Reconciling Ontologies with the Web of Data

Ziawasch Abedjan
Hasso-Plattner-Institut
Potsdam, Germany
ziawasch.abedjan@hpi.uni-potsdam.de

Johannes Lorey
Hasso-Plattner-Institut
Potsdam, Germany
johannes.lorey@hpi.uni-potsdam.de

Felix Naumann
Hasso-Plattner-Institut
Potsdam, Germany
naumann@hpi.uni-potsdam.de

ABSTRACT

To integrate Linked Open Data, which originates from various and heterogeneous sources, the use of well-defined ontologies is essential. However, oftentimes the utilization of these ontologies by data publishers differs from the intended application envisioned by ontology engineers. This may lead to unspecified properties being used ad-hoc as predicates in RDF triples or it may result in infrequent usage of specified properties. These mismatches impede the goals and propagation of the Web of Data as data consumers face difficulties when trying to discover and integrate domain-specific information. In this work, we identify and classify common misusage patterns by employing frequency analysis and rule mining. Based on this analysis, we introduce an algorithm to propose suggestions for a data-driven ontology re-engineering workflow, which we evaluate on two large-scale RDF datasets.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications

Keywords

ontology re-engineering, LOD, RDF, association rules

1. INTRODUCTION

The concept of Linked Open Data (LOD) enables information providers to create and share linkable data across the Web. Besides the increasing number of LOD sources, there also exists a set of rules for publishing Linked Data [6]. Still, consuming and integrating LOD necessitates a thorough analysis and study of the data sources, as individual data providers have different understandings or knowledge of useful vocabulary definitions. Here, reusable knowledge bases and ontologies facilitate comprehending and integrating multiple data sources. These ontologies provide metadata to define the domains and ranges of properties of resources or taxonomical relationship between these resources. In our work, we analyze the differences between specification and usage of such vocabularies and offer a data-driven approach to refine existing ontologies.

We observed a significant distinction between well-defined ontologies and the common understanding of LOD. While LOD in general is designed to be easily extensible so that facts about resources in one dataset can be added ad-hoc and independently of other sources, ontologies usually define structural information for a number of data sources and are therefore less flexible. Hence, integration of LOD, ontology discovery and matching pose major challenges for the pervasion of the Web of Data. While there are best practices for publishing Linked Open Data using established ontologies [6], our analysis shows that due to various reasons certain "misusage" patterns occur frequently. This misuse partly stems from the fact that ontology definitions may be either too specific or too generic. Thus, custom namespace-specific properties often are added to an ontology concept when need arises. It is likely that some of these properties are redundant, as data providers are unaware of one another's additions.

Addressing such quality issues of an ontology can be considered a form of schema analysis. Here, a schema defines the set of properties whose domain is a specific class within the ontology. Hence, redesign of ontologies requires analysis and mining of the underlying data. The following real-world example illustrates the discrepancy between ontology specification and usage. For brevity and readability, we use a number of prefix abbreviations when denoting RDF resources. These abbreviations are defined in Listing 1.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix :     <http://dbpedia.org/ontology/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix dbp:  <http://dbpedia.org/property/> .
@prefix rdf:  <http://www.w3.org/1999/02/
              22-rdf-syntax-ns#> .
```

Listing 1: RDF prefix abbreviations used in this work

1.1 Motivational Example

For a manifestation of the problem addressed by this work consider Listing 2 which shows an excerpt of the `:Settlement` class definition. If a geo-location data provider decides to publish her data using this specification, she might be confused about how to set proper values for some of the properties. The properties in lines 2 and 3 are intuitively applicable as predicates[1] to all instances of `:Settlement`,

[1]In accordance with the well-known RDF triple model (subject-predicate-object), in this paper we refer to instance-level properties as *predicates*.

whereas others seem only useful for a strict subset of instances, such as `:scottishName` and `:distanceToEdinburgh` in lines 4 and 5.

