Benchmarking Data Exchange among Semantic-Web Ontologies

Carlos R. Rivero, Inma Hernández, David Ruiz, and Rafael Corchuelo

Abstract—The increasing popularity of the Web of Data is motivating the need to integrate semantic-web ontologies. Data exchange is one integration approach that aims to populate a target ontology using data that come from one or more source ontologies. Currently, there exist a variety of systems that are suitable to perform data exchange among these ontologies; unfortunately, they have uneven performance, which makes it appealing assessing and ranking them from an empirical point of view. In the bibliography, there exist a number of benchmarks, but they cannot be applied to this context because they are not suitable for testing semantic-web ontologies or they do not focus on data exchange problems. In this paper, we present MostoBM, a benchmark for testing data exchange systems in the context of such ontologies. It provides a catalogue of three real-world and seven synthetic data exchange patterns, which can be instantiated into a variety of scenarios using some parameters. These scenarios help to analyze how the performance of data exchange systems evolves as the exchanging ontologies are scaled in structured and/or data. Finally, we provide an evaluation methodology to compare data exchange systems side by side and to make informed and statistically sound decisions regarding: 1) which data exchange system performs better; and 2) how the performance of a system is influenced by the parameters of our benchmark.

Index Terms—Performance and scalability, data exchange, Web of data

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 [27]. Currently, there is an increasing popularity of semantic-web ontologies, chiefly in the context of Linked Open Data, and they focus on a variety of domains, such as government, life sciences, geographic, media, or publications [17]. Semantic-web ontologies build on the so-called semantic-web technologies, i.e., RDF, RDFS, and OWL ontology languages for modeling structure and data, and the SPARQL query language to query them [3]. For the sake of brevity, we refer to semantic-web ontologies as ontologies.

Ideally, ontologies are shared data models that are developed with the consensus of one or more communities; unfortunately, reaching an agreement in a community is not a trivial task [4], [16]. Furthermore, new ontologies try to reuse existing ontologies as much as possible since it is considered a good practice; unfortunately, it is usual that existing ontologies cannot be completely reused, but require to be adapted [17]. Due to these facts, there exists a variety of heterogenous ontologies to publish data on the Web, and there is a need to integrate them [17].

In the bibliography, there are different approaches to address this problem, such as data exchange, data integration, model matching, or model evolution [6]. In this paper, we focus on data exchange [10], which aims to populate a target ontology using data that come from one or more source ontologies. In the bibliography, there are proposals that use ad hoc techniques, reasoners, or SPARQL query engines for performing data exchange. When using ad hoc techniques, data exchange is based on handcrafted pieces of software that transform source data into target data. When using reasoners, data exchange consists of reclassifying source instances into target instances by means of rules. Finally, when using SPARQL queries, data exchange is performed by executing a number of CONSTRUCT queries that extract data from the source ontologies, transform them, and load the results into the target ontology.

Currently, there exists a variety of systems that implement semantic-web technologies and are, thus, suitable to perform data exchange, for example, Sesame, OWLIM, Jena, TDB, Oracle, or Pellet to mention a few. Unfortunately, they have uneven performance [7], [14], [33], [36], which makes it appealing assessing and ranking them from an empirical point of view, since this helps make informed decisions about which the best system for a particular integration problem is.

A data exchange system is a piece of software that allows to exchange data. In our context, such a system comprises an RDF store, a reasoner, and a query engine. The systems that implement semantic-web technologies provide different services, for example, Pellet is a reasoner, ARQ is a query engine, Jena provides an RDF store and a reasoner, and Oracle or OWLIM provide an RDF store, a reasoner, and a query engine.

In the bibliography, there is a benchmark that focuses on data exchange systems for nested relational models [2]; however, it cannot be applied to our context due to a number of inherent differences between ontologies and nested relational models [25], [29]. In addition, there are several enchmarks to test systems that implement semantic-web

• *The authors are with the University of Sevilla, ETSI Informática, Avda. Reina Mercedes, s/n, Sevilla E-41012, Spain.*
E-mail: {carlosrivero, inmahernandez, druiz, corchu}@us.es.

technologies [7], [12], [14], [24], [31], [33], [36]. Unfortunately, these benchmarks have one or more of the following drawbacks: 1) they do not focus on data exchange problems, i.e., they do not provide source and target ontologies and mechanisms to exchange data; 2) they are domain-specific, i.e., they provide ontologies with a fixed structure in a particular domain, i.e., they only allow to tune the construction of synthetic data but not their structure; and 3) they focus on SELECT queries instead of the CONSTRUCT queries that are required to exchange data.

In this paper, we present MostoBM, a benchmark for testing data exchange systems in the context of ontologies and query engines. Our benchmark provides a catalogue of three real-world and seven synthetic data exchange patterns; seven parameters to construct scenarios that are instantiations of the patterns; and a publicly available tool[1] that facilitates the instantiation of the patterns and the gathering of data about the performance of systems. In addition, we provide an evaluation methodology that allows to compare data exchange systems side by side. To the best of our knowledge, this is the first such benchmark and evaluation methodology in the bibliography.

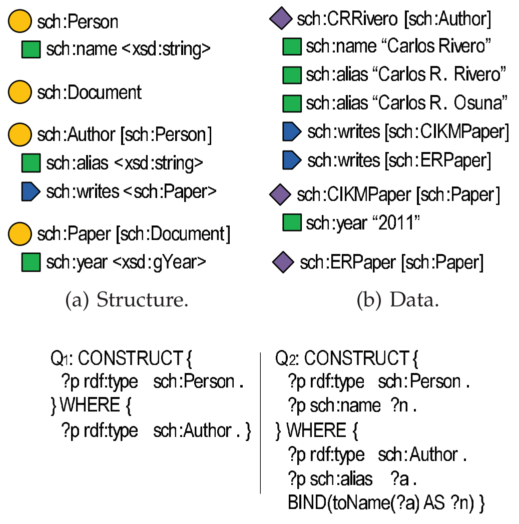
Regarding the catalogue, the three real-world patterns are relevant data exchange problems in the context of Linked Open Data, whereas the seven synthetic patterns are common integration problems that are based on current approaches in the ontology evolution context, and our experience regarding real-world information integration problems. This catalogue is not meant to be exhaustive: the patterns described in this paper are the starting point to a community effort that is expected to extend them.

Regarding the parameters, a benchmark should be scalable and the results that it produces should be deterministic and reproducible [15]. To fulfil these properties, our benchmark provides a number of parameters to construct scenarios, each of which is a three-element tuple (S, T, Q), where S is the source ontology, T is the target ontology, and Q is a set of SPARQL queries to perform data exchange. Our benchmark is based on SPARQL 1.1 and quite a complete subset of the OWL 2 Lite profile that leaves out only subproperty, zero-cardinality, general property, and intersection restrictions. The set of SPARQL queries of each scenario allows to exchange data with a 100 percent of effectiveness.

The parameters allow to scale the data of the source ontology for the real-world patterns, and to scale the structure of source and target ontologies, the data of the source ontology, and the SPARQL queries to perform data exchange for the synthetic patterns. Thanks to them, we can automatically construct the structure of a source and a target ontology with, for instance, a thousand classes, a dozen specialization levels, or the data of a source ontology with a million triples. The scaling of the patterns helps to analyze the performance of data exchange systems in future, when it is assumed that data exchange problems are going to increase their scale in structure and/or data.

Our evaluation methodology helps software engineers make informed and statistically sound decisions based on rankings that focus on: 1) which data exchange system performs better; and 2) how the performance of a system is influenced by the parameters of our benchmark.

1. http://www.tdg-seville.info/carlosrivero/MostoBM.



(a) Structure. (b) Data.

```
Q1: CONSTRUCT {
  ?p rdf:type  sch:Person .
} WHERE {
  ?p rdf:type  sch:Author . }
```

```
Q2: CONSTRUCT {
  ?p rdf:type  sch:Person .
  ?p sch:name  ?n .
} WHERE {
  ?p rdf:type  sch:Author .
  ?p sch:alias  ?a .
  BIND(toName(?a) AS ?n) }
```

(c) Queries.

Fig. 1. Examples of the structure and data of an ontology, and two SPARQL queries.

We presented a preliminary version of our benchmark in [28]; in this version, we extend our initial proposal with real-world patterns (see Section 3), with an evaluation methodology (see Section 6.1), and illustrate how to use it to make informed and statistically sound decisions (see Sections 6.2 and 6.3). An extended technical report that illustrates our benchmark and methodology extensively is also available at [30].

The remainder of the paper is organized as follows: Section 2 reports on preliminaries regarding semantic-web technologies and data exchange. In Sections 3 and 4, we present our catalogue of real-world and synthetic data exchange patterns, respectively. Section 5 describes the parameters of our benchmark. In Section 6, we describe our evaluation methodology and illustrate how to make informed and statistically sound decisions regarding a number of systems. In Section 7, we present the related work. Finally, Section 8 recaps on our main conclusions.

