On Benchmarking Data Translation Systems for Semantic-Web Ontologies

Carlos R. Rivero, Inma Hernández, David Ruiz, Rafael Corchuelo
University of Sevilla, Spain
{carlosrivero, inmahernandez, druiz, corchu}@us.es

ABSTRACT

Data translation, also known as data exchange, is an integration task that aims at populating a target model using data from a source model. This task is gaining importance in the context of semantic-web ontologies due to the increasing interest in graph databases and semantic-web agents. Currently, there are a variety of semantic-web technologies that can be used to implement data translation systems. This makes it difficult to assess them from an empirical point of view. In this paper, we present a benchmark that provides a catalogue of seven data translation patterns that can be instantiated by means of seven parameters. This allows us to create a variety of synthetic, domain-independent scenarios one can use to test existing data translation systems. We also illustrate how to analyse three such systems using our benchmark. The main benefit of our benchmark is that it allows to compare data translation systems side by side within a homogeneous framework.

Keywords

Data exchange, Semantic Web and ontologies

*Supported by the European Commission (FEDER), 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, and TIN2010-09988-E).

A technical report and an implementation regarding this paper are available at: `http://tdg-seville.info/carlosrivero/DTSBench`

1. INTRODUCTION

The most widespread data models in current databases are relational and nested relational, which include relational and semi-structured schemata [3]. However, there is an steady shift towards modelling data by means of semantic-web ontologies [15], which build on the RDF, RDF Schema and OWL ontology languages for modelling structure and data, and the SPARQL query language to query these data [2].

Existing databases comprise a variety of heterogeneous models, created by different organisations for different purposes, and there is a need to integrate them [3, 12]. In the bibliography, there exist various data integration tasks. Data translation, also known as data exchange, is one of these tasks that aims at populating a target model using data of a source model [6, 7, 13].

Our focus in this paper is on data translation systems using semantic-web ontologies that build on standard query engines, i.e., systems that perform the data translation task by executing a number of queries that extract data from the source model, transform them, and load the result into the target model [3, 10, 12].

Currently, there exists a variety of semantic-web technologies that are suitable to implement the data translation task, e.g., Jena, TDB, Oracle, or Pellet to mention a few. This makes it difficult to assess them from an empirical point of view. It is important to notice that some of these technologies are data translation systems by themselves, e.g., Oracle. However, other technologies need to be combined to form a data translation system, e.g., TDB needs to be combined with a reasoner like Pellet.

In the bibliography, there is a benchmark that focuses on data translation systems for nested relational models [1]; however, it cannot be straightforwardly applied to semantic-web ontologies due to a number of inherent differences with nested relational models [11, 13]. In addition, there are several benchmarks in the bibliography to test semantic-web technologies, namely: [14] focuses on evaluating the performance of SPARQL engines; [5] focuses on both loading ontologies and executing SPARQL queries; [16] focuses on the performance of the Oracle reasoner; LUBM [9] takes into account loading ontologies, executing SPARQL queries, and reasoning. Unfortunately, none of these benchmarks focuses on data translation problems, i.e., they do not provide source and target ontologies, and a number of queries to perform the data translation task.

In this paper, we present a benchmark for testing data translation systems that provides a catalogue of seven data translation patterns. These patterns are common integration problems that occur frequently in practice. Note that this catalogue is not meant to be exhaustive: the patterns described in this paper are based on our experience regarding information integration, and they are the starting point to a community effort on extending them. Furthermore, we provide seven parameters to build synthetic scenarios, which are instantiations of the data translation patterns.

The expected benefit of our benchmark is that it provides a homogeneous framework that allows to compare data translation systems in the context of semantic-web ontologies side by side, thus alleviating the decision of software engineers on adopting a specific data translation system to solve particular integration problems. As far as we know, this is the first benchmark in the bibliography to test such systems. Finally, we illustrate how to analyse three data translation systems using our benchmark and we draw a number of conclusions from this experimental analysis.