```
1 :Settlement            rdfs:subClassOf  :PopulatedPlace .
2 :winterTemperature     rdfs:domain      :Settlement .
3 :summerTemperature     rdfs:domain      :Settlement .
4 :scottishName          rdfs:domain      :Settlement .
5 :distanceToEdinburgh   rdfs:domain      :Settlement .
```

Listing 2: Excerpt of the `:Settlement` specification.

However, none of the properties of this class (or any of its parent classes `:PopulatedPlace`, `:Place`, and `owl:Thing`) model the latitude and longitude degrees of a settlement although these are set for many instances of `:Settlement` in DBpedia, e.g., via the `dbp:latd` and `dbp:longd` predicates, respectively. As `dbp:latd` and `dbp:longd` are not explicitly specified for `:Settlement`, ambiguous predicates with similar values also occur in the instance data, e.g., `dbp:latDeg` and `dbp:longDeg`.

Overall, for a more intuitive vocabulary definition some of the properties (e.g., `:scottishName`) could be removed, delegated to a more suitable subclass of `:Settlement` if available, or marked as optional if supported by the ontology language. Additionally, some of the predicates that are already set for a large number of instances of `:Settlement` (e.g., `dbp:latd`) may be included in the class definition.

In general, the denoted discrepancies may cause confusion which ontology and which classes therein to adopt when publishing RDF data. Using vocabularies that are not intended for certain resources or unwarranted extensions to existing ontologies limit machine readability and thus impedes the benefits of Linked Data. The goal of our work is to identify recurring misuse of ontology definitions and help overcome these problems by offering re-engineering suggestions.

1.2 Contributions

To facilitate the process of such ontology re-engineering we contribute:

1. The definition of two general ontology misusage cases.
2. An automated data-driven approach that supports ontology adjustment.
3. A thorough evaluation that shows the usefulness and feasibility of the approaches on popular RDF datasets.

2. PROBLEM STATEMENT

Based on an existing ontology, we identify two typical cases where the specification differs from usage patterns: *overspecification* and *underspecification*. We refer to a certain class as being overspecified, if one or more properties are declared for this class by the ontology, but are rarely (if ever) used for real-world data, e.g., `:scottishName` for `:Settlement`. We deem a class to be underspecified, when in real-world data certain properties are used frequently even though they are not specified by the vocabulary, e.g., `dbp:latd` for `:Settlement`.

Note that a class can simultaneously be overspecified and underspecified (with regard to different properties). Both overspecification and underspecification may stem from data providers being unaware of the specifics of the ontology they employ for the information they publish. Thus, they may neglect certain properties (although suitable) or introduce new ones (even though these may be semantically equivalent to existing ones). However, in our work we focus on ontology engineers by offering usage information as well as re-engineering suggestions to them.

2.1 Overspecification

Over time, RDF ontologies may grow by introducing new class and property definitions. Especially cross-domain ontologies, such as DBpedia and Yago, have evolved extensively since their first specification. However, revising existing class definitions is sometimes neglected during this evolutionary process. This leads to several problems:

- Data providers cannot set proper values for the defined properties, e.g., a `:scottishName` for a non-Scottish `:Settlement`.
- There are multiple semantically equivalent properties defined for a class, e.g., `:occupation` and `:profession` for `:Person`.
- Subclasses have been added and (inherited) properties are now used only in combination with these, e.g., `:philosophicalSchool` used exclusively with `:Philosopher`, but defined for parent class `:Person`.

In most cases for which we identified overspecification, a solution is to remove properties from the class concerned. However, sometimes these properties are still valid for certain subclasses of this class. Thus, before removing a property from the ontology it is essential to verify whether it is used in instances of any subclass. If so, the property needs to be *pushed down* to the appropriate subclass.

2.2 Underspecification

The flexible RDF data model allows the ad-hoc assignment of predicates to instances of a certain class. However, if this predicate is suitable for a large number of instances, it might prove beneficial to add it as a property to the corresponding class definition. Doing so aids data integration and ensures that other data providers are aware of this property. The reasons for underspecification include:

- The class definition lacks certain properties that are commonplace in instance data, e.g., `dbp:latd` for `:Settlement` or `:genre` for `:Band`.
- A property is defined for certain subclasses whereas it covers additional instances of their common parent class, e.g., `:numberOfStudents` is defined for `:University`, but also used for instances of other subclasses of `:EducationalInstitution` such as `:College`.
- Data providers extend the ontology independently of one another, e.g., 150 properties with various namespaces have domain `foaf:Person` in the BTC 2011 crawl[2] whereas FOAF only defines 16 such properties.

To overcome underspecification, properties may be added to a class definition. In our approach, we ensure that our suggestions do not create unnecessary redundancy. If both a class and several of its subclasses are underspecified with respect to a certain property, this property is not added to these subclasses, but *pushed up* to the common parent class instead. Finally, we combine both approaches for under- and overspecification so that the results are consistent and without loss of information.

[2]`http://km.aifb.kit.edu/projects/btc-2011/`

3. DATA-DRIVEN ONTOLOGY RE-ENGINEERING PROCESS

To determine data-driven ontology re-engineering options, one could align specific classes and associated instance data to identify predicates to be removed from or added to the class definition, respectively. However, this would have two limitations: (1) Some properties might be removed instead of being allocated to a more suitable subclass, and (2) Some properties might be added to a class and its subclasses independently without regarding inheritance relationships. Therefore, we propose a holistic approach that processes all classes of a given ontology by also considering class hierarchies.