2 Preliminaries

The OWL ontology language allows to model both the structure and the data of an ontology. Regarding modeling structure, an OWL ontology comprises a set of entities identified by URIs, each of these entities may be a class, a data property, or an object property. Fig. 1a shows the structure of a sample ontology using a tree notation. In the figure, $sch{:}Author$ is a class that models an author, and we denote it as a circle. It is important to notice that, in this example, we use namespace $sch{:}$ as a prefix. A class can be specialized into other classes, for example, $sch{:}Author$ is a subclass of $sch{:}Person$, and we denote it as $sch{:}Author\ [sch{:}Person]$. An example of a data property is $sch{:}name$, which models the name of a person, and we denote it as a square. Data properties have a set of classes as domain and a basic XSD data type as range, for example, the domain of $sch{:}name$ is $\{sch{:}Person\}$, and we denote it by nesting $sch{:}name$ into *sch*:*Person*. The range of *sch*:*name* is *xsd*:*string*, and we denote it as *sch*:*name* <*xsd*:*string*>. An example of an object property is *sch*:*writes*, which models "an author writes a paper," and we denote it as a pentagon. Object properties have a set of classes as domain and range, for example, the domain of *sch*:*writes* is {*sch*:*Author*} and the range is {*sch*:*Paper*}.

Regarding data modeling, a class instance is identified by its own URI, and it may have a number of types (see Fig. 1b). We denote a class instance as a diamond, for example, *sch*:*CRRivero* is a class instance of type *sch*:*Author*. *sch*:*CRRivero* is also, implicitly, an instance of *sch*:*Person*, since *sch*:*Author* is a subclass of *sch*:*Person*; reasoners are used to make this knowledge explicit [21]. In addition, it is possible to have class instances of types that are not related by specialization, for example, assume that *sch*:*CRRivero* is the URL of a web page that provides the biography of a person, therefore, *sch*:*CRRivero* might have both types *sch*:*Person* and *sch*:*Document*.

A data property instance relates a class instance with a literal, and we denote it as a square, for example, *sch*:*name* relates *sch*:*CRRivero* with "*Carlos Rivero*." An object property instance relates two class instances, for example, *sch*:*writes* relates *sch*:*CRRivero* and *sch*:*CIKMPaper*. By default, data and object property instances have a minimal cardinality of zero, for example, there is no data property instance of *sch*:*year* relating *sch*:*ERPaper* and a year; however, there is a data property instance of *sch*:*year* relating *sch*:*CIKMPaper* and "2011". By default, data and object property instances may be multiple, for example, *sch*:*CRRivero* is related to "*Carlos R. Rivero*" and "*Carlos R. Osuna*" by data property *sch*:*alias*.

RDF, which is based on triples, is used to represent both the structure and data of an OWL ontology. A triple comprises three elements: a subject, a predicate, and an object. Sample triples are the following:

(*sch*:*Person*,	*rdf*:*type*,	*owl*:*Class*)
(*sch*:*Author*,	*rdf*:*type*,	*owl*:*Class*)
(*sch*:*Author*,	*rdfs*:*subClassOf*,	*sch*:*Person*)
(*sch*:*CIKMPaper*,	*rdf*:*type*,	*sch*:*Document*)
(*sch*:*CIKMPaper*,	*rdf*:*type*,	*sch*:*Paper*)
(*sch*:*CIKMPaper*,	*sch*:*year*,	"2011")

SPARQL queries are used to retrieve and construct triples from RDF stores. SPARQL provides four types of queries, namely: SELECT, CONSTRUCT, ASK, and DESCRIBE. They are based on triple patterns that are similar to triples, but allow to specify variables in the subject, predicate, and/or object, which are prefixed with a "?". In this paper, we focus on the CONSTRUCT type, since this type of queries allows to both retrieve and construct RDF data, which is required to perform data exchange by means of queries. Fig. 1c shows two examples of SPARQL queries: Q_1 retrieves instances of *sch*:*Author* (the WHERE clause) and reclassifies them as *sch*:*Person* (the CONSTRUCT clause); Q_2 is similar to Q_1 but also transforms the value of *sch*:*alias* into *sch*:*name* by means of a *toName* function using a *BIND* clause, which assigns the value to variable ?*n*.

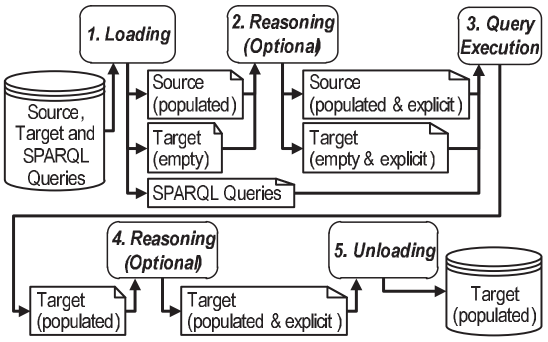


Fig. 2. Steps of data exchange.

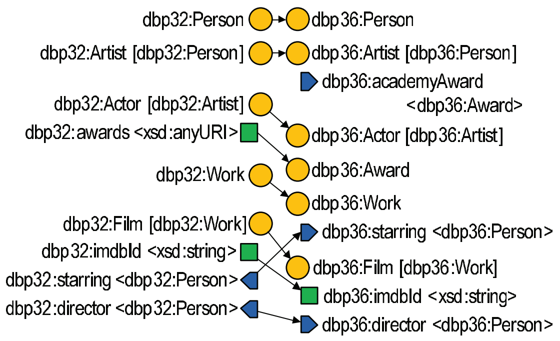
Performing data exchange between OWL ontologies using a SPARQL query engine comprises five steps (see Fig. 2), namely:

1. *Loading.* This step consists of loading the source and target ontologies and the set of SPARQL queries from a persistent storage into the appropriate internal data structures.
2. *Reasoning over source.* This step is optional and deals with making it explicit the knowledge in the source. This step is not required if the knowledge is already explicit in the source ontology, but it is a must in many cases since SPARQL does not deal with RDFS or OWL entailments.
3. *Query execution.* This step consists of executing a set of SPARQL queries over the source ontology to produce instances of the target ontology. The result of this step must be the same regardless of the order in which queries are executed.
4. *Reasoning over target.* This step is also optional and it deals with making it explicit the knowledge in the target ontology.
5. *Unloading.* This step deals with saving the target ontology to a persistent storage.

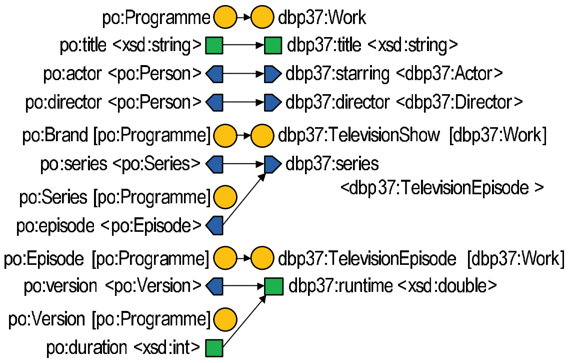
3 Real-World Patterns

Our benchmark provides three real-world data exchange patterns, each of which is instantiated into a variety of scenarios using a number of parameters (see Section 5). These patterns are illustrated in Fig. 3 and presented below. The source ontology is on the left side and the target is on the right side; the arrows represent correspondences between the entities of the ontologies. These correspondences are visual hints to help readers understand each real-world pattern [2]. We selected these three patterns to be part of our benchmark because they represent integration problems that are common in practice in the context of Linked Open Data. We use some prefixes to denote different ontologies, such as *dbp*32:, *dbp*36:, *srv*:, or *po*:.

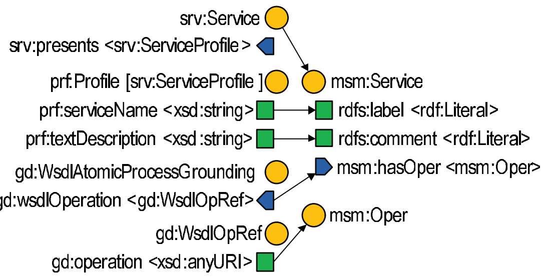
Evolution of an ontology. Usually, ontologies change in response to a certain need [11], including that the domain of interest has changed, the perspective under which the domain is viewed needs to be changed, or due to design flaws in the original ontology. In this context, the source ontology is the ontology before changes are applied and the target ontology is the ontology after changes are applied.


dbp32:Person
dbp36:Person
dbp32:Artist [dbp32:Person]
dbp36:Artist [dbp36:Person]
dbp36:academyAward <dbp36:Award>
dbp32:Actor [dbp32:Artist]
dbp32:awards <xsd:anyURI>
dbp36:Actor [dbp36:Artist]
dbp36:Award
dbp32:Work
dbp36:Work
dbp32:Film [dbp32:Work]
dbp36:starring <dbp36:Person>
dbp32:imdbId <xsd:string>
dbp36:Film [dbp36:Work]
dbp32:starring <dbp32:Person>
dbp36:imdbId <xsd:string>
dbp32:director <dbp32:Person>
dbp36:director <dbp36:Person>

(a) Evolving DBpedia.



(b) Adapting BBC Programmes to DBpedia.


srv:Service
srv:presents <srv:ServiceProfile>
prf:Profile [srv:ServiceProfile]
msm:Service
prf:serviceName <xsd:string>
rdfs:label <rdf:Literal>
prf:textDescription <xsd:string>
rdfs:comment <rdf:Literal>
gd:WsdlAtomicProcessGrounding
msm:hasOper <msm:Oper>
gd:wsdlOperation <gd:WsdlOpRef>
msm:Oper
gd:WsdlOpRef
gd:operation <xsd:anyURI>

(c) Publishing OWL-S services as Linked Open Data.

Fig. 3. Real-world patterns of our benchmark.

This pattern focuses on DBpedia [8], which is a community effort to annotate and make the data stored at Wikipedia accessible by means of an ontology. DBpedia comprises a number of different versions due to a number of changes in its conceptualization. When a new version of DBpedia is devised, the new ontology may be populated by performing data exchange from a previous version to the new one. In this pattern, we perform data exchange from a part of DBpedia 3.2 that focuses on artists, actors, directors, and films to DBpedia 3.6 (see Fig. 3a).

