MostoDEx: A Tool to Exchange RDF Data using Exchange Samples

Carlos R. Rivero[a,*], Inma Hernández[b], David Ruiz[c], Rafael Corchuelo[c]

[a] *University of Idaho, 875 Perimeter Drive, MS 1010, Moscow, ID 83844-1010, United States*
[b] *Universidad Autonoma de Chile, c/ Carlos Antunez, 1920, Santiago, Chile*
[c] *University of Sevilla, ETSI Informática, Avda. Reina Mercedes s/n, Sevilla E-41012, Spain*

Abstract

The Web is evolving into a Web of Data in which RDF data are becoming pervasive, and it is organised into datasets that share a common purpose but have been developed in isolation. This motivates the need to devise complex integration tasks, which are usually performed using schema mappings; generating them automatically is appealing to relieve users from the burden of handcrafting them. Many tools are based on the data models to be integrated: classes, properties, and constraints. Unfortunately, many data models in the Web of Data comprise very few or no constraints at all, so relying on constraints to generate schema mappings is not appealing. Other tools rely on handcrafting the schema mappings, which is not appealing at all. A few other tools rely on exchange samples but require user intervention, or are hybrid and require constraints to be available. In this article, we present MostoDEx, a tool to generate schema mappings between two RDF datasets. It uses a single exchange sample and a set of correspondences, but does not require any constraints to be available or any user intervention. We validated and evaluated MostoDEx using many experiments that prove its effectiveness and efficiency in practice.

Keywords: Data Exchange, RDF, Schema Mappings

1. Introduction

The current Web is progressively evolving into a Web of Data in which RDF (Resource Description Framework) data are becoming pervasive (Heath and Bizer, 2011). There are thousands of datasets available, many of which share a common purpose but have been developed by independent organisations in isolation (Bizer et al., 2009a). There are many initiatives whose goal is to link these datasets, which is the first step to perform complex integration processes (Heath and Bizer, 2011).

Integration usually refers to several crucial tasks, such as data integration (Lenzerini, 2002), data warehousing (Marileo et al., 2012), model evolution (Flouris et al., 2008), model matching (Shvaiko and Euzenat, 2013), record linkage (Wang et al., 2013), or data exchange (Fagin et al., 2005). In this article, we focus on the latter, whose goal is to populate a target dataset using data that come from one or more source datasets. Data exchange has been paid much attention in the database context, i.e., relational, nested-relational, or XML (Arenas and Libkin, 2008; Fagin et al., 2005; Popa et al., 2002). Furthermore, the emergence of RDF is motivating some authors to work on data exchange in the context of the Web of Data (Barceló et al., 2013; Parreiras et al., 2008; Rivero et al., 2013b).

Data exchange is performed by means of schema mappings, which are declarative specifications of the relationships amongst a source and a target datasets (Alexe et al., 2011a). Generating schema mappings automatically is appealing because this relieves users from the burden of handcrafting them, so researchers have focused on helping users generate them (Qian et al., 2012). Many current tools are based on the data models to be integrated (Bonifati et al., 2005; Haas et al., 2005; Marnette et al., 2011; Mecca et al., 2009; Raffio et al., 2008; Rivero et al., 2013c). By data model, we refer to a sets of entities (that is, classes and properties) and a set of constraints that describe additional features of entities (for instance, class A is a specialisation of class B, property P has class C as its domain, and so on). In the Web of Data, there are many data models that comprise very few or no constraints at all, which typically results in data models that merely specify set of entities (Heath and Bizer, 2011; Lausen et al., 2008). Therefore, relying on data models with constraints to generate schema mappings is not appealing in the general context of the Web of Data.

There exist other tools that do not rely on data models. Unfortunately, they rely on handcrafting the schema mappings (Bizer and Schultz, 2010; Dou et al., 2005; Maedche et al., 2002; Mocan and Cimpian, 2007; Parreiras et al., 2008; Ressler et al., 2007), which is not appealing at all; and a few others rely on exchange samples (Alexe et al., 2011b, 2008, 2006; Qian et al., 2012), which make them more appealing, but require user intervention, or are hybrid and require constraints to be available. Note that an exchange sample is an example of source data and how it is exchanged into target data.

In this article, we present MostoDEx[1], a tool to automatically generate schema mappings between two RDF datasets

---

[*] Corresponding author. Tel.: +12088856592.

Email addresses: `crivero@uidaho.edu` (Carlos R. Rivero), `ichernandez@uautonoma.cl` (Inma Hernández), `druiz@us.es` (David Ruiz), `corchu@us.es` (Rafael Corchuelo)

[1] A technical report and a research prototype are available at (Rivero et al., 2013e) and (Rivero et al., 2013d), respectively.

Preprint submitted to The Journal of Systems and Software *July 16, 2014*

using a single exchange sample and a set of *n*:*m* correspondences. An exchange sample comprises a subset of source data and a subset of target data that is the expected result of exchanging the source data. Correspondences are hints that specify which entities in the source and target datasets correspond to each other, i.e., are somewhat related (Bellahsene et al., 2011). These schema mappings can be easily transformed into SPARQL queries.

Our tool does not rely on constraints of the source and target data models and does not require any user intervention, not even to repair the input exchange sample. We have validated our tool using ten data exchange problems amongst various real-world datasets. In our validation, the execution time never exceeded one second, and the data exchanged were as expected by experts in every case, which suggests that it is very efficient in practice and that the generated schema mappings are appropriate. Additionally, we have evaluated the performance of our tool when data exchange problems scale. We used four synthetic data exchange patterns proposed by MostoBM (Rivero et al., 2013a), a benchmark for testing data exchange proposals in the context of the Web of Data. We instantiated the synthetic data exchange patterns into 2 000 non-trivial data exchange problems that we used to evaluate our tool. Our evaluation results suggest that our tool works well as the data exchange problems scale.

The rest of the article is organised as follows: Section 2 presents the tools related to MostoDEx and its main contributions to the state of the art; Section 3 presents some preliminaries that are necessary to understand the internal details of our tool; Section 4 describes how our tool works; Section 5 reports on the validity and scalability evaluation of MostoDEx; and, finally, Section 6 recaps on our main conclusions.

2. Related work

In this section, we present other existing tools that are related to MostoDEx. We present some tools that require the user to handcraft the schema mappings in Section 2.1, others are based on the constraints that comprise the source and target data models to be integrated in Section 2.2, and a last group of tools are based on samples of data to perform data exchange in Section 2.3. Finally, we analyse and discuss the drawbacks of these tools in Section 2.4, which motivated us to work on a new proposal.