Our approach to generate suggestions for predicate removal and inclusions consists of the following steps:

1. Identify typed entities in the data (declaring `rdf:type`) and retrieve relevant ontological information.
2. Generate removal suggestions by detecting rarely used properties.
3. Generate inclusion suggestions by mining predicates

Depending on the dataset, the first task is more or less straightforward: For instance, in the case of DBpedia, explicit instance mapping information as well as a well-defined ontology is available. The last two steps describe our principal approach for generating re-engineering suggestions. In the following, we present the intuition of our approach and the procedure of making ontology adjustment proposals.

3.1 Predicate Analysis and Mining

After having identified the typed entities in a dataset, we construct frequent sets of predicates for instances of each type. A frequent set of predicates is a set of predicates that co-occur for a significant number of instances and hold *minimum support* s [2]. A set of predicates X holds support s if $s\%$ of instances of a specific type in the dataset involve all the predicates of X. Moreover, we can detect dependencies between frequent sets of predicates as association rules: A positive association rule $X \rightarrow Y$ states that the predicates in Y (*consequent*) depend on the predicates in X (*condition*). The confidence *conf* of such an association rule is the conditional probability $P(Y|X)$. Denoting the support of a set X as *supp*(X), *conf*$(X \rightarrow Y)$ is computed as *supp*$(X \cup Y)/$*supp*(X). Relevant rules are those that hold some *minimum confidence* c.

3.2 Automated Ontology Adjustment

Given a dataset with typed instances and a corresponding ontology, we apply frequency and association rule analysis to identify over- and underspecification. Next, we describe the two consecutive steps for the ontology adjustment: Generating property removal and property inclusion suggestions.

3.2.1 Property Removal Suggestions

Identifying cases of overspecification in a class definition is straightforward: Given a minimum support threshold of s, each property that does not hold s in the given data constitutes an overspecification of the current class and should therefore be suggested for removal from this class. Thus, for each property defined for the class its distinct occurrences as predicate for all entities of the given class are counted and the total is compared against the minimum support. The set of all properties that do not meet the minimum support are marked for removal from the specific class definition.

3.2.2 Property Inclusion Suggestions

Having marked removal suggestions, we now determine predicates that are used frequently for instances of a specific class but are not defined as properties for the class itself or any of its parent classes in the ontology. We also consider association rules and propose only those predicates to a class that are highly correlated with already (validly) defined properties of this class. This is the first constraint we define for our suggestions:

Constraint 1 If for two predicates p' and p there is a rule $p' \rightarrow p$ with minimum support s and minimum confidence c, and *domain*$(p') = \mathcal{C}$ and *domain*$(p) \neq \mathcal{C}$ and *domain*$(p) \neq \mathcal{P}_i$ for all of $\mathcal{C}$'s strict parent classes $\mathcal{P}_1, \mathcal{P}_2, \ldots$, predicate p should be proposed as a property for $\mathcal{C}$.

It may happen that under Constraint 1 a predicate is proposed for a class $\mathcal{C}$ as well as for one or more of the subclasses of $\mathcal{C}$. For example, the predicate `:anthem` may hold enough support among the instances of type `:Place` and may also be associated with properties defined for `:Place`. However, it might be the case that most or all those instances of type `:Place` with `:anthem` are in fact instances of the more specific type `:Country` (which is a subclass of `:Place`). Then, the algorithm proposes the property to be added to the more specific class. On the other hand, when a predicate such as `:populationDensity` is proposed for multiple subclasses of `:PopulatedPlace`, such as `:Country`, `:City`, or `:Continent`, as well as for `:PopulatedPlace` itself on behalf of Const. 1 it is more appropriate to propose the predicate for the more general class `:PopulatedPlace`. To avoid redundant suggestions and specifying properties as fittingly as possible we define the following two constraints.

Constraint 2 If Constraint 1 holds for predicate p and class $\mathcal{C}$ as well as for p and exactly one of $\mathcal{C}$'s strict subclasses $\mathcal{S}_i$, property p should be proposed for the subclass $\mathcal{S}_i$ instead of $\mathcal{C}$.

Constraint 3 If Constraint 1 holds for predicate p and class $\mathcal{C}$ as well as for p and more than one of $\mathcal{C}$'s strict subclasses $\mathcal{S}_1, \mathcal{S}_2, \ldots$, where at least two of these subclasses are not subclasses of one other, p should be proposed only for $\mathcal{C}$.

Algorithm 1 illustrates the workflow for generating property inclusion suggestions after the identification of removal candidates. The input of the algorithm includes the complete set of instance triples (i.e., triples about resources for which the type is known) denoted as *triples* as well as the set of classes *classes* of the ontology to be adjusted. For each class c it is indicated whether properties have been defined specifically for the class or inherited from parent classes and whether the properties are removal candidates. The result of the algorithm are enriched class definitions containing inclusion and removal suggestions for each class.

The algorithm traverses the classes in the given ontological hierarchy breadth-first, beginning with the leaves and moving up towards the root. It may happen that one predicate is an inclusion candidate for a class as well as for some of its subclasses. To decide for which class specifically to propose such a property it is necessary that all subclasses have been analyzed before the parent class. This is ensured by the reverse topological sorting of *classes* in line 1. Afterwards, for each c in *classes* rule mining is executed on all triples in *triples* belonging to instances of type c based on given minimum support and confidence.

For each rule discovered in the process, Constraint 1 is checked in line 8. If it holds, the rule's consequence is added to the set of inclusion *suggestions*. As we apply the same