Vocabulary adaptation. It is not uncommon that two ontologies offer the same data structured according to different vocabularies. In this context, it is necessary to adapt the vocabulary of a source ontology to the vocabulary of a target ontology.

This pattern focuses on the BBC Programmes and DBpedia ontologies. The BBC [20] provides ontologies that adhere to the Linked Open Data principles to publicize the music and programmes they broadcast on both radio and television. In this pattern, which is shown in Fig. 3b, we perform data exchange from the Programmes Ontology 2009, which describes programmes including brands, series (seasons), and episodes, to a part of DBpedia 3.7 that models television shows and episodes.

Publication of Linked Open Data. Many existing ontologies do not adhere to the principles of the Linked Open Data initiative, and there is usually a need to transform them to comply with such principles.

This pattern focuses on publishing semantic-web services as Linked Open Data. OWL-S [19] is one of the main approaches for describing semantic-web services that defines an upper ontology in OWL. Minimal service model (MSM) [26] is a web service ontology that allows to publish web services as Linked Open Data. In this pattern, which is shown in Fig. 3c, we publish OWL-S 1.1 services as Linked Open Data by means of MSM 1.0.

4 Synthetic Patterns

A synthetic data exchange pattern represents a common and relevant integration problem. Our benchmark provides a catalogue of seven synthetic data exchange patterns; to design them, we have leveraged our experience on current approaches in the ontology evolution context (see Section 7), and our experience regarding real-world information integration problems. Each pattern represents an intention of change [11], i.e., each pattern represents a number of atomic changes that are applied to an ontology in response to certain needs. In addition, each synthetic pattern is instantiated into a variety of scenarios using a number of parameters (see Section 5). Below, we present our synthetic patterns, which are illustrated in Fig. 4. Note that src: and tgt: prefixes are used for the source and target ontologies, respectively.

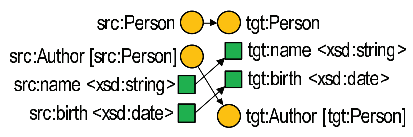
Lift properties. The intention of change is that the user wishes to extract common properties to a superclass in a hierarchy. Therefore, the data properties of a set of subclasses are moved to a common superclass. In the example, the $src{:}name$ and $src{:}birth$ data properties are lifted to $tgt{:}name$ and $tgt{:}birth$, respectively.

Sink properties. The intention of change is that the user wishes to narrow the domain of a number of properties. Therefore, the data properties of a superclass are moved to a number of subclasses. In the example, the $src{:}name$ and $src{:}birth$ data properties are sunk to $tgt{:}name$ and $tgt{:}birth$, respectively.

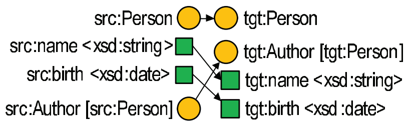
Extract subclasses. The intention of change is that the user wishes to specialize a class. Therefore, a source class is split into several subclasses and the domain of the target data properties is selected among the subclasses. In the example, every instance of $src{:}Person$ is transformed into an instance of $tgt{:}Person$.

Extract superclasses. The intention of change is that the user wishes to generalize a class. So, a class is split into several superclasses, and data properties are distributed among them. After exchanging data in this example, all target instances are of type $tgt{:}Author$, which are implicitly instances of $tgt{:}Person$, too.

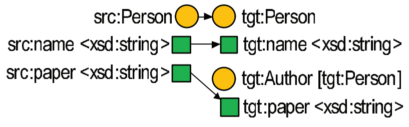
Extract related classes. The intention of change is that the user wishes to extract a number of classes building on a single class. Therefore, the data properties that have this



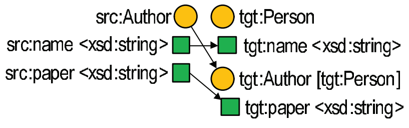
(a) Lift Properties.



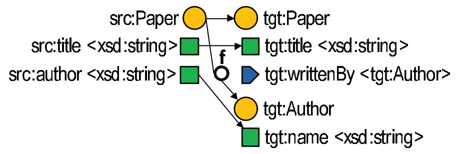
(b) Sink Properties.



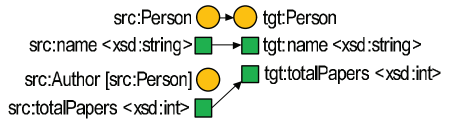
(c) Extract Subclasses.



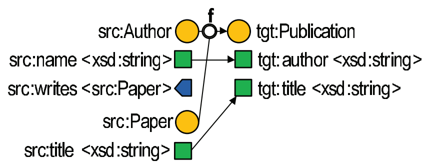
(d) Extract Superclasses.



(e) Extract Related Classes.



(f) Simplify Specialisation.



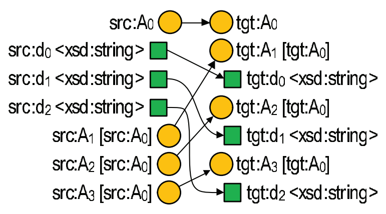
(g) Simplify Related Classes.

Fig. 4. Synthetic patterns of our benchmark.

single class as domain change their domains by the new classes, which are related to the original one by a number of object properties. In the example, the source class $src{:}Paper$ is split into two target classes called $tgt{:}Paper$ and $tgt{:}Author$ that are related by object property $tgt{:}writtenBy$. Instances of $tgt{:}Author$ are constructed by applying a user-defined function f to the instances of $src{:}Paper$.

Simplify specialization. The intention of change is that the user wishes to flat a hierarchy of classes. Thus, a set of specialized classes are flattened into a single class. In the example, $src{:}Person$ and $src{:}Author$, which specializes $src{:}Person$, are simplified to $tgt{:}Person$.

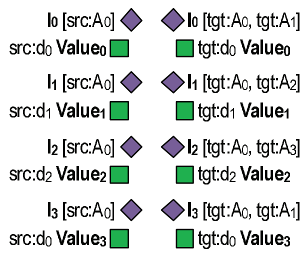
Simplify related classes. The intention of change is that the user wishes to join a set of classes that are related by object



(a) Sample structure.

```
Q1: CONSTRUCT {
  ?a rdf:type tgt:A0.
} WHERE {
  ?a rdf:type src:A0. }
Q2: CONSTRUCT {
  ?a rdf:type tgt:A1.
  ?a tgt:d0   ?d.
} WHERE {
  ?a rdf:type src:A0.
  ?a src:d0   ?d. }
```

(b) Sample queries.



(c) Sample data.

Fig. 5. Sample scenario.

properties. Therefore, several source classes are transformed into one class that aggregates them all. In the example, for every two instances of $src{:}Author$ and $src{:}Paper$ related by $src{:}writes$, a new instance of $tgt{:}Publication$ is constructed.

5 Parameters

Our benchmark takes a number of input parameters that allow to tune the data of the source ontology in the real-world patterns, and both structure and data of the source and/or the target ontologies in the synthetic patterns. Thanks to them, a user can instantiate multiple scenarios per pattern.

The structure parameters are the following:

- *Levels of classes ($L \in \mathbb{N}$):* number of relationships (specializations or object properties) among one class and the rest of the classes in the source or target ontologies. L allows to scale the structure of ontologies in depth.
- *Number of related classes ($C \in \mathbb{N}$):* number of classes related to each class by specialization or object properties. C allows to scale the structure of ontologies in breadth.
- *Number of data properties ($D \in \mathbb{N}$):* of the source and target ontologies.

L and C may be applied to both source and target ontologies, which is the case of the lift properties and sink properties patterns; to the target ontology only, i.e., the extract subclasses, extract superclasses, and extract related classes patterns; or to the source ontology only, i.e., the simplify specialization and simplify related classes patterns.

Fig. 5a shows a sample instantiation of the sink properties pattern in which $L = 1$, $C = 3$, and $D = 3$. Structure parameters can be tuned to construct realistic ontologies, for example, ontologies in the context of life sciences usually comprise a large set of classes that form a wide and/or deep taxonomy. For instance, the gene ontology [5], which represents gene and gene product attributes, comprises roughly 32,000 classes with a dozen specialization levels. Therefore, to construct ontologies that resemble the gene ontology, we need to specify the following parameters $L = 14$, $C = 2$. The total number of classes an ontology comprises is computed by the following formula: $\sum_{i=0}^{L} C^{i}$.

Structure parameters also have an effect on the SPARQL queries constructed by our benchmark, since they vary depending on the structure of the source and target ontologies. Fig. 5b shows two sample queries for the example scenario of the sink properties pattern: Q_1 is responsible for reclassifying A_0 in the source as A_1 in the target, and Q_2 is responsible for both reclassifying A_0 as A_1 in the target, and exchanging the value of $src{:}d_0$ into $tgt{:}d_0$.

Data parameters are used to scale the instances of the source ontology, which are the following:

- *Number of individuals* ($I \in \mathbb{N}\backslash\{0\}$)*:* number of instances of *owl:Thing* in the source ontology.
- *Number of types* ($I_T \in \mathbb{N}$)*:* number of types that each individual shall have.
- *Number of data properties* ($I_D \in \mathbb{N}$)*:* number of data property instances for which a given individual is the subject.
- *Number of object properties* ($I_O \in \mathbb{N}$)*:* number of object property instances for which a given individual is the subject.

