without an explicit model; even more, several models may exist for the same set of data. Therefore, in this context, we cannot usually rely on existing ontological models to understand RDF data since there might be a gap between the models and the data, i.e., the data and the model are usually devised in isolation, without taking each other into account [13]. Furthermore, RDF data may not satisfy a particular ontological model related to these data, which is mandatory to perform a number of tasks, such as data integration [21], data exchange [30], data warehousing [14], or ontology evolution [11]. As a conclusion, current techniques to perform information integration can leverage from the discovering of conceptual models [28].

We present two examples that are not negligible in practice of this gap between ontological models and RDF data (see [3] for additional discussions on this topic), namely:

– Languages to represent ontological models provide constructs to express user-defined constraints that are local, i.e., a user or a community can add them to adapt existing models to local requirements [8]. For instance, the ontological model of DBpedia 3.7 [5], which is a community effort to make the data stored at Wikipedia accessible using the Linked Open Data principles, defines a property called *almaMater* that has type *Person* as domain, and type *EducationalInstitution* as range. It is not difficult to find out that this property has also types *City* and *Country* as ranges in the RDF data. As a conclusion, there are cases in which RDF data may not be modelled according to existing ontological models, i.e., the data may not satisfy the constraints of the models.
– Some ontological models simply define vocabularies with very few constraints. Therefore, it is expected that users of these ontological models apply them in different ways [27]. For instance, the ontological model of DBpedia 3.7 defines a property called *similar* that has neither domain nor range. In the RDF data, we observe that this property has two different behaviours: one in which type *Holiday* is the domain and range of the property, and another one in which type *Place* is the domain and range of the property. As a conclusion, different communities may generate a variety of RDF data that rely on the same ontological models with disparate constraints.

In this article, we present a technique to automatically discover ontological models from raw RDF data that aims to solve the gap between the models and the data. Our technique assumes that the model of a set of RDF data is not known a priori, which is a common situation in practice in the context of the Web of Data. To perform this discovery, we rely on a set of SPARQL 1.1 structural queries that are generic and independent from the RDF data, i.e., they can be applied to discover an ontological model in any set of RDF data.

The output of our technique is a model that includes the types and properties, subtypes, domains and ranges of properties and subproperties. Our technique is suitable to deal with Big RDF Data since our experiments focus on millions of RDF triples from DBpedia 3.5 to 3.8. To the best of our knowledge, this is the first technique to discover such ontological models in the context of RDF data and the Web of Data. In our experimental results, we analyse the quality of the discovered models by comparing them with the original models that are provided by the DBpedia website.

We presented a preliminary 10-page version of these results in [26]; in this version, we present the complete set of SPARQL queries and the algorithm to discover the models from raw RDF data, which, together with our experimental results over the different versions of DBpedia, and the analysis of the discovered models, constitute the major differences.

This article is organised as follows: Section 2 describes the related work; Section 3 presents our technique to discover ontological models from RDF data that relies on a set of SPARQL 1.1 queries; Section 4 describes several experiments to discover the ontological models behind the RDF data of DBpedia 3.5 to 3.8; finally, Section 5 recaps on our main conclusions.

2 Related work

Research efforts on the automatic discovery of data models have focused on the Deep Web, in which web pages are automatically produced by filling web templates using the data of a back-end database [15]. In the context of the Web of Data, the most related work to ours is [3], which consists of a framework to define rules to study whether or not a given RDF dataset conforms to a given ontological model; the framework includes a formal language to express these rules. The main difference with respect to our approach is that we are able to discover a conceptual model without the intervention of the user.

There are a number of proposals in the literature that aim to discover types from instances, i.e., a particular instance has a particular type. The vast majority of these proposals discover different types in web sites by clustering web pages of the same type [7, 12, 18, 22]. Mecca et al. [22] developed an algorithm for clustering search results of web sites by type that discovers the optimal number of words to classify a web page. Blanco et al. [7] devised a technique to automate the clustering of web pages by type in large web sites. The authors do not rely on the content of web pages, but only on the URLs. Hernández et al. [18] devised a technique similar in spirit to [7], but using a smaller subset of web pages as the training set to automatically cluster the web pages. As a conclusion, these proposals are only able to discover types and no relationships amongst them, such as data properties, object properties, or subtypes; furthermore, since the previous techniques rely on clustering, the types thus discovered are anonymous. Giovanni et al. [12] aimed to automatically discover the untyped entities that DBpedia comprises, and they proposed two techniques based on induction and abduction.