This paper is organised as follows: Section 2 presents the data translation task. Section 3 describes our catalogue of data translation patterns. In Section 4, we present the input and output of our benchmark. In Section 5, we evaluate a number of data translation systems using our benchmark. Section 6 presents the related work. And, finally, Section 7 recaps on our main conclusions.

2. THE DATA TRANSLATION TASK

The data translation task between OWL ontologies comprises five steps when it is performed by means of SPARQL queries (cf. Figure 1), namely:

1. Loading: This step consists of loading the source and target ontologies and the set of SPARQL queries from files into appropriate internal models of semantic-web technologies.

2. Reasoning over source: This step is optional and deals with making the knowledge explicit over the source ontology. Note that SPARQL only deals with plain RDF and does not implement RDFS/OWL semantics, therefore, this step may be mandatory in some integration problems.

3. Query executing: This step consists of executing the set of SPARQL queries over the source ontology to produce instances of the target ontology. Note that this step may be performed by every SPARQL query engine, and the result is the same regardless the execution order of queries.

4. Reasoning over target: This step is also optional and it deals with making the knowledge explicit over the target ontology.

5. Unloading: This final step deals with saving the target ontology into a file.

3. DATA TRANSLATION PATTERNS

A data translation pattern represents a common and relevant integration problem that should be supported by every data translation system. Furthermore, each pattern represents an intention of change as occurs in the ontology evolution context, i.e., when an ontology changes in response to a certain need [8].

Data translation patterns may be instantiated into a number of scenarios, each of which is a three element tuple (S, T, Q), in which S is the source ontology, T is the target ontology, and Q is a set of SPARQL queries to perform the data translation task.

Our benchmark provides a catalogue of seven data translation patterns that usually occur in the context of information integration. To devise them, we have leveraged our experience on real-world information integration problems regarding digital libraries (SwetoDBLP and OAEI's 101), semantic web services (OWL-S and MSM), film reviews (DBpedia and Revyu), and ontology evolution (DBpedia 3.2 and DBpedia 3.6). Without an exception, the data translation patterns described in this section occur frequently in these problems.

Below, we present our data translation patterns, which are illustrated in Figure 2. The source ontology is on the left side and the target on the right side; the arrows represent correspondences between entities of both source and target ontologies. Keep in mind that these correspondences are only visual aids to help readers understand the intention of change behind each scenario [4]. In this notation, classes, data properties and object properties are represented as circles, squares and pentagons, respectively. In addition, subclasses are represented between brackets, the domain of a property is represented by nesting the property in a class, and the range is represented between '<' and '>'. Note that *src:* and *tgt:* prefixes are used for the source and target ontologies, respectively. Note also that we omit SPARQL queries due to space limitations.

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, *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, *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 specialise a class. Therefore, a class is split into several subclasses and data properties are distributed amongst them. In the example, every instance of *src:Person* is transformed into an instance of *tgt:Person*. Note that, after performing the data translation task in this example, all target instances are of type *tgt:Person*. However, if the knowledge is made explicit, every instance of *tgt:Person* with a data property instance of *tgt:paper* is also an instance of type *tgt:Author*.

Extract Superclasses: The intention of change is that the user wishes to generalise a class. So, a class is split into several superclasses, and data properties are distributed amongst them. Note that, after performing the data translation task 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 based on a single class. Therefore, the data properties of a class are split into a number of 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