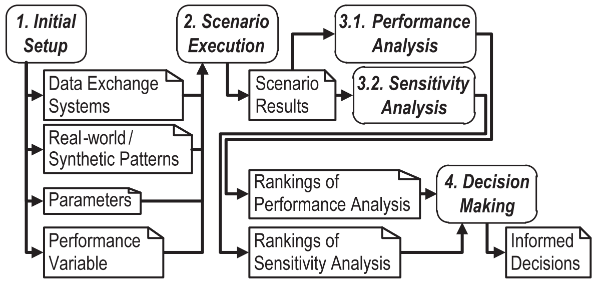
The actual types and instances may be randomly selected from the whole set of classes, data properties and object properties of the ontology to be populated. In our tool, we provide 44 statistical distributions to randomly select them, including uniform, normal, exponential, Zipf, Pareto, and empirical distributions, to mention a few. As a result, it is possible to use statistical distributions that model real-world ontologies.

Fig. 5c shows an example of the data constructed (left side) to populate the source ontology in Fig. 5a, in which the data parameters are the following: $I = 4$, $I_T = 1$, $I_D = 1$, and $I_O = 0$. Furthermore, this figure shows the target instances (right side) that result from performing data exchange with the SPARQL queries in Fig. 5b. We may compute the number of data triples that a populated ontology comprises by means of the following formula: $I(1 + I_T + I_D + I_O)$.

Data parameters can be tuned to populate ontologies that resemble realistic ones, for example, the SwetoDBLP [1] ontology models computer science publications. It comprises roughly 2 million triples of individuals of a single type, 4 million triples of data property instances, and 7 million triples of object property instances. To simulate this ontology, we have to set the following values: $I = 2\,10^6$, $I_T = 1$ (the individuals are of a single type), $I_D = 2$ (4 million triples of data property instances divided by 2 million triples of individuals), and $I_O = 4$ (7 million triples of object properties divided by 2 million triples of individuals).

6 Evaluation Methodology

Having a catalogue of data exchange patterns is not enough to evaluate a data exchange system in practice. It is necessary to rely on a disciplined evaluation methodology if we wish to make informed and statistically sound decisions. We have devised such a methodology and present its workflow in Fig. 6.



1. Initial Setup
Data Exchange Systems
Real-world / Synthetic Patterns
Parameters
Performance Variable
2. Scenario Execution
Scenario Results
3.1. Performance Analysis
3.2. Sensitivity Analysis
Rankings of Performance Analysis
Rankings of Sensitivity Analysis
4. Decision Making
Informed Decisions

Fig. 6. Workflow of the evaluation methodology.

We refer to performing data exchange over a scenario as executing that scenario. In the initial setup step, the user is responsible for selecting the data exchange systems to test, the patterns to test these systems, the values of parameters, and a variable to measure the performance of these systems. The scenario execution step consists of executing scenarios of each pattern using each system. After executing the scenarios, the results are used to compute which system performs better (performance analysis step) and to analyze the influence of parameters in their performance (sensitivity analysis step). Finally, in the decision making step, the user is responsible for making decisions based on the previous analysis.

We describe the details of the methodology in Section 6.1, and illustrate it in Sections 6.2 and 6.3.

6.1 Steps

6.1.1 Initial Setup

First, we need to select the study we are going to conduct. In this initial setup we are responsible for selecting the data exchange systems to test: $\mathbb{M} = \{m_1, \ldots, m_a\}$, $a \geq 2$; the data exchange patterns: $\mathbb{P} = \{p_1, \ldots, p_b\}$, $b \geq 1$; the set of values for the parameters, and a performance variable. Our methodology is devised to focus only on real-world or synthetic patterns, but not both at the same time, since they entail the study of different sets of parameters: in the real-world patterns, we only study data parameters since the structure of the source and target ontologies is fixed; in the synthetic patterns, we study both the structure and data parameters. Regarding structure parameters, we refer to their sets as P_L, P_C, and P_D; regarding the data parameters, we refer to their sets as P_I, P_{IT}, P_{ID}, and P_{IO}; in both cases, the subindex refers to the corresponding parameter.

A configuration is a tuple that comprises a value for each parameter; we denote the set of all possible configurations as $\mathbb{C}$. When dealing with real-world patterns, $\mathbb{C} = P_I \times P_{IT} \times P_{ID} \times P_{IO}$; when dealing with synthetic patterns, $\mathbb{C} = P_L \times P_C \times P_D \times P_I \times P_{IT} \times P_{ID} \times P_{IO}$. The combination of a pattern and a configuration forms a scenario, we denote the set of all possible scenarios as $\mathbb{X} = \mathbb{P} \times \mathbb{C}$. In addition, a setting is a combination of a system and a pattern, we denote the set of all possible settings as $\mathbb{S} = \mathbb{M} \times \mathbb{P}$. Consequently, the total number of scenarios for each pattern is $|\mathbb{C}|$.

Regarding the performance variable, there are two types of variables that we can select to measure the performance of systems: context-sensitive or context-insensitive. On the one hand, context-sensitive variables are affected by other processes that are executed in parallel in the computer, such as antiviruses, automatic updates, or backup tasks. So it is mandatory to execute the scenarios a number of times, usually 25-30 times, and compute the average of the performance variable, removing possible outliers. Some examples are the following: user time, I/O time, or memory faults. On the other hand, context-insensitive variables are not (notoriously) affected by other processes, so it is not needed to execute the scenarios more than once; some examples are the following: CPU time, number of target triples, or memory used.

6.1.2 Scenario Execution

For each system $m \in \mathbb{M}$ and each pattern $p \in \mathbb{P}$, we need to execute the scenarios related to p using m, so we need to execute the scenarios of all possible settings. Typical setups may involve the execution of a large number of scenarios, each of which may take hours or even days to complete; this makes it necessary to use the Monte Carlo method to select a subset of scenarios to execute randomly. The problem that remains is to determine which the size of this subset must be.

Our evaluation methodology comprises a sensitivity analysis for which we use a regression method. This imposes an additional constraint on the number of scenarios to execute, since it is known that the ratio of scenarios to parameters must be at least 10 [22]. As a conclusion, when dealing with real-world patterns, we should execute at least 40 scenarios, since they involve four data parameters, and when dealing with synthetic patterns, we should execute at least 70 scenarios, since they involve three structure and four data parameters. To provide a more precise figure on the number of scenarios to execute, we use an iterative method that relies on Cochran's formula [9], which is based on the variance of the performance variable.

In this iterative method, we introduce a threshold to limit the total number of scenarios to execute: $\mu \in \mathbb{R}, 0 < \mu \leq 1$, since, in the worst case, this number is equal to $|\mathbb{C}|$, which means that the variance being studied is so high that it is not possible to calculate an average at confidence level 95 percent with an estimated error less than 3 percent (the standard values of Cochran's formula). The same occurs with the number of iterations of this method, i.e., a large number of iterations of this method means that the variance being studied is high; therefore, we introduce another threshold regarding the number of iterations: $\delta \in \mathbb{N}, \delta > 0$. We describe the iterative method below:

1. For each setting s_i, we select 40 or 70 scenarios to execute using the Monte Carlo method.
2. We execute the selected scenarios.
3. We use Cochran's formula to compute the new number of scenarios to execute using the variance of the performance variable, which is computed from the scenarios previously executed.
4. Let e be the new number of scenarios to be executed, let k be the current number of iterations, and w the number of scenarios already executed; there are several alternatives: a) $e \leq w$ and $\delta \leq k$: we stop executing more scenarios and continue with the next step of the methodology; b) $\mu\ |\mathbb{C}| \geq e > w$ and $\delta \leq k$: we select $e - w$ scenarios using the Monte Carlo method and return to the second step of this method; c) $e > \mu\ |\mathbb{C}|$ or $\delta > k$: we discard s_i since it is not possible to make informed and statistically sound decisions about this setting.

At the end of this step, we have results of the performance variable for each setting, i.e., a set of tuples of the form (s, c, v), where s is a setting, c is a configuration, and v the corresponding value of the performance variable.

6.1.3 Performance and Sensitivity Analyses

Performance analysis. To compute which system performs better, we need a method to compare the values of the performance variable we gathered in the previous step. We can consider them as samples of an unknown random variable, which implies that we need to rely on statistical inference.

We apply Kruskal-Wallis's H test [18] to the results of each pattern in isolation, and to all of the results. The goal of this test is to determine if there are statistically significant differences among the performance variable for the systems under test. If the result is that there is no difference among the systems, it means that they all behave statistically identically with respect to the performance variable.

Otherwise, we use Wilcoxon's signed rank [18] test; if more than two systems are compared then we use Bonferroni's correction [23]. According to the correction, the p-value must not be compared with α, the confidence level, usually 0.05, but α/a, where a denotes the number of systems to be tested. In other words, given ten systems to test, if we compare m_1 and m_2 using Wilcoxon's signed rank test and it returns a p-value less than $\alpha/10$, then the conclusion is that there is not enough evidence to reject the hypothesis that m_1 performs better than m_2.

As a conclusion, the results of this step are a number of rankings of systems and patterns, i.e., a set of tuples of the form (P, r), in which $P \subseteq \mathbb{P}$ is a subset of patterns and r is a ranking of systems.

Sensitivity analysis. In this step, we need a method to analyze the influence of the parameters in the performance of the systems. For each setting s, we use ReliefF to rank the influence of the parameters on the performance variable [32]. ReliefF is a general parameter estimator that is based on regression, which detects conditional dependences among parameters and the performance variable. As a result, it outputs a number of numerical coefficients for each parameter; the lower a coefficient, the less influence of the corresponding parameter.