2.1. Handcraft-based tools

There are a number of tools that focus on handcrafting schema mappings, which are expressed as queries but can be viewed as implicitly generating schema mappings: WSEE (Mocan and Cimpian, 2007), which stands for the Web Services Execution Environment, builds on a formal framework to describe correspondences in terms of first-order logic formulae that are used to generate schema mappings using the Web Service Modeling Language (WSML). This tool focuses on the problem of data exchange in the context of semantic-web services, i.e., web services that are enriched with semantic annotations to improve their discovery and composition (Forte et al., 2008). This tool is similar in spirit to MAFRA (Maedche et al., 2002) (MApping FRAmework), whose focus is on modelling correspondences in a general-purpose setting. The main difference with the previous tool is that WSEE goes a step beyond formalising correspondences and executes them using a WSML reasoner to exchange data.

MBOTL (Parreiras et al., 2008) (Model-Based Ontology Translation Language) builds on the framework of Model-Driven Engineering in which the ATL (ATLAS Transformation Language) metamodel is extended to support RDF data models, which allows to express constraints on them using OCL (Object Constraint Language). MBOTL comprises a mapping language by means of which users can express schema mappings that are later transformed into the SPARQL query language by means of a library of ATL transformations. This is similar in spirit to R2R (Bizer and Schultz, 2010) (RDF to RDF), OntoMerge (Dou et al., 2005), and Snoogle (Ressler et al., 2007), the difference is the language used to represent the schema mappings: R2R and Snoogle use SPARQL 1.0; whereas OntoMerge uses Web-PDDL schema mappings that are run by means of a first-order logic reasoner.

2.2. Constraint-based tools

They focus on generating schema mappings building on correspondences and constraints on the source and target data models. These tools are able to compute subsets of data in the source dataset that need to be exchanged as a whole, and subsets of data in the target dataset that need to be created as a whole (Rivero et al., 2013b). To compute them, they rely on user-defined constraints and the inherent constraints of certain data models, such as paths from the root to a leaf in a nested-relational data model, or hierarchy relations amongst classes in an RDF data model. Then, several combinations of these subsets of data are used to generate the final schema mappings (Popa et al., 2002).

Clio (Haas et al., 2005) is the state-of-the-art tool in this field. It takes a source and a target nested-relational data models, a number of constraints of each data model, and a number of 1:1 correspondences between them as input, and it generates schema mappings that can be easily transformed into different query languages, such as XQuery, XSLT, or SQL. HePToX (Bonifati et al., 2005) is similar to Clio but it focuses on XML data models, which are a superset of nested-relational data models. Clip (Raffio et al., 2008) allows to generate schema mappings based on *n*:1 correspondences, and it uses a mapping visual language that was specifically designed for nested-relational data models, which includes grouping functions, aggregation functions, or dependent correspondences. +Spicy (Mecca et al., 2009) allows to compute core schema mappings that generate non-redundant target data when performing data exchange. ++Spicy (Marnette et al., 2011) improves +Spicy by allowing more expressive target constraints. MostoDE (Rivero et al., 2013c) is able to work with RDF data models whose constraints are interpreted as graphs that are traversed to compute source and target kernels. A kernel comprises a subset of the source data model that needs to be exchanged as a whole, and a subset of the target data model that needs to be created as a whole. Kernels are translated into schema mappings that are represented in SPARQL 1.1.

2.3. Sample-based tools

These tools aim to generate schema mappings from a set of exchange samples. In the relational or nested-relational contexts, SPIDER (Alexe et al., 2006) helps users understand and maintain the schema mappings generated by Clio by extracting exchange samples from the source and target datasets, and it illustrates the following: 1) relationships in a specific schema mapping, 2) sample source data that this schema mapping would extract when performing data exchange, and 3) the target data generated by those source data. Muse (Alexe et al., 2008) aids users in generating and understanding schema mappings building on exchange samples. It assumes that source and target data models, together with their constraints, exist, and it infers grouping functions by analysing the answers to some questions it poses to the users.

EIRENE (Alexe et al., 2011b) generates a number of schema mappings by means of a finite set of exchange samples. This tool computes whether or not two input exchange samples have incoherences from a structural point of view, i.e., whether or not these two exchange samples generate schema mappings that will result in erroneous target data. If the input set of exchange samples does not have any incoherences, then it generates the schema mappings. MWeaver (Qian et al., 2012) is based on exchange samples and it focuses on target data only. Users are responsible for providing the target data that they wish to be created; then, every piece of data that appears in both source and target data represents a correspondence between two entities. Correspondences and source and target constraints are used to generate schema mappings.

2.4. Discussion

Table 1 summarises the comparison of current tools to generate schema mappings. The ✓ symbol denotes that the tool supports a feature, and symbol × implies that the tool does not support a feature. The features we have analysed are the following: 1) F_1 determines if a tool requires the intervention of the user during the generation of the schema mappings; 2) F_2 determines if a tool requires the existence of source and target constraints to generate the schema mappings; 3) F_3 determines if a tool allows *n*:*m* correspondences; 4) F_4 determines if a tool performs automatic completions when the same source data lead to different target data; 5) F_5 determines if a tool has been tested with real-world scenarios; 6) F_6 determines if the scalability of a tool has been tested.

Regarding handcraft-based tools (Bizer and Schultz, 2010; Dou et al., 2005; Maedche et al., 2002; Mocan and Cimpian, 2007; Parreiras et al., 2008; Ressler et al., 2007), they focus on handcrafting schema mappings, which is not appealing since users have to write them, check whether they work as expected or not, make changes if necessary, and restart this cycle (Petropoulos et al., 2007). Contrarily, our tool automatically generates schema mappings without the intervention of the user, and it uses a single exchange sample and a number of correspondences as input.

	$\mathbf{F_1}$	$\mathbf{F_2}$	$\mathbf{F_3}$	$\mathbf{F_4}$	$\mathbf{F_5}$	$\mathbf{F_6}$
Handcraft-based tools						
(Bizer and Schultz, 2010)	×	✓	×	×	✓	×
(Dou et al., 2005)	×	×	×	×	✓	×
(Maedche et al., 2002)	×	×	✓	×	×	×
(Mocan and Cimpian, 2007)	×	×	×	×	✓	×
(Parreiras et al., 2008)	×	✓	×	×	✓	×
(Ressler et al., 2007)	×	✓	×	×	✓	×
Constraint-based tools						
(Bonifati et al., 2005)	✓	×	×	×	✓	×
(Haas et al., 2005)	✓	×	×	×	✓	✓
(Marnette et al., 2011)	✓	×	×	×	✓	✓
(Mecca et al., 2009)	✓	×	×	×	✓	×
(Raffio et al., 2008)	×	×	×	×	✓	×
(Rivero et al., 2013c)	✓	×	×	×	✓	✓
Sample-based tools						
(Alexe et al., 2011b)	×	✓	×	×	×	×
(Alexe et al., 2008)	✓	×	×	×	×	×
(Alexe et al., 2006)	✓	×	×	×	×	×
(Qian et al., 2012)	✓	×	×	×	×	×
MostoDEx	✓	✓	✓	✓	✓	✓

Table 1: Comparison of tools to generate schema mappings.