There are a few proposals from the field of web information extraction [9, 32, 36] that are able to infer a model from the semi-structured data that is rendered in a web page. Kayed and Chang [19] presented a technique that compares a number of web pages in order to find common patterns that delimit the information to be extracted; Arasu and Garcia-Molina [2] presented a technique that is similar in spirit, but differs in the way that the shared patterns are computed; Crescenzi and Mecca [10] presented a technique that tries to identify similar subtrees in a DOM tree and then aligns them in order to discover where the information of interest is and their model; Sleiman and Corchuelo [33] presented the latest proposal in this field, which is also based on finding shared patterns as a means to identify the information of interest in a web page. All of the previous techniques can discover a hierarchical model that basically focuses on identifying the main class/es in a web page and then their data properties, which are anonymous since the techniques are totally unsupervised. That is, en expert must identify the semantics of the classes and data properties thus discovered, and no subclasses, subproperties, or object properties are identified.

Other proposals allow to discover complex data models that include types, properties, domains and ranges. These proposals are not fully-automated since they require the intervention of a user. Tao et al. [35] presented a proposal that automatically infers a data model by means of a form, and they deal with any kind of form, not necessarily HTML forms. In this case, the user is responsible for handcrafting these forms; unfortunately, this approach is not appealing since integration costs may be increased if the user has to intervene [23]. Furthermore, this proposal is not able to deal with subtypes.

Hernández et al. [17] devised a proposal that deals with discovering the data model behind a web site. This proposal takes a set of URL patterns that describe the types in a web site as input. Its goal is to discover properties amongst the different types that, in addition to the URL patterns of types, form a data model. The main drawback of this proposal is that it requires the intervention of the user: the final data model comprises a number of anonymous properties and the user is responsible for naming them, which may increase integration costs. In addition, this proposal is not able to discover data properties or subtypes.

Finally, Su et al. [34] developed a fully-automated proposal that discovers an ontological model that is based on the HTML forms of a web site, and the HTML results of issuing queries by means of these forms. In this case, there is no intervention of a user to discover the final ontological model, which is performed by means of a number of matchings amongst the HTML results and the HTML forms. To build the final model, the authors apply nine heuristics, such as "if a matching is unique, a new attribute is created", or "if the matching is n:1, $n+1$

Prefix	**URI**
rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
xsd	`http://www.w3.org/2001/XMLSchema#`
dbpo	`http://dbpedia.org/ontology/`
dbpd	`http://dbpedia.org/resource/`
sch	`http://schema.org/`
foaf	`http://xmlns.com/foaf/0.1/`

Table 1: Prefixes used throughout the article.

attributes are created". The main drawback of this proposal is that it does not discover subtypes or the name of the properties, i.e., the final model is more a nested-relational model than an ontological model. Note that a nested-relational model is defined by means of a tree that comprises a number of nodes, which may be nested and have a number of attributes, and it is also possible to specify referential constraints that relate these attributes [25].

Our survey of the literature reveals that the existing proposals focus on analysing the conformity of an ontological model using user-defined rules, or learning these models from web documents. To the best of our knowledge, there is not a single proposal to learn an ontological model from RDF data.

3 Discovering ontological models

In this section, we describe our proposal. First, we provide a few preliminary definitions; then, we present the SPARQL 1.1 queries on which it relies; finally, we describe our algorithm. We use a running example that is based on DBpedia 3.5, which comprises $16,114,546$ triples.