```
Algorithm 1: Property Inclusion Suggestion Algorithm
Data: classes : class definitions (including original and
      cleaned schema)
Data: triples : instance triples for all available classes
Data: minSupp, minConf : minimum support and
      confidence values
1  classes.topologicalSortAscending ();
2  foreach $c \in$ classes do
3      rules ← genRules (minSupp, minConf, T, c);
4      schema ← c.cleanProperties;
5      inheritedSchema ← c.cleanInheritedProperties;
6      candidates ← ∅;
7      foreach $r \in$ rules do
8          if r.condition $\in$ schema $\wedge$
9          r.consequence $\notin$ inheritedSchema then
10             suggestions.add (r.consequence);
11     foreach $s \in$ suggestions do
12         subclasses = c.getSubclassesWith (s);
13         if |subclasses| = 1 then
14             suggestions.remove (s);
15         if |subclasses| > 1 then
16             foreach subclass $\in$ subclasses do
17                 subclass.removeProperty (s);
18     c.addSuggestions (suggestions);
```

minimum support here as in the property removal suggestion step, no property is proposed for the same class for which it has been marked for removal earlier. Note that *schema* and *inheritedSchema* contain only those properties that have not been marked when identifying property removal suggestions, i.e., every property that has been marked for removal in the previous step may appear as an inclusion candidate for some of the subclasses of *c*.

Having scanned *rules* for appropriate *candidates*, the next step is to test the current class *c* and *suggestions* for the Constraints 2 and 3. Thus, for each suggestions *s* it is checked whether there are *subclasses* of the current class *c* that include *s* in their schema or inclusion suggestions list. According to Constraints 2 and 3, the algorithm either removes *s* from all definitions in subclasses (either specified or proposed) of *c* or from the current *suggestions* of *c*.

Finally, the remaining *suggestions* are added to the current class *c*. These inclusion suggestions can be used to extend the original class definitions.

4. CASE STUDY

To assess the quality of our ontology re-engineering proposals, we applied our approach on the DBpedia dataset and corresponding ontology, and evaluated the resulting suggestions. The DBpedia ontology is manually generated using Wikipedia infobox templates [3], and evolves over time as the infobox templates are changed.

We performed our evaluation on the DBpedia 3.6 and (at the time of writing most current) DBpedia 3.7 datasets along with the respective DBpedia ontology versions utilized for the data. In Tab. 1, we list the total amounts of (unique) triples and subjects in the individual datasets as well as the total amounts of unique classes and properties in the ontologies.

DBpedia	#Triples	#Subjects	#Classes	#Properties
3.6	17,518,364	1,638,746	272	1,335
3.7	13,794,426	1,827,474	319	1,643

Table 1: Total of unique values in DBpedia 3.6 and 3.7

We identified 503 removal suggestions in the DBpedia 3.6 ontology and 622 removal suggestions in the DBpedia 3.7 ontology, all with support $\leq 1\%$. Table 2 shows sample results of overspecification in DBpedia 3.7. Some of the removal suggestions can be moved to a more suitable subclass, Tab. 3 presents such an alternative allocation of the bottom five properties in Tab. 2. Clearly, the proposed assignment is more appropriate than the actual specification, as the suggestion support is in orders of magnitude higher than the original support. While for some properties the suggestion support still seems low (e.g., `:countySeat`), instances of these classes still include more than 90% of the occurrences of the properties among the originally assigned parent class.

Property	Class	Support
:scottishName	:Settlement	0.000%
:distanceToEdinburgh	:Settlement	0.021%
:waistSize	:Person	0.013%
:philosophicalSchool	:Person	0.202%
:countySeat	:PopulatedPlace	0.831%
:anthem	:PopulatedPlace	0.147%
:depth	:Place	0.723%
:numberOfGraduateStudents	:EducationalInstitution	0.300%

Table 2: Overspecified properties for DBpedia 3.7

Suggested Property	Class	Support