Regarding constraint-based tools (Bonifati et al., 2005; Haas et al., 2005; Marnette et al., 2011; Mecca et al., 2009; Raffio et al., 2008; Rivero et al., 2013c), they are not so appealing in the general context of the Web of Data because (Heath and Bizer, 2011): 1) the main difference between RDF and other data modelling languages is that it allows to represent data without an explicit data model; 2) it is not possible to model the whole Web of Data with a single data model, and several data models may exist for the same RDF dataset; 3) data models in this context usually comprise very few constraints or no constraints at all, which entails that they are only simple vocabularies to create web data. Contrarily, our tool does not rely on constraints but on a single exchange sample and a set of correspondences.

Some sample-based tools assume that source and target data models exist, together with their constraints (Alexe et al., 2008, 2006; Qian et al., 2012). Therefore, their main drawback, as in the previous case, is that it is not appealing to rely on source and target data models, together with their constraints, in the general context of the Web of Data. Finally, EIRENE (Alexe et al., 2011b) does not have the previous drawback, but it requires the user to provide an exchange sample for each schema mapping to be automatically generated. Furthermore, if this tool finds the input exchange samples inappropriate to generate schema mappings, the user is responsible for repairing them. Contrarily, our tool requires the user to provide a single exchange sample and a set of correspondences and finds repairs automatically.

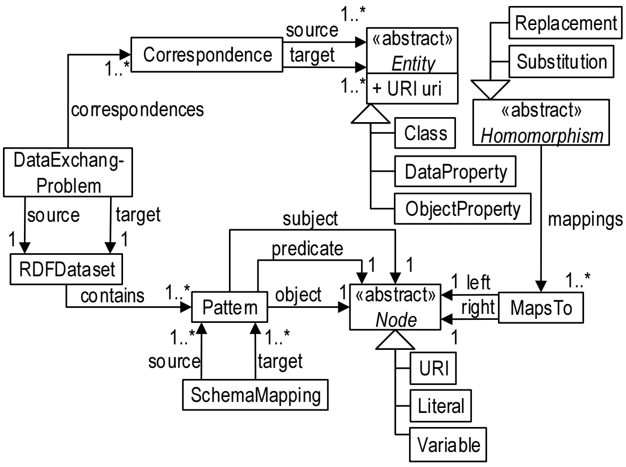


Figure 1: Conceptual model.

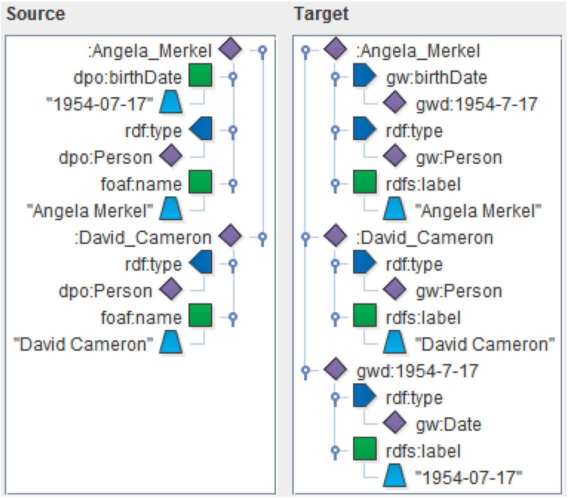
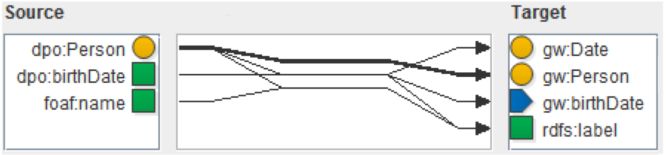


Figure 2: Running example: exchange sample.

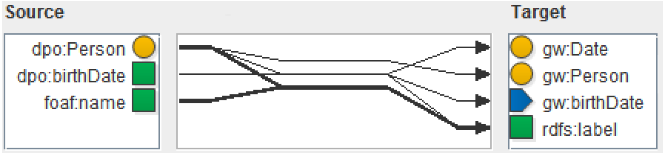
Finally, when dealing with large RDF datasets, a key feature of these tools is their scalability (Fernández et al., 2013). Testing the scalability of these tools is challenging since it requires to collect sufficiently large datasets, and to provide the input data of the tools and the expected output to validate them. Currently, this can be a daunting task since, to the best of our knowledge, there are not any tools to help users perform this validation. Furthermore, most of these tools are research prototypes, therefore, it is not likely that they take scalability issues into account. We have analysed the scalability of our tool using synthetic data exchange problems that are generated with the help of MostoBM (Rivero et al., 2013a), and we report on our results in Section 5.2.

3. Preliminaries

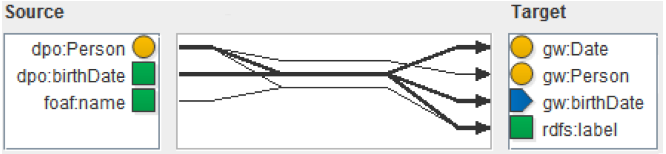
In this section, we present some preliminaries that are necessary to understand our tool. We initially introduce our research methodology in Section 3.1. Afterwards, our tool relies on a conceptual model that is presented in Section 3.2. Furthermore, Section 3.3 describes the running example that we use to illustrate it throughout this article.



(a) Correspondence v_1.



(b) Correspondence v_2.



(c) Correspondence v_3.

Figure 3: Running example: correspondences.

Prefix	URI
:	`http://dbpedia.org/resource/`
rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
rdfs	`http://www.w3.org/2000/01/rdf-schema#`
foaf	`http://xmlns.com/foaf/0.1/`
dpo	`http://dbpedia.org/ontology/`
gw	`http://govwild.org/0.6/GWOntology.rdf#`
gwd	`http://govwild.org/id/date/`

Table 2: Summary of prefixes.

3.1. *Research methodology*

Our research methodology is based on the Unified Process framework (UP) (Kruchten, 2003). This choice is supported by the experience of our research group in applying it to research or technology transfer. The proposed life cycle in UP is iterative and incremental, which is suitable for the development of high dynamic software projects or scientific publications in this area.