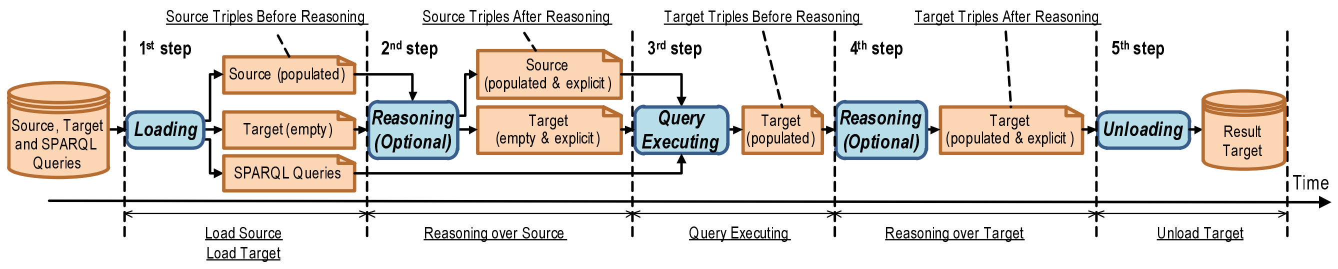


Figure 1: Steps of the data translation task

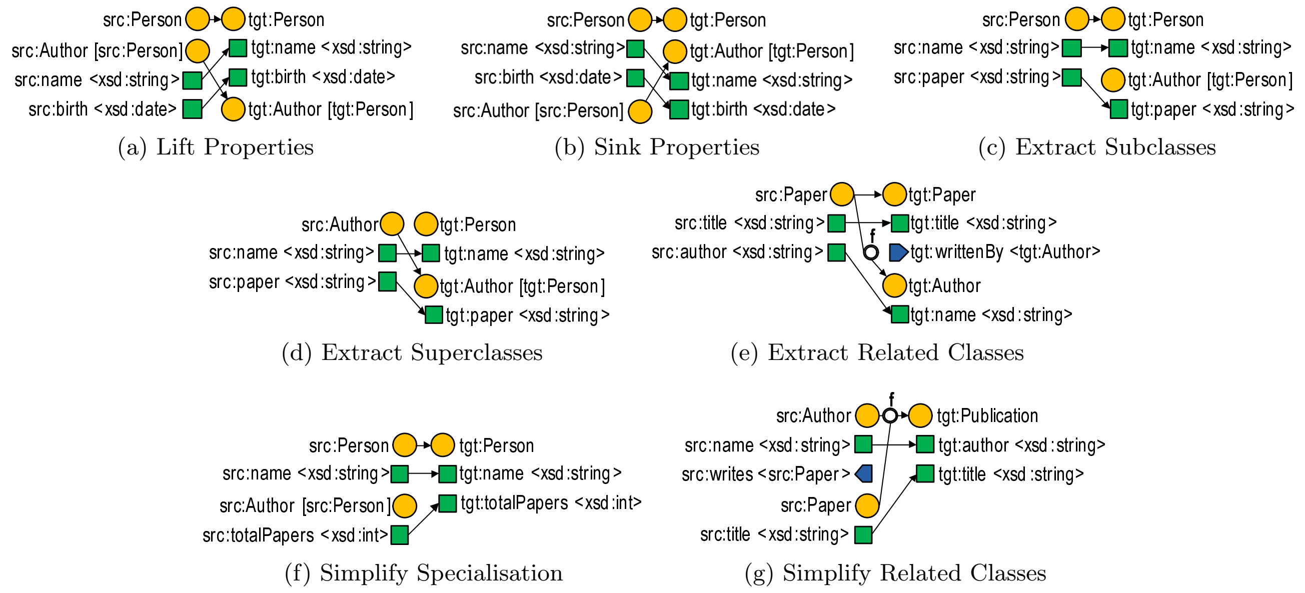


Figure 2: Data translation patterns of our benchmark

called *tgt:Paper* and *tgt:Author* that are related by object property *tgt:writtenBy*. Note that instances of *tgt:Author* are generated by means of function f over the instances of *src:Paper*.

Simplify Specialisation: The intention of change is that the user wishes to remove a hierarchy of classes. So, a set of specialised classes are flattened into a single class. In the example, *src:Person* and *src:Author*, which is an specialisation of *src:Person*, are simplified into *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 properties. Therefore, several source classes, which are related by a number of object properties, are transformed into one class that aggregates them. In the example, for every instance of *src:Author* and *src:Paper* related by *src:writes*, a new instance of *tgt:Publication* is generated.

4. THE BENCHMARK

A benchmark, amongst other properties, should be scalable and the results that it produces should be deterministic and reproducible. To fulfill these properties, we introduce a number of parameters to control the automatic instantiation of our data translation patterns. Furthermore, we describe the measures that we are able to compute.