:philosophicalSchool	:Philosopher	76.225%
:countySeat	:AdministrativeRegion	9.470%
:anthem	:Country	18.730%
:depth	:Lake	33.698%
:numberOfGraduateStudents	:College	93.590%

Table 3: Pushed properties for DBpedia 3.7

For underspecification, we applied Algorithm 1 (*minSupp*: 1%, *minConf*: 70%) on DBpedia 3.6 and DBpedia 3.7. The choice of the thresholds corresponds to traditional association rule mining settings and proved reasonable in our scenario. A higher *minSupp* threshold results in more removal suggestions and less inclusion suggestions. A higher *minConf* threshold reduces only the number of inclusion suggestions. We evaluated all suggested properties for the classes of the ontology manually by labeling whether their assignment to a specific class was *useful*, *not useful*, or *undecided*. An assignment was *undecided* when labelers could not determine the appropriateness of the assignment.

Overall, the majority of the suggestions have been labeled as useful, including the pushed properties mentioned in Tab. 3. Some of our proposed properties for the DBpedia 3.6 dataset have indeed been included in the DBpedia 3.7 ontology, such as **`:numberOfEpisodes`** and **`:numberOfSeasons`** for the class **`:TelevisionShow`**, thus further validating our results. Table 4 illustrates the amount and quality of class property suggestions for DBpedia 3.6 and 3.7.

DBpedia	Total	Useful	Not Useful	Undecided
3.6	283	234 (83%)	15	34
3.7	317	268 (85%)	31	18

Table 4: Suggestion quality for DBpedia 3.6 and 3.7

Suggestions marked as undecided are those for which we could not decide whether they enhance the class definition or not. This was often the case, when a similar or synonymous property had already been defined for a class in the ontology (e.g., for `:Person`, `:Person/weight` is specified, `:weight` is suggested). For DBpedia 3.6 we discovered that of the 283 suggested properties 206 had no specified domain, 17 were pushed up or down from a sub- or superclass, 12 had completely different domains, and 48 had synonymous specified properties. Of the 317 suggested properties in DBpedia 3.7, 225 had no domain, 18 were pushed, 17 had different domains, and 57 had synonymous specified properties.

Note that we considered only properties from the DBpedia ontology namespace, but properties from other namespaces might also be valid suggestions. Overall, our evaluation shows that data-driven suggestions lead to useful results. The runtime of our algorithm is in the order of a few hours (including the time for creating the transaction database for association rule mining) even for large datasets and mostly depends on the parameters used for rule mining.

5. RELATED WORK

We present related work on data mining in the Semantic Web as well as work on analyzing ontology usage and guiding ontology engineering.

There are only a few approaches and projects that apply data mining on the Semantic Web. Mostly, they are in the fields of inductive logic programming and approaches that make use of the description logic of a knowledge base [7, 8]. Those approaches concentrate on mining answer-sets of queries towards a knowledge base. An association rules based approach for mining the Semantic Web is proposed by Nebot et al. [10], where a SPARQL endpoint allows the user to define targets of mining in any desired graph context. Their approach also focuses on entities and ignores predicates. Our approach corresponds to one of the six mining configurations outlined in [1] that also includes predicate mining. ProLOD is a tool for profiling Linked Open Data, which includes association rule mining on predicates for the purpose of schema analysis [4]. In this work, we present a concrete application exploiting the mined predicates.

Several works in the field on ontology engineering aim at establishing and enriching ontology specifications by using machine learning techniques [5]. The authors of [9] present a semi-automatic approach for cross-domain ontology learning. Similarly, in [12] machine learning methods are employed to refine the definition of the Wikipedia infobox-class ontology. In contrast to these works, our approach allows incremental re-engineering of an existing ontology based on provided instance data without any training data or third party data sources, such as WordNet.