Preliminaries RDF data comprise triples of two kinds, namely: types and properties. A triple comprises three elements: the subject, the predicate, and the object, respectively. Both subjects and predicates are URIs, and objects may be URIs or literals. In the rest of this article, we use a number of prefixes that are presented in Table 1. A type triple relates a URI with a particular type by means of a type predicate, e.g., $(dbpd{:}Clint\_Eastwood, rdf{:}type, dbpo{:}Actor)$ states that Clint Eastwood is an actor. A data property triple relates a URI with a literal by means of a specific property, for example, $(dbpd{:}Clint\_Eastwood, dbpo{:}birthDate,$ "$1930-05-31$"^^$xsd{:}date)$ is a triple that states that the birth date of Clint Eastwood is May 31, 1930, which has $xsd{:}date$ type. An object property triple relates two URIs by means of a property, for example, $(dbpd{:}Dirty\_Harry, dbpo{:}starring,$ $dbpd{:}Clint\_Eastwood)$ is a triple stating that film Dirty Harry is starred by Clint Eastwood.

Discovery queries Initially, our technique discovers the types in the raw RDF data. These types may be class types, such as *dbpo*:*Person* or *dbpo*:*Actor*, or basic data types, such as *xsd*:*date* or *xsd*:*integer*. To discover the class types, we issue the following SPARQL query over the data:

Q_1:
```
SELECT DISTINCT ?t
WHERE {
    ?s rdf:type ?t . }
```

This query projects the class types of all instances without repetition. Furthermore, to discover the basic data types, we use the following query:

Q_2:
```
SELECT DISTINCT (XSDType(?o) AS ?t)
WHERE {
    ?s ?p ?o .
    FILTER (isLiteral(?o)) }
```

In this case, we use a custom function devised by us called *XSDType* that takes a literal as input and it outputs its basic data type. We also ensure that object $?o$ is actually a literal by means of the *isLiteral* function, which is standard in SPARQL.

In the second step, our technique discovers properties from the input RDF data, such as *dbpo*:*birthDate*, *dbpo*:*starring*, or *dbpo*:*director*, by means of a SPARQL query that projects the predicates that relate all triples without repetition as follows:

Q_3:
```
SELECT DISTINCT ?p
WHERE {
    ?s ?p ?o . }
```

In the third step, we discover subtypes amongst the previously discovered class types. To perform this, we iterate two times over the whole set of class types, so, for each pair of types t_1 and t_2, assuming that $t_1 \neq t_2$, we have that t_1 is subtype of t_2 if each instance of type t_1 is also an instance of type t_2. Note that we do not compute the subtypes of basic data types. An example is that *dbpo*:*MusicalWork* is subtype of *dbpo*: *Work*. We use the following SPARQL query template:

Q_4:
```
ASK {
    ?s rdf:type t1 .
    FILTER NOT EXISTS {
        ?s rdf:type t2 . } }
```

This template takes two parameters that are denoted as t_1 and t_2. Furthermore, it is important to notice that we use the negation of the previous query template, i.e., t_1 is subtype of t_2 if the template returns false.

The fourth step consists of computing the domain and ranges of the properties. We distinguish between data properties, which have a class type as domain and a basic data type as range, and object properties, which have two class types as domain and range. For instance, the domain of *dbpo*:*starring* is *dbpo*:*Work* and the range is *dbpo*:*Person*; the domain of *dbpo*:*birthDate* is *dbpo*:*Person* and the range is *xsd*:*date*. To compute the domain and ranges of the data properties we use the following query:

Q_5:
```
SELECT DISTINCT ?d ?p (XSDType(?o) AS ?r)
WHERE {
    ?s  ?p ?o .
    ?s  rdf:type  ?d .
    FILTER (isLiteral(?o)) }
```

This query uses our custom function *XSDType* to extract the basic data type from $?o$ and also ensures that it is a literal. We issue a similar SPARQL query to discover the domain and ranges of the object properties as follows:

Q_6:
```
SELECT DISTINCT ?d ?p ?r
WHERE {
    ?s  ?p ?o .
    ?s  rdf:type  ?d .
    ?o  rdf:type  ?r . }
```

Queries Q_5 and Q_6 may seem redundant with respect to query Q_3, however, it is important to notice that the latter deals with discovering all of the properties that are present in the RDF data, whereas queries Q_5 and Q_6 focus on discovering domains and ranges. Therefore, if we avoid query Q_3, then we cannot discover properties that do not have an explicit domain and/or range.