4.1 Input

Our benchmark takes a number of input parameters that allow to tune both structure and data of source and/or target ontologies. Note that, thanks to these parameters, our benchmark generates synthetic ontologies in both structure and data, which are domain-independent. Note also that the structure of ontologies we generate are trees, each of which comprises, at least, a single root class.

Structure parameters are used to tune the structure of these ontologies, which are the following:

- Levels of classes (L): number of relationships (specialisations or object properties) from the root to each leaf in the source or target ontologies, $L \in \mathbb{N}$.
- Number of related classes (C): number of classes related to each class by specialisation or object properties, $C \in \mathbb{N}$.
- Number of data properties (D): the total number of data properties of the source and target ontologies, $D \in \mathbb{N}$.

Consequently, L allows to scale the structure of ontologies in depth, C in width, and D allows to scale the number of data properties for a particular scenario. L and C may be applied over both source and target ontologies, which is the case of the Lift Properties and Sink Properties patterns; over the target ontology only, i.e., the Extract Subclasses, Extract Superclasses and Extract Related Classes patterns; or over the source ontology only, i.e., the Simplify Specialisation and Simplify Related Classes patterns. Note that structure parameters also affect the SPARQL queries generated by our benchmark, since they vary depending on the structure of the source and target ontologies. Note also that the total number of classes an ontology comprises is computed by the following formula:

$$\sum_{i=0}^{L} C^i.$$

Regarding data parameters, they are used to tune the instances of the source ontology. Note that the goal of the data translation task is to populate the target ontology; therefore, these parameters are only applied to the source ontology since the target ontology remains unpopulated. Data parameters are the following:

- Number of individuals (I): number of individuals (instances of *owl:Thing*) that the populated ontology contains, $I \in \mathbb{N} \backslash \{0\}$.
- Number of types (I_T): number of types that each individual has, $I_T \in \mathbb{N}$.
- Number of data properties (I_D): number of data property instances that a given individual has as domain, $I_D \in \mathbb{N}$.
- Number of object properties (I_O): number of object property instances that a given individual has as domain, $I_O \in \mathbb{N}$.

Note that we may compute the number of data triples that a populated ontology comprises by means of the following formula: $Triples = I + II_T + II_D + II_O$.

4.2 Output

The measures that our benchmark is able to compute are underlined in Figure 1. These measures are the following:

1) Time and memory consumed by loading and unloading ontologies: they represent the time and memory consumed by the data translation system when the source and target ontologies are loaded from files into internal representations, and when the target ontology is unloaded for this internal representation into a file. Note that data translation systems have uneven performance when loading ontologies [5, 9, 14, 16]; in this case, we are able to test if the time a system takes to load an ontology depends only on the number of instances or also on its structure.

2) Time and memory consumed by reasoning over source and/or target ontologies: these measures represent the time and memory consumed by the data translation system when making the knowledge explicit over source and/or target ontologies. In this case, we are able to test if the reasoning time of a data translation system depends on the number of triples and/or the structure of the ontology.

3) Number of triples before and after reasoning: when making the knowledge explicit, there is usually an increment of triples. These measures compute the triples of the ontologies before and after reasoning; in this case, the number of generated triples tests the expressivity of a system.

4) Time and memory consumed by query executing: they represent the time and memory consumed by the data translation system when executing the set of SPARQL queries over the source ontology to generate target data. In this case, we are able to test how the number of triples of the source ontology have influence over the execution time. Furthermore, the structure of the ontology has also influence over this time since a complex ontology structure entails a large number of triple patterns in the SPARQL queries.

5. EVALUATION

In this section, we evaluate three data translation systems based on our benchmark, which are the following: System 1 comprises Jena and Pellet, System 2 is formed by TDB and Oracle, and System 3 comprises TDB and Pellet. Jena and TDB are used to load/unload ontologies and to execute SPARQL queries. In these technologies, ontologies and queries are stored and executed in main memory and the file system, respectively. Furthermore, Pellet and Oracle perform reasoning in main memory and a database schema, respectively.