Before using ReliefF, we need to normalize the values of the performance variable that we have measured in our scenarios, since ReliefF coefficients would not be comparable otherwise. To perform this normalization, we take the minimum and maximum values of the performance variable in the execution of scenarios for each setting, v_m and v_M, respectively, and we use the following formula:

$v_i' = (v_i - v_m)/v_M$, where v_i is the value of the performance variable in each scenario and v_i' is the normalized value.

After applying ReliefF to our results, we obtain a ranking of the influence of parameters on the performance variable for each setting. Then, it is possible to rank the influence of parameters by clustering systems and/or patterns. Both types of rankings are computed by means of the majoritarian compromise method [34], which clusters individual rankings into a single global ranking. To rank the influence by system, we take the individual rankings of each system and use the majoritarian compromise method to compute the global ranking. Similarly, to rank the influence by pattern, we compute the global ranking taking the individual rankings of each pattern into account.

This step outputs a number of rankings on the influence of parameters for each setting, for each system and every pattern, and for each pattern and every system, i.e., tuples of the form (m, p, r_x), (m, P, r_y), and (M, p, r_z), respectively, where m denotes a system, p denotes a pattern, $P \subseteq \mathbb{P}$ denotes a subset of patterns, $M \subseteq \mathbb{M}$ denotes a subset of systems, and r_x, r_y, and r_z denote rankings on the influence of parameters.

6.1.4 Decision Making

In this step, the user use the previous rankings to make informed and statistically sound decisions on which the best system is regarding the patterns and the analysed performance variable.

6.2 Example with Real-World Patterns

To perform this example of our evaluation methodology, we used a tool implemented using Java that allows to execute the scenarios on several data exchange systems [30]. The tool was run 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, Oracle 11.2.0.1.0, Jena 2.6.4, ARQ 2.8.7, TDB 0.8.7, Pellet 2.2.2, and OWLIM 4.2. Regarding the execution of scenarios, we forced our benchmark to wait for 10 seconds every two scenario execution to ensure that the Java garbage collector had finished collecting old objects from memory. Furthermore, we dropped and created the Oracle's tablespace in which we stored the ontologies in each scenario execution.

Note that, other benchmarks perform warm-ups of the systems under test before running their experiments, since they focus on query response or reasoning time, which may be improved by these warm-ups [7]. However, in our benchmark, we are interested in real-world data exchange problems, in which the exchange of data is performed offline, so we avoid warm-ups to fill caches since it is not the usual behavior in real-world problems.

6.2.1 Initial Setup

For this example, we selected 10 data exchange systems to test (recall that a system comprises an RDF store, a reasoner, and a query engine). The selected systems were the following: $m_1 =$ Jena & ARQ & Jena Reasoner; $m_2 =$ Jena & ARQ & Pellet; $m_3 =$ Jena & ARQ & Oracle Reasoner; $m_4 =$ Jena & ARQ & OWLIM Reasoner; $m_5 =$ TDB & ARQ & Pellet; $m_6 =$ TDB & ARQ & Oracle Reasoner; $m_7 =$ TDB & ARQ & OWLIM Reasoner; $m_8 =$ Oracle & Oracle & Pellet; $m_9 =$ Oracle & Oracle & Oracle Reasoner; $m_{10} =$ Oracle & Oracle & OWLIM Reasoner. Furthermore, we selected all real-world patterns to test: $p_1 =$ Evolution of an ontology; $p_2 =$ Vocabulary adaptation; $p_3 =$ Publication of Linked Open Data. We also selected the following values: $P_I = \{1{,}250, 500, 750, 1{,}000, 1{,}250, 1{,}500, 1{,}750, 2{,}000, 2{,}250\}$; $P_{IT} = P_{ID} = P_{IO} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, which amounts to 10,000 scenarios for each pattern. Finally, we selected CPU time as the performance variable.

TABLE 1
Results of the Analysis of Sample Real-World Patterns

Pattern / System	p_1			p_2			p_3		
	Ratio	Iter.	Dec.	Ratio	Iter.	Dec.	Ratio	Iter.	Dec.
m_1	36.56%	2	×	49.99%	1	×	49.99%	1	×
m_2	49.49%	1	×	48.78%	1	×	33.87%	2	×
m_3	1.88%	2	✓	37.51%	2	×	5.11%	2	✓
m_4	2.63%	4	✓	0.45%	1	✓	1.67%	2	✓
m_5	49.62%	1	×	48.82%	1	×	46.78%	1	×
m_6	4.98%	2	✓	47.19%	1	×	24.63%	1	×
m_7	7.71%	3	✓	4.79%	3	✓	7.82%	3	✓
m_8	48.43%	1	×	48.95%	1	×	27.12%	1	×
m_9	25.62%	1	×	1.50%	2	✓	0.32%	1	✓
m_{10}	34.95%	1	×	25.18%	2	×	28.53%	2	×

(a) Sample size analysis.

Patterns	Rankings
$\{p_1\}$	$m_7 > m_4 > m_6 > m_3$
$\{p_2\}$	$m_9 > m_7 > m_4$
$\{p_3\}$	$m_7 > m_4 > m_6 > m_9$
$\{p_1, p_2, p_3\}$	$m_7 > m_4$

(b) Performance analysis.

Systems	Patterns	Rankings
$\{m_3\}$	$\{p_1\}$	$I > I_T > I_O > I_D$
$\{m_4\}$	$\{p_1\}$	$I > I_T > I_D > I_O$
$\{m_6\}$	$\{p_1\}$	$I > I_T > I_D > I_O$
$\{m_7\}$	$\{p_1\}$	$I > I_T > I_O > I_D$
$\{m_4\}$	$\{p_2\}$	$I > I_D > I_T > I_O$
$\{m_7\}$	$\{p_2\}$	$I > I_T > I_O > I_D$
$\{m_9\}$	$\{p_2\}$	$I > I_D > I_T > I_O$
$\{m_3\}$	$\{p_3\}$	$I_T > I > I_O > I_D$
$\{m_4\}$	$\{p_3\}$	$I > I_T > I_O > I_D$
$\{m_7\}$	$\{p_3\}$	$I > I_T > I_D > I_O$
$\{m_9\}$	$\{p_3\}$	$I > I_T > I_O > I_D$
$\{m_3\}$	$\{p_1, p_3\}$	$I = I_T > I_O > I_D$
$\{m_4\}$	$\{p_1, p_2, p_3\}$	$I > I_T > I_D > I_O$
$\{m_7\}$	$\{p_1, p_2, p_3\}$	$I > I_T > I_O > I_D$
$\{m_9\}$	$\{p_2, p_3\}$	$I > I_T > I_D > I_O$
$\{m_3, m_4, m_6, m_7\}$	$\{p_1\}$	$I > I_T > I_O = I_D$
$\{m_4, m_7, m_9\}$	$\{p_2\}$	$I > I_T = I_D > I_O$
$\{m_3, m_4, m_7, m_9\}$	$\{p_3\}$	$I > I_T > I_O > I_D$
$\{m_4, m_7\}$	$\{p_1, p_2, p_3\}$	$I > I_T > I_D > I_O$

(c) Sensitivity analysis.

6.2.2 Scenario Execution

We used our iterative method to execute the scenarios for each system and pattern using thresholds $\mu = 0.20$, $\delta = 5$; in other words, we discarded a system if the ratio of scenarios to be executed was greater than 20 percent or the iterations of the method were greater than five. Table 1a shows our results for each system and pattern; the first column shows the ratio between the scenarios that we should execute and the total number of scenarios according to Cochran's formula, the second column indicates the number of iterations using our method, and the third column indicates our decision, i.e., if we accept or discard a system. We discarded six systems for patterns $\{p_1, p_3\}$, and seven systems for pattern p_2. (Please, recall that this means that

TABLE 2
Sample Size Analysis (Sample Synthetic Patterns)

Pattern	p_4			p_5			p_6			p_7			p_8			p_9			p_{10}		
System	Ratio	Iter.	Dec.	Ratio	Iter.	Dec.	Ratio	Iter.	Dec.	Ratio	Iter.	Dec.	Ratio	Iter.	Dec.	Ratio	Iter.	Dec.	Ratio	Iter.	Dec.
m_1	48.79%	1	×	49.99%	1	×	49.99%	1	×	49.99%	1	×	49.98%	1	×	49.95%	1	×	49.95%	1	×
m_2	49.09%	1	×	49.63%	1	×	39.37%	2	×	49.99%	1	×	49.99%	1	×	48.78%	1	×	48.63%	1	×
m_3	3.96%	2	✓	3.15%	2	✓	1.05%	2	✓	32.02%	2	×	15.21%	4	✓	8.19%	3	✓	48.17%	1	×
m_4	5.54%	3	✓	4.96%	2	✓	2.92%	2	✓	4.74%	2	✓	26.02%	2	×	1.83%	2	✓	49.93%	1	×
m_5	48.81%	1	×	49.52%	1	×	47.31%	1	×	49.99%	1	×	49.99%	1	×	48.82%	1	×	49.56%	1	×
m_6	9.44%	4	✓	9.32%	3	✓	4.47%	3	✓	27.74%	2	×	28.26%	2	×	3.90%	3	✓	49.63%	1	×
m_7	12.54%	4	✓	11.14%	4	✓	8.85%	3	✓	10.12%	3	✓	36.79%	2	×	6.14%	4	✓	49.95%	1	×
m_8	49.25%	1	×	48.92%	1	×	46.15%	1	×	49.99%	1	×	49.98%	1	×	48.95%	1	×	48.12%	1	×
m_9	25.96%	2	×	24.17%	3	×	20.70%	2	×	19.05%	4	✓	47.47%	1	×	5.79%	2	✓	47.08%	1	×
m_{10}	29.26%	2	×	40.79%	2	×	33.72%	2	×	36.29%	2	×	37.37%	2	×	29.87%	2	×	49.88%	1	×

the variance of the performance variable is very high, which makes it impossible to draw statistically sound conclusions.) In Table 1a, the ratio between executed scenarios and parameters for the remaining systems does not fall below ten, i.e., all of them are greater than 40.