In the final step, our technique aims to discover subproperties of the previously discovered object properties, i.e., for every triple of a given property, we may find another triple of the corresponding subproperty. We perform similar computations as in the template to discover subtypes. An example is that *dbpo*:*musicalBand* is subproperty of *dbpo*:*musicalArtist*. We use the following SPARQL query template in which we use its negation to discover the subproperties, i.e., p_1 is subproperty of p_2 if the template returns false:

Q_7:
```
ASK {
    ?s p1 ?o .
```

```
algorithm discoverModel
input     Data: Set of RDF Triples
output    Types: Set of Type; Properties: Set of Property

- Step 1 (types): Discover class and basic data types
$Types := DiscoverTypes(Data)$

- Step 2 (properties): Discover data and object properties
$Properties := DiscoverProperties(Data)$

- Step 3 (subtypes): Discover sub class types
for each $t_1 \in Types$
    if ($t_1.IsClassType()$)
        for each $t_2 \in Types$
            if ($t_2.IsClassType() \wedge t_1 \neq t_2 \wedge IsSubtype(t_1, t_2)$)
                $t_1.AddSubtype(t_2)$

- Step 4 (domains and ranges): Discover domains and ranges of
  properties
$DiscoverDomainAndRanges(Data, Properties)$

- Step 5 (subproperties): Discover object sub properties
for each $p_1 \in Properties$
    if ($p_1.IsObjectProperty()$)
        for each $p_2 \in Properties$
            if ($p_2.IsObjectProperty() \wedge p_1 \neq p_2 \wedge IsSubproperty(p_1, p_2)$)
                $p_1.AddSubproperty(p_2)$
```

Figure 1: Algorithm to discover ontological models from RDF data.

```
FILTER NOT EXISTS {
    ?s p2 ?o . } }
```

Algorithm Figure 1 presents our algorithm to discover ontological models. It takes a set of RDF triples as input, and it outputs the types and properties of the discovered ontological model. A *Type* is specialised into either a class and a basic data type. A *Property* is specialised into either a data or an object property. Furthermore, we use the following ancillary functions: *IsClassType* returns *true* if the type is a class type; *AddSubtype* adds a new subtype to the current type; *IsObjectProperty* returns *true* if the property is an object property; and *AddSubproperty* adds a new subproperty to the current property.

In the first step, we call algorithm *DiscoverTypes* that executes queries Q_1 and Q_2 to discover the class and basic data types in the RDF data. The second step consists of calling algorithm *DiscoverProperties*, which executes query Q_3 to discover the properties. Note that, at this time, we do not distinguish between data and object properties, since this depends on the ranges of the properties, i.e., whether or not the ranges are basic data types. Additionally, *Properties* in this algorithm is an input/output parameter. In the third step, we perform a cartesian product amongst all of the previously discovered class types and, if the *IsSubtype* algorithm returns *true* by executing query Q_4, this entails that t_1 is a subtype of t_2.

The fourth step of our algorithm consists of discovering the domain and ranges of the properties, which is performed by the *DiscoverDomainAndRanges* algorithm, which executes queries Q_5 and Q_6 to compute every domain and range types of a given property. Finally, the fifth step discovers the subproperties in a similar way as we have already discovered the subtypes in the RDF data using query Q_7.

4 Experimental results

In this section, we present our experimental results. First, we describe the experimentation environment; then, we present our results; finally, we evaluate them.

Environment We implemented our technique using Java 1.6 and OWLIM Lite 4.2, which comprises an RDF store and a SPARQL query engine. Furthermore, we ran our experiments on a virtual computer that was equipped with a four-threaded Intel Xeon 3.00 GHz CPU and 16 GB RAM, running on Windows Server 2008 (64-bits), JRE 1.6.0.