In the following, we use three tests over the previous data translation systems and we describe evidences that arise from the analysis of the results. In this evaluation, we executed the experiments on a machine with a four core Intel Xeon 3.00 GHz CPU and 16 GB RAM, running on Windows Server 2008 (64-bits) and JRE 1.6.0. The execution times on which we report are the result of averaging 25 actual executions, leaving out outliers on both the left and right 5% tails of the distributions.

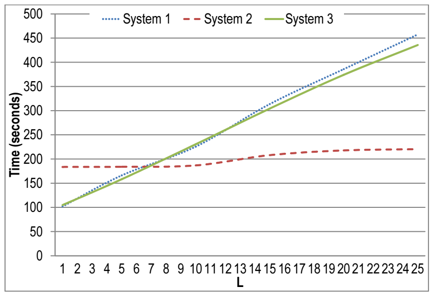
The first test evaluates the performance of data translation systems when scaling the structure of the source and/or target ontologies in depth. Therefore, in this test, we fix all structure and data parameters except L, which ranges from 1 to 25 by 1. The rest of parameters are as follows: $\{C = 1, D = 25, I = 2500, I_T = 5, I_D = 5, I_O = 5\}$.

The second test evaluates the performance of these systems when scaling the structure in width; so the parameters are equal to the previous test except C, which ranges from 1 to 25 by 1, and $L = 1$. Finally, the third test evaluates the performance of these systems when scaling the data of ontologies. In this case, the parameters are fixed as in the previous tests, except $\{L = 1, C = 1\}$ and I, which ranges from 500 to $5,050$ by 175.

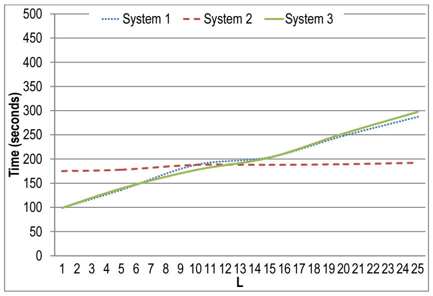
Note that the values of the parameters were selected to generate non-trivial ontologies. In this context, the number of triples in the ontologies of these tests ranges from $3,500$ triples to $80,000$ triples, roughly. The results after performing the tests are shown in Figures 3, 4 and 5, which present the time to perform the data translation task in the Y axis, and the varying input parameter in the X axis.

From our experimental analysis we can draw the following conclusions:

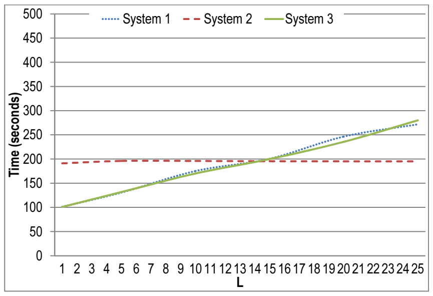
- Systems 1 and 3 perform similarly for all tested scenarios. Consequently, as long as the ontologies involved fit in memory, there does not seem to be any differences between using Jena and TDB.
- Systems 1 and 3 seem to perform linearly in both L and C.
- It seems that neither L nor C have an impact on the