It comprises the following steps:

1. Identifying research context: previous to this piece of research work, we identified that exchanging data amongst RDF datasets was an interesting topic and decided to focus on the automatic generation of schema mappings. In MostoDE (Rivero et al., 2013c), we studied this automatic generation based on source and target constraints. In this article, our focus consists on generating them automatically using exchange samples.

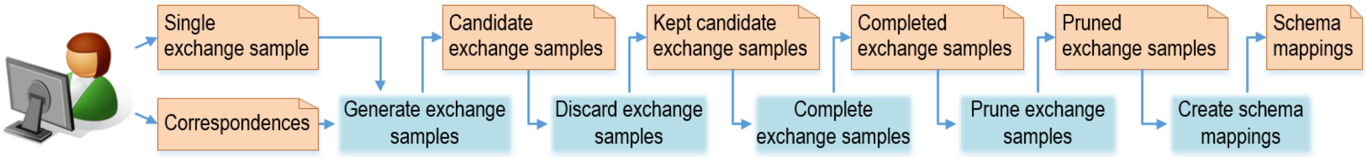


Figure 4: Overview of our schema mapping generation process.

2. Systematic review of the bibliography: we updated the references that we identified when analysing the bibliography for our MostoDE article.
3. Identifying comparison features: we identified those features that are common to existing tools in our research context. These features are described in Section 2.4.
4. Identifying drawbacks: using the previous features, we analysed existing tools in the bibliography regarding whether they have these features or not. The conclusion was that, to the best of our knowledge, no tool has all of the features.
5. Design and implementation of our tool: we devised MostoDEx to take all of the identified features into account.
6. Design of the experiments: every tool should be tested using real-world scenarios to evaluate its effectiveness and efficiency. Furthermore, it is mandatory to evaluate its scalability. We devised 10 real-world data exchange problems to test our tool (see Section 5.1), and 2 000 synthetic data exchange problems to evaluate its scalability (see Section 5.2).

3.2. Conceptual model

An RDF dataset comprises a set of triples, each of which is a three-tuple whose components, which are called subject, predicate, and object, can be URIs (Uniform Resource Identifier) and literals of simple types. A schema mapping is a two-tuple whose components are sets of triple patterns that are implicitly connected using logical ANDs. A triple pattern generalises the concept of triple by allowing the subject and/or the object to be variables or blank nodes. In this article, we refer to triple patterns as patterns for the sake of brevity. Schema mappings may be easily transformed into SPARQL queries in which the two sets of patterns form the WHERE and the CONSTRUCT clauses, respectively. Note that the set of triple patterns includes the set of triples; that is why we usually use the term pattern to refer to both triple patterns and triples.

A homomorphism maps the constants, variables, or blank nodes of a set of patterns onto the constants, variables, or blank nodes of another set of patterns. Homomorphisms can be either replacements or substitutions: a replacement is a finite map from constants to constants and a substitution is a finite map from constants to variables or blank nodes.

Regarding our tool, we restrict our attention to the triples that describe data, that is, triples of the form $(c, rdf{:}type, C)$, in which c is a constant and C is a class, or (c_1, p, c_2), in which c_1 and c_2 are constants and p is a property. An exchange sample comprises a source dataset and a target dataset. An *n*:*m* correspondence relates a set of entities with a different set of entities. A data exchange problem comprises a single exchange sample and a set of correspondences that relate some of the source entities with some of the target entities.

Our algorithms use the following projection functions: *source* to get the source dataset of an exchange sample, the source entities of a given correspondence, or the source triples of a given dataset; *target* to get the target dataset of an exchange sample, the target entities of a given correspondence, or the target triples of a given dataset; *sample* and *correspondences* to get the single exchange sample or the correspondences of a data exchange problem, respectively; and *constants* to get the constants in a set of patterns.

Figure 1 presents an UML-like conceptual model, in which a *DataExchangeProblem* comprises a source *RDFDataset* (the source exchange sample), a target *RDFDataset* (the target exchange sample), and a number of *Correspondences*. Each *Correspondence* has a number of source and target *Entities*, each of which is represented by a *URI* and can be either a *Class*, *DataProperty* or *ObjectProperty*. An *RDFDataset* comprises a set of *Patterns*, each of which is a triple that contains a subject, a predicate and an object *Nodes*. A *Node* can be either a *URI*, a *Literal* or a *Variable*. A *SchemaMapping* comprises a set of source and target *Patterns*. Finally, a *Homomorphism* can be either a *Replacement* or a *Substitution* that maps to a set of *Nodes*.

3.3. Running example

Figures 2 and 3 present a real-world data exchange problem that we use to illustrate our tool. Our goal is to generate a number of schema mappings to perform data exchange from a part of DBpedia 3.8 to a part of GovWILD. On the one hand, DBpedia (Bizer et al., 2009b) is a community effort to annotate and make the data stored at Wikipedia accessible by means of RDF technologies. On the other hand, GovWILD (Böhm et al., 2012) is a public RDF dataset that comprises data from US and EU governments that are connected with financial data of governments or public funds.

The exchange sample in Figure 2 comprises a set of source triples regarding Angela Merkel and David Cameron, their names, and her date of birth; and a set of target triples that specify how these data are structured according to the target entities. This exchange sample is represented using a tree-based graphical notation in which each root node is the subject of a triple, and triples are grouped by subject. A URI or a blank node is represented using a diamond, a literal using a trapezium, a data property using a square, and an object property using a pentagon. We use the prefixes in Table 2, in which the first row specifies the default URI.

Figure 3 shows three correspondences, namely: v_1 relates a person in the DBpedia and the GovWILD datasets; v_2 states that the name of a person in DBpedia is related to the label in GovWILD; and v_3 indicates that a person and her/his date of birth in DBpedia is related to a new URI of class *gw*:*Date* in GovWILD. Correspondences are represented using a tree-based graphical notation in which each root node is an entity, which is represented using a circle, a square, or a pentagon if it is a class, a data property, or an object property, respectively.

4. Generating schema mappings

Our tool takes a data exchange problem as input, which comprises a single exchange sample and a set of correspondences. The single exchange sample is expected to be an equivalent sample of the source and target data that the user wishes to exchange. Furthermore, our tool takes a number of *n*:*m* correspondences over the source and target entities as input. This set indicates the relationships that exist amongst the source and target entities in the data exchange problem that we wish to solve. It is expected that the user has to relate the source entities that should be exchanged as a whole, and the target entities that need to be created as a whole.