6.2.3 Performance and Sensitivity Analyses

Performance analysis. We took the timings of the previous step, and we first used Kruskal-Wallis's H test to compute if there were significant differences among the systems regarding the selected patterns. Kruskal-Wallis's H test outputted the following p-values: for pattern p_1: p-value $= 4.27\ 10^{-114} < \alpha/4$; for pattern p_2: p-value $= 7.23\ 10^{-98} < \alpha/3$; for pattern p_3: p-value $= 4.82\ 10^{-194} < \alpha/4$; and for patterns for pattern $\{p_1, p_2, p_3\}$: p-value $= 0.00 < \alpha/2$. All of these p-values are less than α/n (Bonferroni's correction), in which n is the number of accepted systems; this guarantees that there are significant differences among the systems. Then, we used Wilcoxon's signed rank test to compute pair ranks among the data exchange systems and we combined the results, which are shown in Table 1b. For instance, m_4 performs better than m_7 and m_9 for pattern p_2; similarly, m_4 performs better than m_7 for patterns $\{p_1, p_2, p_3\}$.

Sensitivity analysis. We first normalized the times of the previous step. Then, we used ReliefF to compute a ranking of parameters by setting, and the majoritarian compromise method to combine these rankings. The results are shown in Table 1c. For instance, I influences more than the rest of parameters regarding system m_6 and pattern p_1; similarly, I_T influences more than the rest of parameters regarding system m_3 and pattern p_3.

6.2.4 Decision Making

The following are examples of informed and statistically sound decisions we can make on account of the previous results:

- It is appealing to use m_4 since it spent the shortest CPU time in performing data exchange in the selected real-world patterns.
- It is not appealing to use m_7 since it spent the longest CPU time in performing data exchange.
- If we expect data exchange problems similar to p_1 and I is going to scale, all systems will be much affected by the scaling of I.

6.3 Example with Synthetic Patterns

In this example, we used the same tool, virtual computer and software as in Section 6.2.

6.3.1 Initial Setup

For this example, we selected the same ten data exchange systems to test as in Section 6.2. Furthermore, we selected all synthetic patterns to test: $p_4 =$ Lift Properties; $p_5 =$ Sink Properties; $p_6 =$ Extract Subclasses; $p_7 =$ Extract Superclasses; $p_8 =$ Extract Related Classes; $p_9 =$ Simplify Specialization; and $p_{10} =$ Simplify Related Classes. We also selected the following values: $P_L = P_C = P_{IT} = P_{ID} = P_{IO} = \{1, 2, 3, 4, 5\}$; $P_D = \{25, 50, 75, 100, 125\}$; $P_I = \{1{,}250, 500, 750, 1{,}000\}$, which amounts to 15,625 scenarios for patterns $\{p_4, p_5, p_6, p_7, p_8, p_9\}$, in which I_O is not used since these patterns have no object properties in the source. Furthermore, the number amounts to 28,125 scenarios for pattern p_{10}. Finally, we selected CPU time as the performance variable.

6.3.2 Scenario Execution

We used our iterative method to execute the scenarios for each system and pattern using thresholds $\mu = 0.20$, $\delta = 5$; in other words, we discarded a system if the ratio of scenarios to be executed was greater than 20 percent or the iterations of the method were greater than five. Table 2 shows our results. We discarded five systems for pattern p_9, six systems for patterns $\{p_4, p_5, p_6\}$, seven systems for pattern p_7, nine systems for pattern p_8, and every system for pattern p_{10}. The last case entails that the variability of the performance variable is too high, which prevents us from making informed and statistically sound decisions using the previous thresholds. In Table 2, the ratios between executed scenarios and parameters for the remaining systems does not fall below 10, i.e., they are greater than 70.

6.3.3 Performance and Sensitivity Analyses

Performance analysis. We used Kruskal-Wallis's H test to compute if there were significant differences among the systems regarding the selected patterns. Kruskal-Wallis's H test outputted the following p-values: for pattern p_4: p-value $= 7.16\ 10^{-287} < \alpha/4$; for pattern p_5: p-value $= 2.38\ 10^{-275} < \alpha/4$; for pattern p_6: p-value $= 5.07\ 10^{-172} < \alpha/4$; for pattern p_7: p-value $= 8.24\ 10^{-95} < \alpha/3$; for pattern p_9: p-value $= 3.93\ 10^{-142} < \alpha/5$; and for patterns $\{p_4, p_5, p_6, p_7, p_9\}$: p-value $= 0.00 < \alpha/2$. Note that all of these p-values

TABLE 3
Results of the Performance and Sensitivity Analysis of Sample Synthetic Patterns

Patterns	Rankings
$\{p_4\}$	$m_4 > m_7 > m_6 > m_3$
$\{p_5\}$	$m_7 > m_4 > m_6 > m_3$
$\{p_6\}$	$m_7 > m_4 > m_6 > m_3$
$\{p_7\}$	$m_9 > m_7 > m_4$
$\{p_9\}$	$m_7 > m_6 > m_9 > m_4 > m_3$
$\{p_4, p_5, p_6, p_7, p_9\}$	$m_7 > m_4$

(a) Performance analysis.

Systems	Patterns	Rankings
$\{m_3\}$	$\{p_4\}$	$L > I > C > I_O > I_D > I_T > D$
$\{m_4\}$	$\{p_4\}$	$I > L > C > I_O > I_D > D > I_T$
$\{m_6\}$	$\{p_4\}$	$I > L > C > I_O > I_D > D > I_T$
$\{m_7\}$	$\{p_4\}$	$I > L > C > I_O > I_D > D > I_T$
$\{m_3\}$	$\{p_5\}$	$L > C > I > I_O > I_D > I_T > D$
$\{m_4\}$	$\{p_5\}$	$I > L > C > I_O > I_T > D > I_D$
$\{m_6\}$	$\{p_5\}$	$L > C > I > I_O > I_T > I_D > D$
$\{m_7\}$	$\{p_5\}$	$I > D > L > I_T > C > I_D > I_O$
$\{m_3\}$	$\{p_6\}$	$I > I_D > I_O > C > L > D > I_T$
$\{m_4\}$	$\{p_6\}$	$I > I_D > I_O > D > L > C > I_T$
$\{m_6\}$	$\{p_6\}$	$I > I_D > I_O > L > D > C > I_T$
$\{m_7\}$	$\{p_6\}$	$I > I_D > I_O > D > L > C > I_T$
$\{m_4\}$	$\{p_7\}$	$I > L > I_O > D > C > I_D > I_T$
$\{m_7\}$	$\{p_7\}$	$I > I_D > I_O > L > C > D > I_T$
$\{m_9\}$	$\{p_7\}$	$I > I_D > L > I_O > C > D > I_T$
$\{m_3\}$	$\{p_8\}$	$I > C > L > I_O > D > I_D > I_T$
$\{m_3\}$	$\{p_9\}$	$I > I_D > I_O > D > L > I_T > C$
$\{m_4\}$	$\{p_9\}$	$I > D > I_O > L > I_D > C > I_T$
$\{m_6\}$	$\{p_9\}$	$I > I_D > I_O > C > L > D > I_T$
$\{m_7\}$	$\{p_9\}$	$I > I_D > I_O > D > L > C > I_T$
$\{m_9\}$	$\{p_9\}$	$I > L > C > I_O > D > I_D > I_T$
$\{m_3\}$	$\{p_4, p_5, p_6, p_8, p_9\}$	$I > L > I_O = C > I_D > D > I_T$
$\{m_4\}$	$\{p_4, p_5, p_6, p_7, p_9\}$	$I > L > I_O > D > C > I_D > I_T$
$\{m_6\}$	$\{p_4, p_5, p_6, p_9\}$	$I > L > I_O > I_D = C > D > I_T$
$\{m_7\}$	$\{p_4, p_5, p_6, p_7, p_9\}$	$I > I_D > I_O = L > D > C > I_T$
$\{m_9\}$	$\{p_7, p_9\}$	$I > L > C = I_O = I_D > D > I_T$
$\{m_3, m_4, m_6, m_7\}$	$\{p_4\}$	$I > L > C > I_O > I_D > D > I_T$
$\{m_3, m_4, m_6, m_7\}$	$\{p_5\}$	$L > I > C > I_O > I_T > D > I_D$
$\{m_3, m_4, m_6, m_7\}$	$\{p_6\}$	$I > I_D > I_O > D > L > C > I_T$
$\{m_4, m_7, m_9\}$	$\{p_7\}$	$I > I_D > I_O > L > C > D > I_T$
$\{m_3, m_4, m_6, m_7, m_9\}$	$\{p_9\}$	$I > I_D > I_O > L > D > C > I_T$
$\{m_4, m_7\}$	$\{p_4, p_5, p_6, p_7, p_9\}$	$I > L > I_O > I_D = D > C > I_T$

(b) Sensitivity analysis.

are less than α/n (Bonferroni's correction), in which n is the number of accepted systems; this guarantees that there are significant differences among the systems. Then, we used Wilcoxon's signed rank test to compute pair ranks among the systems, and then we combined the results. The final results are shown in Table 3a, in which it can be seen that m_3 outperforms the rest of the systems for pattern p_6; similarly, m_4 performs better than m_7 for patterns $\{p_4, p_5, p_6, p_7, p_9\}$.