(a) Lift Properties

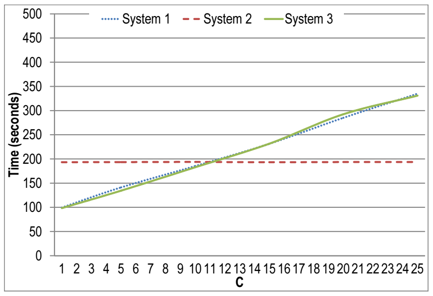


(b) Extract Subclasses

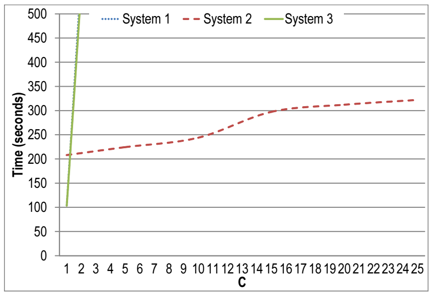


(c) Simplify Specialisation

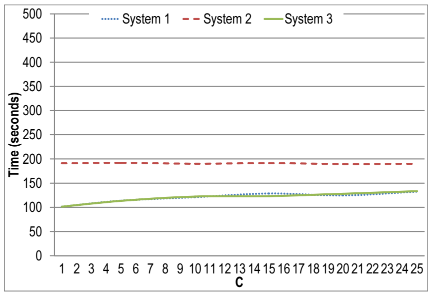
Figure 3: Scaling in depth



(a) Extract Superclasses



(b) Extract Related Classes



(c) Simplify Specialisation

Figure 4: Scaling in width

performance of System 2, which seems almost invariant to changes to these parameters.

- System 2 seems to outperform the others when L and I vary. The result is similar when C varies, the only exception is the Simplify Specialisation pattern. The reason is that Pellet outperforms Oracle when performing reasoning over these scenarios.

- Systems 1 and 3 seems to perform exponentially when I varies. Meanwhile, System 2 seems to perform linearly in this case. The reason is that TDB and Oracle are optimised to work with large sets of triples. Therefore, when these triples are increased from $8,000$ to $16,000$ (between $I = 1000$ and $I = 2000$), TDB and Oracle manage resources that perform linearly beyond this point.

6. RELATED WORK

In the bibliography, there exist benchmarks for data translation systems and semantic-web ontologies.

Regarding data translation benchmarks, Alexe et al. [1] devised a benchmark that is used to test data translation systems in the context of nested relational models. Unfortunately, this benchmark is not suitable in our context since semantic-web ontologies have a number of inherent differences with respect to nested relational models [11, 13].

Regarding benchmarks for semantic-web ontologies, Guo et al. [9] presented LUBM, a benchmark to compare systems that support semantic-web ontologies, which is based on the university domain. This benchmark provides a single ontology, a data generator algorithm that allows to create scalable synthetic data, and fourteen SPARQL queries of the SELECT type. Note that this benchmark is similar to our benchmark; however, it has a number of crucial differences:

- LUBM focuses on a single ontology. Contrarily, our benchmark focuses on data translation problems that comprise source and target ontologies, and a number of queries to perform the data translation task.

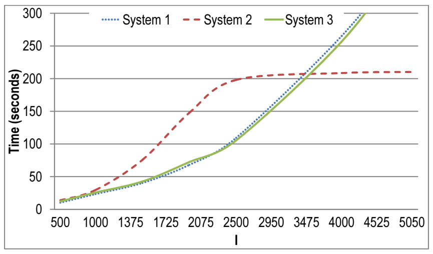
- LUBM provides a data generator to populate an ontology in the university context with synthetic data. On the contrary, our benchmark is domain-independent and it allows to tune the structure and data of source and target ontologies for each pattern instantiation.

- LUBM provides fourteen SPARQL queries of the SELECT type. Our benchmark provides a query generator that allows to automatically build synthetic queries to perform the data translation task. Furthermore, our SPARQL queries are of the CONSTRUCT type, which are suitable to perform this task.

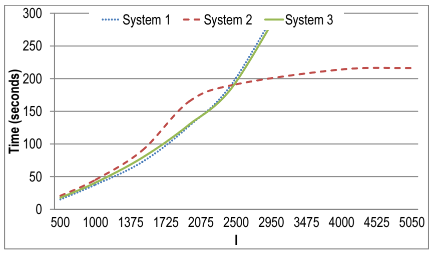
Furthermore, other benchmarks for semantic-web ontologies[1] focus on analysing the performance of the steps of the data translation task in isolation, and they cannot be used to test the data translation task as a whole.