We performed our experiments to discover the ontological models behind the RDF data of DBpedia 3.5 to 3.8. To give an overall idea of the size of each RDF dataset, the total number of triples of the different versions of DBpedia are as follows: DBpedia 3.5 comprises 16,114,546 triples; DBpedia 3.5.1 comprises 16,653,097 triples; DBpedia 3.6 comprises 20,169,651 triples; DBpedia 3.7 comprises 26,988,080 triples; and DBpedia 3.8 comprises 34,035,463 triples.

Results Table 2 shows our experimental results, in which the first column of the table stands for the different steps of our technique (note that there is an initial step called "Loading" that stands for the time taken to load the corresponding datasets into OWLIM Lite); the second column deals with the total number of constraints that the original ontological model of DBpedia comprises; the third column deals with the total number of constraints that we have discovered; and the fourth column shows the time in seconds taken by our technique to compute each step. Furthermore, each row corresponds to one of the steps of our technique.

Evaluation After discovering these ontological models, it is mandatory to evaluate their quality. To perform this, we leverage the ontological models provided by DBpedia.org, which aim to provide a generic ontological model of the data that each version of DBpedia comprises. We evaluated that, in general, these

Step	Orig.	Constr.	Time
Loading	–	–	255 s.
Types	255	242	51 s.
Properties	1,274	1,079	7 s.
Subtypes	27	608	26 s.
Dom. & ranges	757	44,789	433 s.
Subproperties	0	46	89 s.

(a) DBpedia 3.5.

Step	Orig.	Constr.	Time
Loading	–	–	224 s.
Types	257	241	54 s.
Properties	1,276	1,088	7 s.
Subtypes	27	604	31 s.
Dom. & ranges	772	45,319	475 s.
Subproperties	0	46	92 s.

(b) DBpedia 3.5.1.

Step	Orig.	Constr.	Time
Loading	–	–	267 s.
Types	272	251	72 s.
Properties	1,335	1,185	9 s.
Subtypes	27	628	29 s.
Dom. & ranges	856	54,258	1,149 s.
Subproperties	0	42	96 s.

(c) DBpedia 3.6.

Step	Orig.	Constr.	Time
Loading	–	–	379 s.
Types	319	327	89 s.
Properties	1,643	1,379	13 s.
Subtypes	84	1,308	83 s.
Dom. & ranges	974	141,154	1,119 s.
Subproperties	0	37	142 s.

(d) DBpedia 3.7.

Step	Orig.	Constr.	Time
Loading	–	–	539 s.
Types	359	349	113 s.
Properties	1,775	1,401	17 s.
Subtypes	22	1,521	123 s.
Dom. & ranges	1,007	170,976	3,346 s.
Subproperties	0	46	156 s.

(e) DBpedia 3.8.

Table 2: Summary of results of discovering ontological models behind RDF data.

original ontological models are contained in our discovered models; however, there are some differences as Table 2 shows, which are as follows:

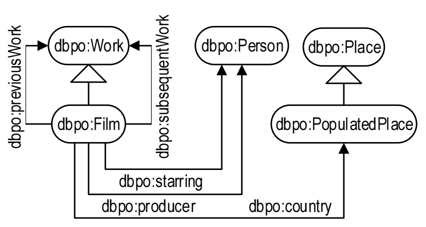
1. Types: our technique discovers less types than the original ontological models since, in our case, we focus just on those types that appear in the raw RDF data. The original ontological models comprise a number of types that are not used in the current datasets but are intended to be used in the future. To mention a few examples of these unused types, we highlight *dbpo*:*Celebrity*, *dbpo*:*VicePrimeMinister*, or *dbpo*:*BoxingLeague* in DB-

pedia 3.5, and *dbpo*:*Pope*, *dbpo*:*Royalty*, or *dbpo*:*VolleyballPlayer* in DBpedia 3.7. Furthermore, there are some types that our technique discovers but belong to external ontological models, since this is consider a good practice in the context of the Web of Data [4]. For instance, DBpedia 3.5 uses types from the OpenGIS ontology, whereas DBpedia 3.8 uses types from the Dublin Core, FOAF, SKOS, GeoRSS, Basic Geo, and BIBO external ontological models.