Sensitivity analysis. We normalized the previous times and used ReliefF to compute a ranking of parameters by setting, and the majoritarian compromise method to combine these rankings by systems and patterns. The final results are shown in Table 3b. For instance, L influences more than the rest of parameters regarding system m_6 and pattern p_5; similarly, I influences more than the rest of parameters regarding systems $\{m_4, m_7\}$ and patterns $\{p_4, p_5, p_6, p_7, p_9\}$.

6.3.4 *Decision Making*

The following are examples of informed and statistically sound decisions we can make on account of the previous results:

- It is appealing to use m_3 since it spent the shortest CPU time in performing data exchange in four out of seven selected patterns.
- It is not appealing to use m_7 since it spent the longest CPU time in performing data exchange in three out of seven selected patterns.
- If we expect data exchange problems similar to p_5 and the number of individuals (I) is going to scale, it is better to use systems m_3 or m_6 since they are not much affected by the scaling of I.
- If we expect data exchange problems similar to p_9 and the number of individuals (I) is going to scale, the CPU time will be affected since all systems are much affected by the scaling of I.

7 RELATED WORK

On the one hand, our benchmark provides a catalogue of seven synthetic data exchange patterns, so we compare them with other patterns described by different authors (see Section 7.1). On the other hand, we compare our benchmark with other existing benchmarks in the bibliography (see Section 7.2).

7.1 Data Exchange Patterns

Stojanovic et al. [35] and Noy and Klein [25] worked on ontology evolution, which can be seen as a data exchange problem in which the source is the original ontology, and the target is an evolution of the source ontology. Stojanovic et al. [35] identified 16 atomic changes an ontology may undergo, for example, adding or removing classes, subclasses, properties, subproperties, property domains, or property ranges. These changes can be seen as the simplest operations building on which the evolution of an ontology may be specified. Some combinations of changes are very frequent in practice, which motivated the authors to devise a catalogue of 12 common composite changes. Noy and Klein [25] extended the catalogue of atomic changes to twenty two, accounting for, for example, the reclassification of an instance as a class or viceversa, the declaration of two classes to be disjoint, moving properties to a subclass or superclass, or moving classes among a subclass hierarchy.

Unfortunately, the specification of the evolution of an ontology does not take how data are exchanged into account. Our catalogue of synthetic patterns summarizes common changes we have found in real-world information integration problems: not only specify they how the structure of the source ontology evolves, but also how source data must be exchanged by means of SPARQL queries of the CONSTRUCT type. Our synthetic patterns are instantiated by tuning a number of structure parameters

TABLE 4
Changes Involved in Our Synthetic Patterns

Data Exchange Patterns	Stojanovic et al.	Noy and Klein
Lift Properties	Pull up properties, Move properties, Deep concept copy	Create slot, Delete slot, Attach slot to class, Remove slot from class, Move slot from sub to superclass, Widen a restriction for a slot, Encapsulate slots into a new class
Sink Properties	Pull down properties, Move properties, Deep concept copy	Create slot, Delete slot, Attach slot to class, Move slot from sub to superclass, Narrow a restriction for a slot, Encapsulate slots into a new class
Extract Subclasses	Extract subconcepts, Move properties, Shallow concept copy	Create class, Attach slot to class, Remove slot from class, Add subclass link, Move slot from super to subclass, Encapsulate slots into a new class, Narrow a restriction for a slot, Split a class into several classes
Extract Superclasses	Extract superconcept, Move properties, Shallow concept copy	Create class, Create slot, Attach slot to class, Remove slot from class, Add subclass link, Move slot from sub to superclass, Widen a restriction for a slot, Encapsulate slots into a new class
Extract Related Classes	Extract related concept, Move properties, Shallow concept copy, Move instance	Create class, Attach slot to class, Create slot, Remove slot from class, Move slots to referenced class, Encapsulate slots into a new class, Split a class into several classes
Simplify Specialisation	Pull up properties, Move properties, Shallow concept copy	Delete class, Attach slot to class, Remove slot from class, Remove subclass link, Merge classes, Move slot from sub to superclass, Encapsulate slots into a new class, Widen a restriction for a slot
Simplify Related Classes	Merge concepts, Move properties, Move instance	Delete class, Delete slot, Attach slot to class, Remove slot from class, Merge classes, Encapsulate slots into a new class

that allow to construct scenarios that range from simple atomic changes to complex composite changes.

Table 4 summarizes the changes that our synthetic patterns involve regarding the composite changes devised by Stojanovic et al. [35], and the atomic changes identified by Noy and Klein [25]. Our patterns are related to eleven composite changes identified by Stojanovic et al. [35]. The unique composite change that is not related to our patterns is the Deep property copy, since we do not deal with specializations of properties in the synthetic patterns. We do not deal with these specializations since they are not so common in the Web of Data [13]. Furthermore, our patterns are related to 16 atomic changes that were identified by Noy and Klein [25], since our patterns describe intentions of change in a more coarse-grain than these atomic changes, a single pattern is related to more than one atomic change. The atomic changes that are not related to the synthetic patterns deal with the reclassification of instances as classes, declaring disjoint classes or transitive properties, and sinking or lifting classes within a hierarchy. We do not deal with these atomic changes since they can be dealt with by means of a reasoner [21].

Alexe et al. [2] devised a benchmark that provides 11 data exchange patterns in the information integration context. Their focus was on nested relational models, which makes it difficult to extrapolate it to our context due to a number of differences between these models and ontologies [25], [29], namely:

- A nested relational model defines a tree that comprises a number of nodes, which may be nested and have an arbitrary number of attributes, and it is also possible to specify referential constraints to relate attributes; contrarily, an ontology is not a tree, but a graph in which there is not a root node and it can have cycles.
- An instance in a nested relational model has a unique type that corresponds to an existing node; contrarily, an ontology instance may have multiple types, i.e., multiple classes that need not be related by specialization.
- In nested relational models, queries to exchange data are encoded using XQuery or XSLT, which build on the structure of the XML documents on which they are executed; contrarily, in an ontology, these queries must be encoded in a language that is independent from the structure of the documents used to represent an ontology, since the same ontology may be serialized to multiple languages, for example, XML, N3, or Turtle.

7.2 Benchmarks

The most-related benchmark is LODIB [31], which was devised to test the performance of data exchange systems in the context of Linked Data. LODIB focuses on three e-commerce data exchange patterns, each of which comprises a different source ontology and the same target ontology. This benchmark allows to automatically populate and scale the data of the source ontologies. Unfortunately, LODIB uses four fixed ontologies (three sources and one target), so it is not possible to test the performance of data exchange systems when the structure of the ontologies scale.

Guo et al. [14] presented LUBM, a benchmark to compare systems that support ontologies. This benchmark provides a single university ontology, a data constructor algorithm that allows to create scalable synthetic data, and 14 SPARQL queries of the SELECT type. This benchmark is similar to ours; however, it has a number of crucial differences:

- LUBM focuses on a single ontology. Contrarily, our benchmark focuses on data exchange problems that comprise source and target ontologies and a set of queries.
- LUBM provides a data constructor to populate an ontology in the university context with synthetic data. On the contrary, our benchmark allows to tune the structure and/or data of source and target ontologies for each pattern instantiation.
- LUBM provides 14 SPARQL queries of the SELECT type. Our benchmark provides a query constructor that allows to automatically construct SPARQL queries of the CONSTRUCT type to perform data exchange for the synthetic patterns.

Wu et al. [36] presented the conclusions regarding an implementation of an inference engine for Oracle, which implements the RDFS and OWL entailments, i.e., it performs reasoning over OWL and RDFS ontologies. They described a performance study based on two examples: LUBM, which comprises data that range from 6.7 to 133.7 million triples, and UniProt, which comprises roughly 5 million triples.

Bizer and Schultz [7] presented BSBM, a benchmark to compare the performance of SPARQL queries using native RDF stores and SPARQL-to-SQL query rewriters. It focuses on an e-commerce pattern, and they provide a data constructor and a test driver: the former allows to create large ontologies and offers RDF and relational outputs to compare the approaches; the latter emulates a realistic workload by simulating multiple clients that concurrently execute queries. The benchmark consists of 12 SPARQL queries that are divided into 10 SELECT queries, one DESCRIBE query, and one CONSTRUCT query.

Schmidt et al. [33] presented a benchmark to test SPARQL query engines that is based on DBLP, and comprises both a data constructor and a set of benchmark queries in SPARQL. They study the DBLP data set to construct a realistic set of synthetic data by measuring probability distributions for certain properties, for example, authors or cites in papers. The benchmark comprises seventeen queries that are divided into fourteen SELECT queries and three ASK queries.

Garcia-Castro and Gómez-Pérez [12] described a benchmark to test the interoperability among different systems that implement semantic-web technologies, which represent ontologies in different ontology languages, such as RDFS or OWL. They claim that it is necessary to be aware of these interoperability problems when combining such systems. Morsey et al. [24] presented DBPSB, a benchmark to test the performance of RDF stores and SPARQL query engines using RDF data that do not resemble a relational schema. It automatically generates RDF data of multiple sizes by duplicating the original data and changing their namespaces. The authors selected 25 representative SPARQL queries of the SELECT type that were extracted by analyzing real-world queries issued to the official DBpedia SPARQL endpoint.