Wu et al. [16] presented the experience of the authors when implementing an inference engine for Oracle. This engine implements the RDFS and OWL entailments, i.e., it performs reasoning over RDFS and OWL ontologies. Furthermore, they presented a performance study based on two examples: LUBM, which data range from 6.7 to 133.7 million triples, and UniProt, which comprises 5 million triples.

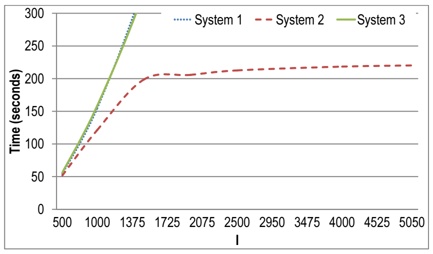
[1]http://esw.w3.org/RdfStoreBenchmarking



(a) Sink Properties



(b) Extract Related Classes



(c) Simplify Related Classes

Figure 5: Data scaling

Bizer and Schultz [5] presented BSBM, a benchmark to compare the performance of SPARQL queries using native RDF stores and SPARQL-to-SQL query rewriters. Their benchmark focuses on an e-commerce use case, and they provide a data generator and a test driver: the former allows to create large datasets and it 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 twelve SPARQL queries that are divided in ten SELECT queries, one DESCRIBE query, and one CONSTRUCT query.

Schmidt et al. [14] presented SP^{2}Bench, a benchmark to test SPARQL query management systems, which is based on DBLP and comprises both a data generator and a set of benchmark queries in SPARQL. They study the DBLP dataset to generate a realistic set of synthetic data by measuring probability distributions for certain attributes, e.g., authors or cites in articles or papers. The benchmark comprises seventeen queries that are divided in fourteen SELECT queries and three ASK queries.

7. CONCLUSIONS

In this paper, we present a benchmark to test data translation systems in the context of semantic-web ontologies. As far as we know, this is the first benchmark in the bibliography in such a context. Existing benchmarks are not suitable to test these systems since they focus on nested relational models, which are not applicable to semantic-web ontologies due to a number of inherent differences with nested relational models; or they do not focus on data translation problems, so they do not provide source and target ontologies and a number of queries to perform this task.

Our benchmark provides a catalogue of seven data translation patterns that represent common and relevant integration problems that should be supported by every data translation system. Furthermore, our benchmark is able to instantiate these patterns based on seven parameters that allow to control the generation of both the structure and data of ontologies. In this paper, we also evaluate three data translation systems based on our benchmark.

Finally, the main benefit of this benchmark is that it provides a homogeneous framework that allows to compare empirically data translation systems side by side, thus alleviating the decision of software engineers on adopting these systems to solve integration problems.

8. REFERENCES

[1] B. Alexe, W. C. Tan, and Y. Velegrakis. STBenchmark: towards a benchmark for mapping systems. *PVLDB*, 1(1):230–244, 2008.

[2] G. Antoniou and F. van Harmelen. *A Semantic Web Primer, 2nd Edition*. The MIT Press, 2008.

[3] P. A. Bernstein and L. M. Haas. Information integration in the enterprise. *Commun. ACM*, 51(9):72–79, 2008.

[4] P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD*, pages 1–12, 2007.

[5] C. Bizer and A. Schultz. The Berlin SPARQL benchmark. *Int. J. Semantic Web Inf. Syst.*, 2009.

[6] C. Drumm, M. Schmitt, H. H. Do, and E. Rahm. Quickmig: automatic schema matching for data migration projects. In *CIKM*, pages 107–116, 2007.

[7] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

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

[9] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.

[10] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. In *SIGMOD*, pages 805–810, 2005.

[11] B. Motik, I. Horrocks, and U. Sattler. Bridging the gap between OWL and relational databases. *J. Web Sem.*, 7(2):74–89, 2009.

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

[13] C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. Generating SPARQL executable mappings to integrate ontologies. In *ER*, 2011.

[14] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP^{2}Bench: A SPARQL performance benchmark. In *ICDE*, pages 222–233, 2009.

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

[16] 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. In *ICDE*, pages 1239–1248, 2008.