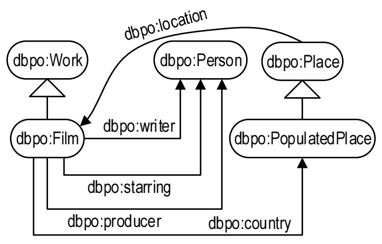
2. Subtypes: our technique discovers more subtypes than the original ontological models, which comprise just a few *rdfs*:*subClassOf* constraints, e.g., DBpedia 3.7 comprises 84 subtype constraints whilst the rest of versions do not exceed 27 subtype constraints. The number of subtype constraints discovered by our technique ranges from 604 to 1,521.
3. Properties: as was the case for types, our technique only discovers types that are currently used in the datasets. Some examples of these unused properties are *dbpo*:*youthYears*, *dbpo*:*throws*, or *dbpo*:*accessDate* in DBpedia 3.5, and *dbpo*:*youthYears*, *dbpo*:*richestCountry*, or *dbpo*:*carNumber* in DBpedia 3.7. Furthermore, our technique discovers additional properties from external ontological models. For instance, DBpedia 3.5 uses properties from the FOAF and Basic Geo ontologies, whereas DBpedia 3.8 uses properties from the Dublin Core, FOAF, SKOS, GeoRSS, and Basic Geo external ontological models.
4. Domain and ranges: a common feature of the original ontological models of DBpedia is that they minimise the domains and ranges of the properties. This is due to the fact that their goal is to provide a general vocabulary instead of a complete ontological model. This is the reason why our technique discovers thousands of domains and ranges. For instance, the original ontological model of DBpedia 3.8 comprises 1,007 domains and ranges whilst our technique discovers 170,976.
5. Subproperties: without an exception, no original ontological model of DBpedia defines subproperty constraints. Our technique is able to discover them from the raw RDF data, e.g., DBpedia 3.8 comprises 46 subproperty constraints.

To provide an overall idea of the ontological models discovered by our technique, we present a part of them in Figure 2. To compute them, we focused on a central type, *dbpo*:*Film*, and analysed its supertypes and properties, i.e., when this type is domain or range of a given property. Since the number of supertypes and properties may be huge, we used a threshold that consisted of retrieving just those types and properties that are more frequent in the raw RDF data, i.e., the number of triples that contain those types and properties must be sufficiently large. Our results also show the evolution that the data in DBpedia has undergone.

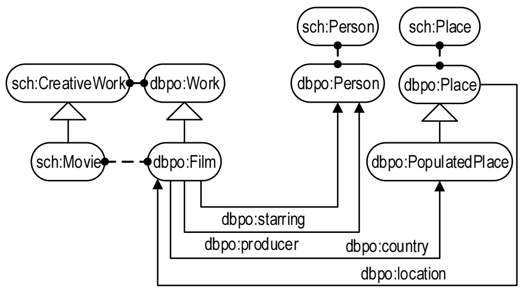
We checked that the discovered ontological models for DBpedia 3.5 and 3.5.1 were the same (see Figure 2a). The resulting model comprises five types as follows: *dbpo*:*Work*, *dbpo*:*Film*, *dbpo*:*Person*, *dbpo*:*Place*, and *dbpo*:*PopulatedPlace*; the following two subtype constraints: *dbpo*:*Film* is subtype of types *dbpo*:*Work*,



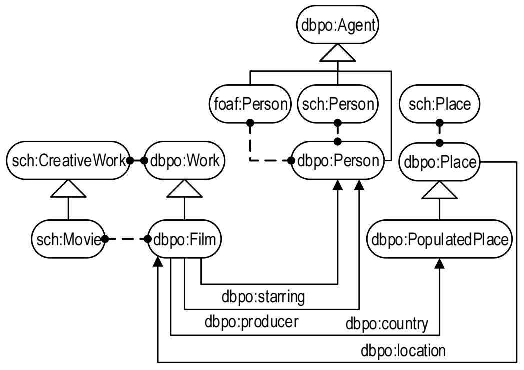
(a) DBpedia 3.5 and 3.5.1.



(b) DBpedia 3.6.



(c) DBpedia 3.7.



(d) DBpedia 3.8.