Finally, Alexe et al. [2] devised a benchmark that is used to test data exchange systems in the context of nested relational models. Unfortunately, this benchmark is not suitable in our context since ontologies have a number of inherent differences with respect to nested relational models (see Section 7.1).

8 Conclusions

In this paper, we present a benchmark to test data exchange systems in the context of ontologies that build on SPARQL query engines. Existing benchmarks in the bibliography are not suitable to test such systems since:

1. they focus on nested relational models, which are not applicable to ontologies due to a number of inherent differences between them;
2. they do not focus on data exchange problems, which implies that they do not provide source and target ontologies and queries to exchange data;
3. they provide ontologies with a fixed structure in a particular domain and do not allow to tune their structure; or
4. they focus on SELECT queries instead of CONSTRUCT queries that are required to exchange data.

Our benchmark provides a catalogue of three real-world and seven synthetic data exchange patterns. The former are relevant data exchange problems in the context of Linked Open Data. The latter are common integration problems based on current approaches in the context of ontology evolution, and on our experience in information integration. This catalogue of patterns is not meant to be exhaustive: we expect a community effort to extend them.

These patterns can be instantiated into synthetic scenarios by means of seven parameters that allow to control the construction of both the structure and/or data of ontologies. The scaling of the patterns helps analyze the performance of systems when data exchange problems increase their scale in structure and/or data. Finally, our benchmark provides an evaluation methodology that allows to compare systems side by side, and to make informed and statistically sound decisions about their performance. It is applied to a number of patterns and systems, and allows to rank which system performs better or how the performance of a system is influenced by the parameters.

Acknowledgments

The authors would like to thank Christian Bizer, Andreas Schultz, and José C. Riquelme for their helpful suggestions on an earlier draft of this paper. The virtual computer to test the systems was provided by the Communications and Computing Service of the University of Sevilla. The work was supported by the European Commission, the Spanish and the Andalusian R&D&I programmes (grants TIN2007-64119, P07-TIC-2602, P08-TIC-4100, TIN2008-04718-E, TIN2010-21744, TIN2010-09809-E, TIN2010-10811-E, TIN2010-09988-E, and TIN2011-15497-E).

References

[1] B. Aleman-Meza, F. Hakimpour, I.B. Arpinar, and A.P. Sheth, "SwetoDblp Ontology of Computer Science Publications," *J. Web Semantics,* vol. 5, no. 3, pp. 151-155, 2007.
[2] B. Alexe, W.C. Tan, and Y. Velegrakis, "STBenchmark: Towards a Benchmark for Mapping Systems," *Proc. VLDB Endowment,* vol. 1, no. 1, pp. 230-244, 2008.
[3] G. Antoniou and F. van Harmelen, *A Semantic Web Primer.* The MIT Press, 2008.
[4] J.L. Arjona, R. Corchuelo, D. Ruiz, and M. Toro, "From Wrapping to Knowledge," *IEEE Trans. Knowledge Data Eng.,* vol. 19, no. 2, pp. 310-323, Feb. 2007.
[5] M. Bada, R. Stevens, C.A. Goble, Y. Gil, M. Ashburner, J.M. Cherry, J.A. Blake, M.A. Harris, and S. Lewis, "A Short Study on the Success of the Gene Ontology," *J. Web Semantics,* vol, 1, no. 2, pp. 235-240, 2004.
[6] *Schema Matching and Mapping,* Z. Bellahsene, A. Bonifati, and E. Rahm, eds. Springer, 2011.
[7] C. Bizer and A. Schultz, "The Berlin SPARQL Benchmark," *Int' J. Semantic Web Information Systems,* vol. 5, no. 2, pp. 1-24, 2009.
[8] 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 Semantics,* vol. 7, no. 3, pp. 154-165, 2009.
[9] W.G. Cochran, *Sampling Techniques.* John Wiley&Sons, 1977.
[10] R. Fagin, P.G. Kolaitis, R.J. Miller, and L. Popa, "Data Exchange: Semantics and Query Answering," *Theoretical Computer Science,* vol. 336, no. 1, pp. 89-124, 2005.
[11] G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, and G. Antoniou, "Ontology Change: Classification and Survey," *Knowledge Eng. Rev.,* vol. 23, no. 2, pp. 117-152, 2008.

[12] R. Garcia-Castro and A. Gómez-Pérez, "Interoperability Results for Semantic-Web Technologies Using OWL as the Interchange Language," *J. Web Semantics*, vol. 8, no. 4, pp. 278-291, 2010.

[13] B. Glimm, A. Hogan, M. Krötzsch, and A. Polleres, "OWL: Yet to Arrive on the Web of Data?" *Proc. Conf. Linked Data on the Web (LDOW)*, 2012.

[14] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A Benchmark for OWL Knowledge Base Systems," *J. Web Semantics*, vol. 3, nos. 2/3, pp. 158-182, 2005.

[15] Y. Guo, A. Qasem, Z. Pan, and J. Heflin, "A Requirements Driven Framework for Benchmarking Semantic Web Knowledge Base Systems," *IEEE Trans. Knowledge Data Eng.*, vol. 19, no. 2, pp. 297-309, Feb. 2007.

[16] A.Y. Halevy, Z.G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov, "The Piazza Peer Data Management System," *IEEE Trans. Knowledge Data Eng.*, vol. 16, no. 7, pp. 787-798, July 2004.

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

[18] M. Hollander and D.A. Wolfe, *Nonparametric Statistical Methods.* Wiley-Interscience, 1999.

[19] M. Klusch, B. Fries, and K.P. Sycara, "OWLS-MX: A Hybrid Semantic Web Service Matchmaker for OWL-S Services," *J. Web Semantics*, vol. 7, no. 2, pp. 121-133, 2009.

[20] G. Kobilarov, T. Scott, Y. Raimond, S. Oliver, C. Sizemore, M. Smethurst, C. Bizer, and R. Lee, "Media meets Semantic Web: How the BBC Uses DBpedia and Linked Data to Make Connections," *Proc. Sixth European Semantic Web Conf. The Semantic Web: Research and Applications (ESWC)*, pp. 723-737, 2009.

[21] G. Meditskos and N. Bassiliades, "A Rule-Based Object-Oriented OWL Reasoner," *IEEE Trans. Knowledge Data Eng.*, vol. 20, no. 3, pp. 397-410, Mar. 2008.

[22] D.E. Miller and J.T. Kunce, "Prediction and Statistical Overkill Revisited," *Measurement and Evaluation in Guidance*, vol. 6, no. 3, pp. 157-163, 1973.

[23] R.G. Miller, *Simultaneous Statistical Inference.* Springer, 1981.

[24] M. Morsey, J. Lehmann, S. Auer, and A.C.N. Ngomo, "DBpedia SPARQL Benchmark: Performance Assessment with Real Queries on Real Data," *Proc. 10th Int'l Conf. The Semantic Web (ISWC)*, pp. 454-469, 2011.

[25] N.F. Noy and M.C.A. Klein, "Ontology evolution: Not the Same as Schema Evolution," *Knowledge Information Systems*, vol. 6, no. 4, pp. 428-440, 2004.

[26] C. Pedrinaci and J. Domingue, "Toward the Next Wave of Services: Linked Services for the Web of Data," *J. Universal Computer Science*, vol. 16, no. 13, pp. 1694-1719, 2010.

[27] A. Polleres and D. Huynh, "Special Issue: The Web of Data," *J. Web Semantics*, vol. 7, no. 3, p. 135, 2009.

[28] C.R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo, "On Benchmarking Data Translation Systems for Semantic-Web Ontologies," *Proc. Conf. Information and Knowledge Management (CIKM)*, pp. 1613-1618, 2011.

[29] C.R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo, "Generating SPARQL Executable Mappings to Integrate Ontologies," *Proc. 30th Int'l Conf. Conceptual Modeling (ER)*, pp. 118-131, 2011.

[30] C.R. Rivero, D. Ruiz, and R. Corchuelo, "On Benchmarking Data Translation Systems for Semantic-Web Ontologies," technical report, Univ. of Sevilla, http://tdg-seville.info/Download.ashx?id=205, 2012.

[31] C.R. Rivero, A. Schultz, C. Bizer, and D. Ruiz, "Benchmarking the Performance of Linked-Data Translation Systems," *Proc. Int'l Workshop Linked Data on the Web (LDOW)*, 2012.

[32] M. Robnik-Sikonja and I. Kononenko, "Theoretical and Empirical Analysis of ReliefF and RReliefF," *Machine Learning*, vol. 53, nos. 1/2, pp. 23-69, 2003.

[33] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel, "SP^2Bench: A SPARQL Performance Benchmark," *Proc. Int'l Conf. Data Eng. (ICDE)*, pp. 222-233, 2009.

[34] M.R. Sertel and B. Yilmaz, "The Majoritarian Compromise is Majoritarian-Optimal and Subgame-Perfect Implementable," *Social Choice and Welfare*, vol. 16, pp. 615-627, 1999.

[35] L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic, "User-Driven Ontology Evolution Management," *Proc. 13th Int'l Conf. Knowledge Eng. and Knowledge Management. Ontologies and the Semantic Web (EKAW)*, pp. 285-300, 2002.

[36] Z. Wu, G. Eadon, S. Das, E.I. Chong, V. Kolovski, J. Srinivasan, and M. Annamalai, "Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules In Oracle," *Proc. IEEE 24th Int'l Conf. Data Eng. (ICDE)*, pp. 1239-1248, 2008.