The authors of [11] present a schema induction approach based on association rules to recreate axioms of the DBpedia ontology. As their approach generates an ontology axiom for every generated rule, redundant or conflicting axioms are created, too. Our approach, differs from their work in two major aspects: First, we do not want to recreate an ontology but to improve a given ontology.Second, we have defined constraints that allow our approach to create sound suggestions for improving an ontology definition by avoiding the generation of redundant and conflicting suggestions.

6. CONCLUSIONS AND FUTURE WORK

We presented an approach to evaluate the usage of an ontology in real-world RDF data and suggest possible modifications to the ontology's definition based on this evaluation. An ontology engineer can use these suggestions to revise the specification, thereby improving the intuitiveness of the ontology and aiding its propagation.

In this work, we identified and described two misconceptions of the vocabulary definition regarding its application to real-world data: over- and underspecification. To cope with these challenges, we presented an automated approach that facilitates ontology re-engineering by suggesting the removal or incorporation of properties for given ontology classes.

We evaluated our approach on the DBpedia dataset along with the DBpedia ontology. As illustrated in Sec. 4, there are significant mismatches between the intended use of certain well-known ontology specifications and how they are employed. These might by caused by illegitimate use of the vocabulary by data publishers, ontology evolution, or general design flaws in the vocabulary, amongst other things.

In future work we want to analyze negative correlations and association rules for ontology re-engineering. We believe that negative correlations are promising for the discovery of redundancies in form of synonyms and exclusive property clusters that can trigger the creation of subclasses.

7. REFERENCES

[1] Z. Abedjan and F. Naumann. Context and target configurations for mining RDF data. In *Proceedings of the International Workshop on Search and Mining Entity-Relationship Data (SMER)*, Glasgow, 2011.

[2] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 487–499, Santiago de Chile, Chile, 1994.

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

[4] C. Böhm, F. Naumann, Z. Abedjan, D. Fenz, T. Grütze, D. Hefenbrock, M. Pohl, and D. Sonnabend. Profiling linked open data with ProLOD. In *Proceedings of the International Workshop on New Trends in Information Integration (NTII)*, pages 175–178, 2010.

[5] P. Buitelaar and P. Cimiano, editors. *Ontology Learning and Population: Bridging the Gap between Text and Knowledge*, volume 167 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, 2008.

[6] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 1st edition, 2011.

[7] J. Józefowska, A. Lawrynowicz, and T. Lukaszewski. The role of semantics in mining frequent patterns from knowledge bases in description logics with rules. *Theory Pract. Log. Program.*, 10:251–289, 2010.

[8] F. A. Lisi and F. Esposito. Mining the semantic web: A logic-based methodology. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems (ISMIS)*, pages 102–111, Saratoga Springs, 2005.

[9] A. Maedche and S. Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16:72–79, 2001.

[10] V. Nebot and R. Berlanga. Mining association rules from semantic web data. In *Proceedings of the International Conference on Industrial Engineering and other Applications of Applied Intelligent Systems (IEA/AIE)*, volume 2, pages 504–513, Cordoba, Spain, 2010.

[11] J. Völker and M. Niepert. Statistical schema induction. In *Proceedings of the Extended Semantic Web Conference (ESWC)*, pages 124–138, Heraklion, Greece, 2011.

[12] F. Wu and D. S. Weld. Automatically refining the Wikipedia infobox ontology. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 635–644, Beijing, China, 2008.