Our tool generates a number of schema mappings to exchange data between the source and target datasets. Figure 4 presents an overview of our technique to generate schema mappings that comprises five steps, namely: 1) "Generate exchange samples" takes a single exchange sample and a number of correspondences as input, and automatically generates a set of candidate exchange samples. 2) "Discard exchange samples" discards previously generated candidate exchange samples that are not useful to generate the final set of schema mappings. 3) "Complete exchange samples" adds target data to the different exchange samples if the same source data can lead to different target data. 4) "Prune exchange samples" removes exchange samples that generate the same schema mappings. 5) "Create schema mappings" transforms each exchange sample into a schema mapping. Figure 5 presents the main algorithm that implements such workflow. In our algorithms, we use the following control structures: *for each*, *if*, and *while*; the following logical connectives: negation ($\neg$), and ($\wedge$), or ($\vee$); the following set operators: constructor ($\{\ldots\}$), union ($\cup$), intersection ($\cap$), a finite power set ($\mathbb{F}$). Furthermore, we also use the count operator ($|\ldots|$) and a mapping function ($\mapsto$).

These steps are explained in the rest of this section.

4.1. First step

This step automatically computes a number of candidate exchange samples, each of which comprises a subset of source data that need to be exchanged as a whole, and a subset of target data that need to be created as a whole. To compute them, for each correspondence in isolation, we combine all of the pieces of connected data that contain the entities in the correspondence.

```
algorithm    $generateSchemaMappings$
input        $p : DataExchangeProblem$
output       $M : \mathbb{F}\, SchemaMapping$
variables    $D, D' : \mathbb{F}\, ExchangeSample$

$D := \emptyset$
for each $v : Correspondence \mid v \in correspondences(p)$ do
    — First step
    $D' := createCandidateExchangeSamples(v, sample(p))$
    — Second step
    $D := D \cup discardCandidateExchangeSamples(D')$
end for
— Third step
$D := completeExchangeSamples(D)$
— Fourth step
$D := pruneExchangeSamples(D)$
— Fifth step
$M := createSchemaMappings(D)$
```

Figure 5: Generating schema mappings.

```
algorithm    $createCandidateExchangeSamples$
input        $v : Correspondence; d : ExchangeSample$
output       $D : \mathbb{F}\, ExchangeSample$
variables    $G_S, G_T : \mathbb{F}\, Dataset; T_S, T_T : Dataset$

$D := \emptyset$
— Compute the triples that are related
— to correspondence $v$
$G_S := computeRelatedTriples(source(v), source(d))$
$G_T := computeRelatedTriples(target(v), target(d))$
— Compute the subdatasets that contain
— the triples that are related to $v$
for each $T_S : Dataset \mid T_S \in \prod G_S \wedge$
        $|computeConnectedComponents(T_S)| = 1$ do
    for each $T_T : Dataset \mid T_T \in \prod G_T \wedge$
        $|computeConnectedComponents(T_T)| = 1$ do
        $D := D \cup \{(T_S, T_T)\}$
    end for
end for
```

Figure 6: Generating candidate exchange samples.

Figure 6 shows our algorithm to generate candidate exchange samples from a given correspondence and a single exchange sample. First, we compute the triples related to correspondence v for the single exchange sample d, i.e., the triples that comprise the entities related by v. They are stored in a set of datasets. Then, we compute the distributive cartesian product of both the triples related to $source(v)$ and the triples related to $target(v)$, which is denoted as $\prod$. We iterate over each set of source and target datasets, and we transform them into exchange samples only if each dataset comprises a unique connected component.

Example 1. *To illustrate this step, we focus on correspondence v_2 in our running example. Its source entities are dpo:Person and foaf:name. The triples that comprise dpo:Person are the following:*

(t_1) *:Angela_Merkel* *rdf*:*type* *dpo*:*Person*

(t_2) :*David_Cameron* *rdf*:*type* *dpo*:*Person*

and the triples that comprise foaf:*name are the following:*

(t_3) :*Angela_Merkel* *foaf*:*name* *“Angela Merkel”*
(t_4) :*David_Cameron* *foaf*:*name* *“David Cameron”*

The computeRelatedTriples algorithm outputs the following set in this case: $G_S = \{\{t_1, t_2\}, \{t_3, t_4\}\}$; *the distributive cartesian product of* G_S *is* $\prod G_S = \{\{t_1, t_3\}, \{t_1, t_4\}, \{t_2, t_3\}, \{t_2, t_4\}\}$.

The target entity of v_2 *is rdfs*:*label, and the triples that comprise it are the following:*

(t_5) :*Angela_Merkel* *rdfs*:*label* *“Angela Merkel”*
(t_6) :*David_Cameron* *rdfs*:*label* *“David Cameron”*
(t_7) *gwd*:1954–7–17 *rdfs*:*label* *“*1954–07–17*”*

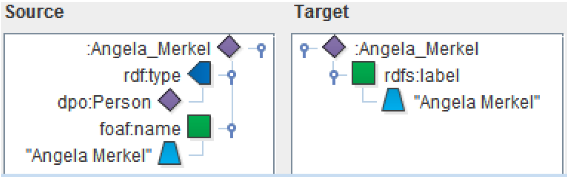
Note that $G_T = \{\{t_5, t_6, t_7\}\} = \prod G_T$. *Additionally, each of the subsets in* $\{\{t_2, t_3\}, \{t_1, t_4\}\} \subseteq \prod G_S$ *has two connected components, since there is no triple that does not have any triple in common with at least another triple. Therefore, we discard these sets of triples. Candidate exchange samples are generated by combining the source triples in* $\prod G_S$ *and the target triples in* $\prod G_T$, *namely:* $d_{21} = (\{t_1, t_3\}, \{t_5\})$, $d_{22} = (\{t_1, t_3\}, \{t_6\})$, $d_{23} = (\{t_1, t_3\}, \{t_7\})$, $d_{24} = (\{t_2, t_4\}, \{t_5\})$, $d_{25} = (\{t_2, t_4\}, \{t_6\})$, $d_{26} = (\{t_2, t_4\}, \{t_7\})$, *which are depicted in Figure 7.*

4.2. Second step

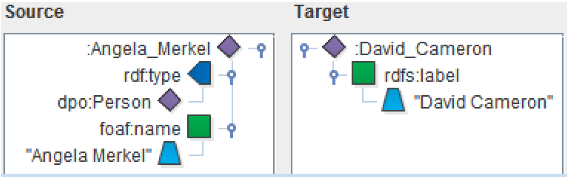
This step consists of discarding candidate exchange samples that are not useful to generate the final set of schema mappings. We keep candidate exchange samples in which there is, at least, a subset of target data that can be generated using the source data, and we minimise the target data that do not exist in the source. The intuition behind this step is that we keep only the exchange samples that provide the maximum information to generate the target data, i.e., when these exchange samples are transformed into schema mappings, they comprise as less blank nodes as possible.