Figure 2: A part of the ontological models that result from our experiments.

and *dbpo*:*PopulatedPlace* is subtype of *dbpo*:*Place*. Additionally, we discover the following five properties, namely: *dbpo*:*starring*, *dbpo*:*producer*, *dbpo*:*country*, *dbpo*:*previousWork*, and *dbpo*:*subsequentWork*. We represent domains and ranges as arrows, e.g., the domain of *dbpo*:*previousWork* is *dbpo*:*Film* and its range is *dbpo*:*Work*. In DBpedia 3.6 (see Figure 2b), the number of triples that comprise properties *dbpo*:*previousWork* and *dbpo*:*subsequentWork* decrease and, therefore, they disappear from our discovered model. Additionally, two new properties appear, namely: *dbpo*:*writer* and *dbpo*:*location*. In DBpedia 3.7 (see Figure 2c), we discover new types that correspond to external ontological models, i.e., *sch*:*Movie*, *sch*:*creativeWork*, *sch*:*Person*, and *sch*:*Place*, which are equivalent to *dbpo*:*Film*, *dbpo*:*Work*, *dbpo*:*Person*, and *dbpo*:*Place*, respectively. Furthermore, property *dbpo*:*writer* disappears from the ontological model. Finally, in DBpedia 3.8 (see Figure 2d), two new types appear, namely: *dbpo*:*Agent* and *foaf*:*Person*.

5 Conclusions

In the context of the Web of Data, there exists a gap between existing ontological models and RDF data due to the following reasons: 1) RDF data does not have to satisfy the constraints in any existing ontological model; 2) different communities may generate a variety of RDF data that rely on the same ontological models with disparate constraints. This gap is not negligible and may hinder the practical application of RDF data and ontological models in other tasks, such as data integration, data exchange, data warehousing, or ontology evolution. To solve this gap, we present a technique to discover ontological models from raw RDF data that relies on a set of SPARQL 1.1 structural queries. The output of our technique is an ontological model that includes types and properties, subtypes, domains and ranges of properties and subproperties.

According to our evaluation results, our technique is able to discover models from Big RDF Data, since the evaluated datasets comprise millions of triples. We have focused on DBpedia to perform our experiments, and our results show that our discovered ontological models contain the original models provided by DBpedia.org, which is an assessment of the quality of our models. We also present a part of some of these ontological models to show the evolution of the RDF data that DBpedia comprises.

To the best of our knowledge, our technique is the first to appear in the literature to tackle these problems. Definitely, it will help perform integration tasks in the context of Big RDF Data because the techniques in this field rely on the constraints provided by the ontological models to be integrated.

Acknowledgements

Supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes (grants TIN2007-64119, P07-TIC-2602, P08-TIC-4100, TIN2010-21744, TIN2010-09809-E, TIN2010-10811-E, and TIN2010-09988-E).

References

[1] G. Antoniou and F. van Harmelen. *A Semantic Web Primer*. The MIT Press, 2008.

[2] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD Conference*, pages 337–348, 2003.

[3] M. Arenas, G. Díaz, A. Fokoue, A. Kementsietsidis, and K. Srinivas. A principled approach to bridging the gap between graph data and their schemas. *PVLDB*, 7(8):601–612, 2014.

[4] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data: The story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[5] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia: A crystallization point for the Web of Data. *J. Web Sem.*, 7(3):154–165, 2009.

[6] C. Bizer, P. Boncz, M. L. Brodie, and O. Erling. The meaningful use of Big Data: Four perspectives - four challenges. *SIGMOD Record*, 40(4):56–60, 2011.

[7] L. Blanco, N. N. Dalvi, and A. Machanavajjhala. Highly efficient algorithms for structural clustering of large websites. In *WWW*, pages 437–446, 2011.

[8] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. Contextualizing ontologies. *J. Web Sem.*, 1(4):325–343, 2004.

[9] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan. A survey of web information extraction systems. *IEEE Trans. Knowl. Data Eng.*, 18(10): 1411–1428, 2006.

[10] V. Crescenzi and G. Mecca. Automatic information extraction from large websites. *J. ACM*, 51(5):731–779, 2004.

[11] G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, and G. Antoniou. Ontology change: Classification and survey. *Knowledge Eng. Review*, 23(2):117–152, 2008.

[12] A. Giovanni, A. Gangemi, V. Presutti, and P. Ciancarini. Type inference through the analysis of wikipedia links. In *LDOW*, 2012.

[13] B. Glimm, A. Hogan, M. Krötzsch, and A. Polleres. OWL: Yet to arrive on the Web of Data? In *LDOW*, 2012.

[14] O. Glorio, J.-N. Mazón, I. Garrigós, and J. Trujillo. A personalization process for spatial data warehouse development. *Decision Support Systems*, 52(4):884–898, 2012.

[15] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the Deep Web. *Commun. ACM*, 50(5):94–101, 2007.

[16] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space.* Morgan & Claypool, 2011.

[17] I. Hernández, C. R. Rivero, D. Ruiz, and R. Corchuelo. Discovering conceptual models behind web sites using URL patterns. In *ER*, pages 610–613, 2012.

[18] I. Hernández, C. R. Rivero, D. Ruiz, and R. Corchuelo. CALA: An unsupervised URL-based web page classification system. *Knowl.-Based Syst.*, 57:168–180, 2014.

[19] M. Kayed and C.-H. Chang. FiVaTech: Page-level web data extraction from template pages. *IEEE Trans. Knowl. Data Eng.*, 22(2):249–263, 2010.

[20] LOD Cloud. Linked Open Data cloud, Apr. 2012. URL `http://thedatahub.org/group/lodcloud`.

[21] K. Makris, N. Gioldasis, N. Bikakis, and S. Christodoulakis. SPARQL-RW: Transparent query access over mapped RDF data sources. In *EDBT*, 2012.

[22] G. Mecca, S. Raunich, and A. Pappalardo. A new algorithm for clustering search results. *Data Knowl. Eng.*, 62(3):504–522, 2007.

[23] M. Petropoulos, A. Deutsch, Y. Papakonstantinou, and Y. Katsis. Exporting and interactively querying web service-accessed sources: The CLIDE system. *ACM Trans. Database Syst.*, 32(4):22, 2007.

[24] A. Polleres and D. Huynh. Special issue: The Web of Data. *J. Web Sem.*, 7(3):135, 2009.

[25] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating web data. In *VLDB*, pages 598–609, 2002.

[26] C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. Towards discovering ontological models from big RDF data. In *ER Workshops*, pages 131–140, 2012.

[27] C. R. Rivero, A. Schultz, C. Bizer, and D. Ruiz. Benchmarking the performance of Linked Data translation systems. In *LDOW*, 2012.

[28] C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. MostoDE: A tool to exchange data amongst semantic-web ontologies. *Journal of Systems and Software*, 86(6):1517–1529, 2013.

[29] C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. Exchanging data amongst Linked Data applications. *Knowl. Inf. Syst.*, 37(3):693–729, 2013.

[30] C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. Benchmarking data exchange among semantic-web ontologies. *IEEE Trans. Knowl. Data Eng.*, 25(9):1997–2009, 2013.

[31] N. Shadbolt, T. Berners-Lee, and W. Hall. The Semantic Web revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[32] H. A. Sleiman and R. Corchuelo. A survey on region extractors from web documents. *IEEE Trans. Knowl. Data Eng.*, 25(9):1960–1981, 2013.

[33] H. A. Sleiman and R. Corchuelo. Trinity: On using trinary trees for unsupervised web data extraction. *IEEE Trans. Knowl. Data Eng.*, page Preprint, 2014.

[34] W. Su, J. Wang, and F. H. Lochovsky. ODE: Ontology-assisted data extraction. *ACM Trans. Database Syst.*, 34(2):12, 2009.

[35] C. Tao, D. W. Embley, and S. W. Liddle. FOCIH: Form-based ontology creation and information harvesting. In *ER*, pages 346–359, 2009.

[36] J. Turmo, A. Ageno, and N. Català. Adaptive information extraction. *ACM Comput. Surv.*, 38(2), 2006.