Figure 8 shows our algorithm to discard candidate exchange samples. An exchange example is kept or discarded according to its number of covered and uncovered constants. A constant in the target is said to be covered if there is, at least, a triple in the source that involves that constant; otherwise, it is said to be uncovered. The algorithm first computes the minimum number of uncovered constants in the input set of exchange samples; it then iterates over this set and discards every exchange sample that does not have at least a covered constant or has more uncovered constants than the minimum.

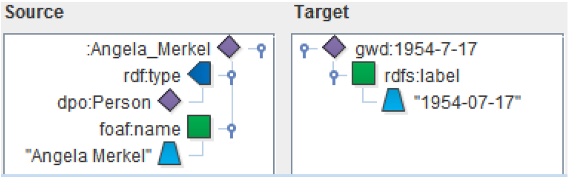
Example 2. *Our tool generates six exchange samples for correspondence* v_2 *(see Figure 7), and the minimum number of uncovered constants in these exchange samples is equal to zero, since every constant in* d_{21} *is covered; therefore, our tool discards exchange samples* d_{22}, d_{23}, d_{24}, *and* d_{26} *in the second step because each of them has two uncovered constants:* :*David_Cameron and “David Cameron”, gwd*:1954–7–17 *and “*1954–07–17*”,* :*Angela_Merkel and “Angela Merkel”, and gwd*:1954–7–17 *and “*1954–07–17*”, respectively.*



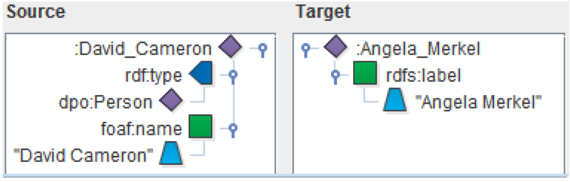
(a) Exchange sample d_{21}.



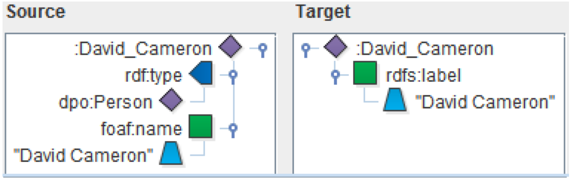
(b) Exchange sample d_{22}.



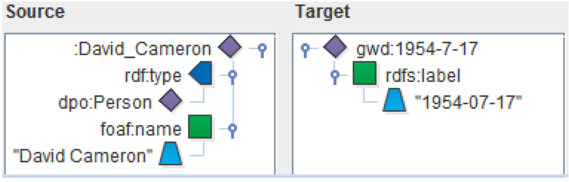
(c) Exchange sample d_{23}.



(d) Exchange sample d_{24}.



(e) Exchange sample d_{25}.



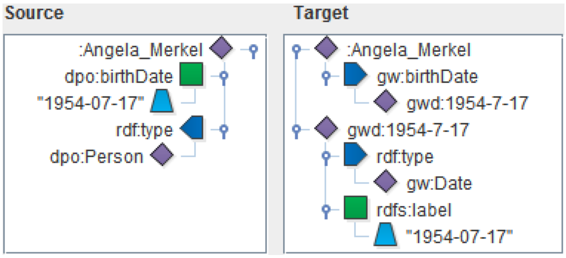
(f) Exchange sample d_{26}.

Figure 7: Exchange samples generated in the first step for correspondence v_2.

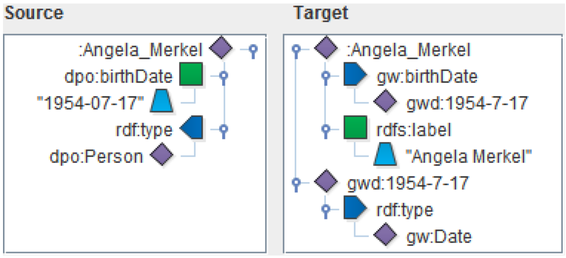
Furthermore, in Figure 9, we present the schema mappings that our tool outputs for correspondence v_3 *of our running example. Note that the minimum number of uncovered constants in these exchange samples is equal to one, since gwd*:1954–7–17 *is not present in the source in any exchange sample. Therefore, our tool discards* d_{32} *since it has two uncovered constants: gwd*:1954–7–17 *and “Angela Merkel”.*

```
algorithm    $discardCandidateExchangeSamples$
input        $D : \mathbb{F}\, ExchangeSample$
output       $D' : \mathbb{F}\, ExchangeSample$
variables    $m : \mathbb{Z}$

$D' := \emptyset$
— Compute the minimum number of
— uncovered constants
$m := \min\{n : Z; d : ExchangeExample \mid d \in D \wedge$
    $n = |(constants(target(d)) \setminus constants(source(d)))| \bullet n\}$
— Discard some exchange samples
for each $d : ExchangeSample \mid d \in D$ do
    if $|(constants(target(d)) \cap constants(source(d)))| > 0 \wedge$
        $|(constants(target(d)) \setminus constants(source(d)))| = m$
    then
        $D' := D' \cup \{d\}$
    end if
end for
```

Figure 8: Discarding candidate exchange samples.



(a) Exchange sample d_{31}.



(b) Exchange sample d_{32}.

Figure 9: Exchange samples generated in the first step for correspondence v_3.

4.3. Third step

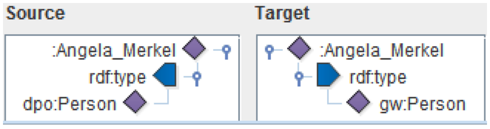
The third step consists of completing exchange samples, i.e., if the same source data can lead to different data in different exchange samples, it is then necessary to complete those exchange samples by adding target data to them. Therefore, we identify the exchange samples that have the same source data but differ in the target data, and we complete them without the user intervention. The completion of exchange samples depends on the specification of the input exchange sample. Our completion process is similar to the process described in (Alexe et al., 2011a), which proves that it is a sound and complete process.

The algorithm in Figure 10 takes a set of exchange samples as input and outputs a number of complete exchange samples. It computes if the input set needs to be completed because the same source data generates different target data. To perform this, we compute the replacements between the exchange samples that have the same source data and, if they have some missing triples, we automatically add them to complete the target data. In this case, *restart* indicates if new triples have been added to the exchange samples, and we iterate until no new triple is added. We extract two different exchange samples from the input set d_1 and d_2, respectively. We compute the replacements between their source triples, and we apply each replacement to the target triples of d_1; if the resulting triples are not present in the target triples of d_2, we have to add them.

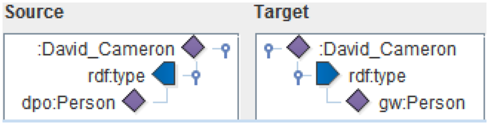
```
algorithm    $completeExchangeSamples$
input        $D : \mathbb{F}\, ExchangeSample$
output       $O : \mathbb{F}\, ExchangeSample$
variables    $H_S : \mathbb{F}\, Replacement; T_T : Dataset; restart : Boolean$

$O := D$
$restart := true$
— Iterate until no new triple is added
while $restart$ do
    $restart := false$
    — Iterate over the whole set of exchange samples
    for each $d_1 : ExchangeSample \mid d_1 \in O \wedge \neg restart$ do
        — Iterate again over the whole set of exchange samples
        for each $d_2 : ExchangeSample \mid d_2 \in O \wedge d_1 \neq d_2 \wedge$
                $\neg restart$ do
            — Compute the replacements
            $H_S := computeReplacements(source(d_1), source(d_2))$
            — Iterate over the replacements
            for each $h : Replacement \mid h \in H_S \wedge \neg restart$ do
                — Apply the replacement
                $T_T := applyHomomorphism(h, target(d_1))$
                — If the new triples are not included
                if $target(d_2) \nsubseteq T_T$ then
                    — Complete the exchange sample
                    $O := O \setminus \{d_2\}$
                    $O := O \cup \{(source(d_2), target(d_2) \cup T_T)\}$
                    $restart := true$
                end if
            end for
        end for
    end for
end while
```

Figure 10: Completing exchange samples.

Example 3. *To illustrate this step, we present the two exchange samples that resulted from correspondence v_1 after the second step (see Figure 11). There exists a single replacement between source(d_{12}) and source(d_{31}) (see Figure 9), which is the following: {:David_Cameron ↦ :Angela_Merkel}. When we apply it to target(d_{12}), it results in the following triple: :Angela_Merkel rdf:type gw:Person. This triple is not included in target(d_{31}), so it is necessary to add this triple to target(d_{31}), and the exchange sample is completed as d'_{31}, which is depicted in Figure 12. The intuition behind this is that we have mapped an instance of dpo:Person as gw:Person in exchange sample d_{12}; however, in exchange sample d_{31}, an instance of*

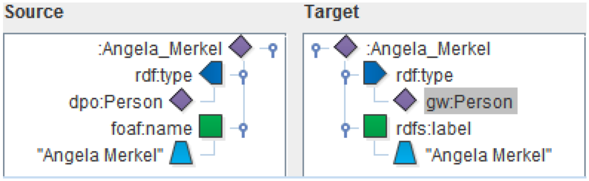


(a) Exchange sample d_{11}.

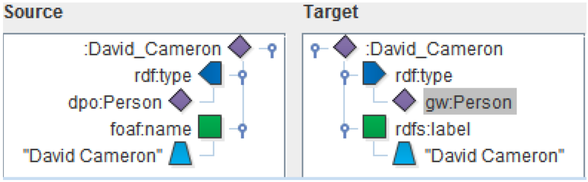


(b) Exchange sample d_{12}.

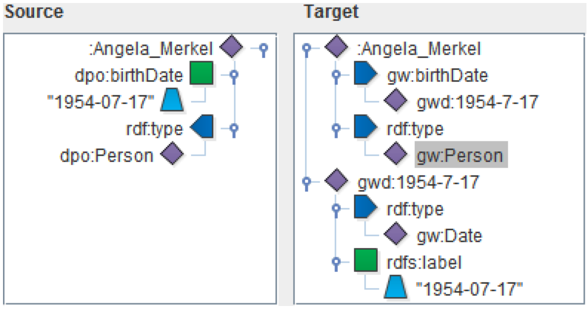
Figure 11: Exchange samples of correspondence v_1 after the second step.



(a) Exchange sample d'_{21}.



(b) Exchange sample d'_{25}.



(c) Exchange sample d'_{31}.

Figure 12: Completed exchange samples.

dpo:Person is not mapped onto an instance of gw:Person.

Note also that Figure 12 presents d'_{21} and d'_{25}, which result from completing exchange samples d_{21} and d_{25}, respectively (see Figure 7). Our tool automatically completes the input exchange samples, which is a clear advantage with respect to some of the existing tools in the bibliography that require the intervention of the user to complete them (Alexe et al., 2011b).

4.4. Fourth step

In this step, our tool prunes redundant exchange samples, i.e., samples that generate the same schema mappings. Figure 13 shows our algorithm to prune these exchange samples. Replacements are used to detect them, i.e., two exchange samples d_1 and d_2 are redundant if there exist, at least, four replacements from the source and target triples of d_1 to the source and target triples of d_2, and from the source and target triples of d_2 to the source and target triples of d_1.

```
algorithm   $pruneExchangeSamples$
input       $D : \mathbb{F}\, ExchangeSample$
output      $D' : \mathbb{F}\, ExchangeSample$

$D' := \emptyset$
— Iterate over exchange samples
for each $d_1 : ExchangeSample \mid d_1 \in D$ do
    — Compute the replacements between samples
    if $\#D' = 0 \lor$
      $(\forall d_2 : ExchangeSample \mid d_2 \in D' \bullet$
        $computeReplacements(source(d_1), source(d_2)) = \emptyset \lor$
        $computeReplacements(target(d_1), target(d_2)) = \emptyset \lor$
        $computeReplacements(source(d_2), source(d_1)) = \emptyset \lor$
        $computeReplacements(target(d_2), target(d_1)) = \emptyset)$
    then
        — Add the exchange sample since it is
        — not redundant
        $D' := D' \cup \{d_1\}$
    end if
end for
```

Figure 13: Pruning exchange samples.

Example 4. *In our running example, d_{11} and d_{12} (see Figure 11) are redundant since there exist two replacements from $source(d_{11})$ to $source(d_{12})$ and vice versa, and two other replacements from $target(d_{11})$ to $target(d_{12})$ and vice versa. Therefore, our tool prunes one of them randomly, e.g., d_{11}. The same happens with exchange samples d'_{21} and d'_{25} (see Figure 12), our tool prunes one of them randomly, e.g., d'_{25}.*

4.5. Fifth step

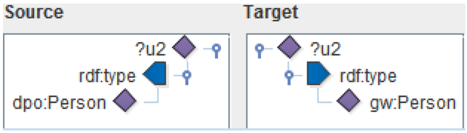
This final step transforms each exchange sample into a schema mapping, which is built by substituting the source and target constants by variables, or blank nodes to generate labelled nulls (Fagin et al., 2005; Mallea et al., 2011). These schema mappings may be easily transformed into SPARQL queries to exchange data between the integrated datasets.

Figure 14 shows our algorithm to transform each exchange sample into a schema mapping, which is built by substituting source and target data by variables or blank nodes, depending on whether the target data is known or not. To perform this, for each exchange sample, we retrieve its source and target constants. Then, we compute a source and a target substitution as follows: for those constants in the source, we add a fresh variable to both substitutions. For those constants that are present in the target but not in the source, we add a fresh blank node to the target substitution. Finally, we apply both substitutions to the source and target triples to generate the source and target patterns of the schema mapping.

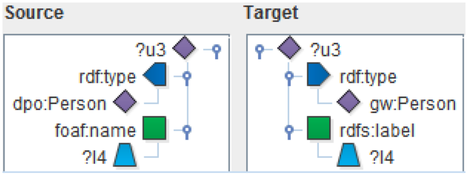
Example 5. *In our running example, our tool generates three schema mappings that result from transforming exchange samples d_{12}, d'_{21}, and d'_{31} (see Figures 11 and 12, respectively).*

```
algorithm   $createSchemaMappings$
input       $D : \mathbb{F}\ ExchangeSample$
output      $M : \mathbb{F}\ SchemaMapping$
variables   $C_S, C_T : \mathbb{F}\ Constant;\ h_S, h_T : Substitution;$
            $T_S, T_T : \mathbb{F}\ Pattern;\ x : Variable$

$M := \emptyset$
for each $d : ExchangeSample \mid d \in D$ do
    — Retrieve source and target constants
    $C_S := constants(source(d))$
    $C_T := constants(target(d))$
    — Compute source and target substitutions
    $h_S := \emptyset$
    $h_T := \emptyset$
    for each $c : Constant \mid c \in C_S$ do
        $x := freshVariable()$
        $h_S := h_S \cup \{c \mapsto x\}$
        $h_T := h_T \cup \{c \mapsto x\}$
    end for
    for each $c : Constant \mid c \in C_T \setminus C_S$ do
        $h_T := h_T \cup \{c \mapsto freshBlankNode()\}$
    end for
    — Apply substitutions
    $T_S := applyHomomorphism(h_S, source(d))$
    $T_T := applyHomomorphism(h_T, target(d))$
    — Update output set
    $M := M \cup \{(T_S, T_T)\}$
end for
```

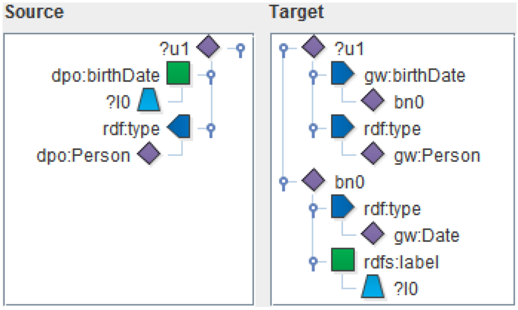
Figure 14: Creating schema mappings.



(a) Exchange sample m_{12}.



(b) Exchange sample m_{21}.



(c) Exchange sample m_{31}.

Figure 15: Final schema mappings.

Our tool transforms exchange sample d_{12} into schema mapping m_{12} by using the following source and target substitution: {:David_Cameron $\mapsto$?u2}. Both substitutions are the same because all of the target constants are already present in the source substitution; so no blank nodes are generated.

Furthermore, our tool transforms exchange sample d'_{21} into schema mapping m_{21} by computing the following source and target substitutions: {:Angela_Merkel $\mapsto$?u3, "Angela Merkel" $\mapsto$?l4}. Note that both substitutions are also the same. Our tool also transforms exchange sample d'_{31} into schema mapping m_{31}. It computes the following source substitution: {:Angela_Merkel $\mapsto$?u1, "1954–07–17" $\mapsto$?l0}, and the following target substitution: {:Angela_Merkel $\mapsto$?u1, "1954–07–17" $\mapsto$?l0, gwd:1954–7–17 $\mapsto$ _:bn0}. The latter comprises a blank node since constant gwd:1954–7–17 is not present in the source.

Figure 15 depicts schema mappings m_{12}, m_{21}, and m_{31}.

5. Evaluation

Our tool is supported by a graphical interface that has been implemented using Java 1.6 and Jena TDB 0.9.3 (Carroll et al., 2004). Furthermore, we have used Guava 13.0.1 to implement ancillary set operations (Google, 2014), and JGraphT 0.8.3 to compute the connected components of a set of patterns (Naveh, 2014). Our tool has a Setup module and five additional modules, each of which implements a step of our proposal, namely: Generate, Discard, Complete, Prune, and Transform.

In the Setup module, the user may select the files in which the source and target data for the single exchange sample are stored. When both files are selected, the user is responsible for providing a number of *n*:*m* correspondences between source and target entities. The Generate module is responsible for taking the single exchange sample and the correspondences of the previous module as input, and generating the whole set of candidate exchange samples. The Discard module takes the set of candidate exchange samples as input and discards exchange samples from this set. The Complete module is responsible for taking the previous samples as input and completing them, i.e., if the same source data generate different target data in different exchange samples, it is necessary to complete those samples by adding new triples to the target data. The Prune module is responsible for pruning exchange samples that are redundant, i.e., they are transformed into the same schema mappings. Finally, the Transform module takes the previous exchange samples and transforms them into a number of schema mappings.

Our experiments were run on a virtual computer that was equipped with a four-threaded Intel Xeon 3.00 GHz CPU and 16 GiB RAM, running on Windows Server 2008 (64-bits). In the rest of this section, we present the validity evaluation in Section 5.1, the scalability evaluation in Section 5.2 of our tool, and some of its limitations in Section 5.3.

5.1. Validity evaluation

Repository. We have setup a repository of ten representative real-world data exchange problems. For each data exchange

Acronym	Data exchange problem
DF-P	DBpedia to Freebase (People)
FD-P	Freebase to DBpedia (People)
DF-TS	DBpedia to Freebase (Television Shows)
FD-TS	Freebase to DBpedia (Television Shows)
DF-F	DBpedia to Freebase (Films)
FD-F	Freebase to DBpedia (Films)
DF-U	DBpedia to Freebase (Universities)
FD-U	Freebase to DBpedia (Universities)
DG	DBpedia to GovWILD
GD	GovWILD to DBpedia

Table 3: Acronyms of the data exchange problems of our repository.

problem, our repository provides a set of handcrafted schema mappings that are expected to perform data exchange appropriately and source data to perform data exchange. The datasets that we use in our repository are the following: Freebase (Bollacker et al., 2008), DBpedia (Bizer et al., 2009b), and GovWILD (Böhm et al., 2012). Table 3 presents these data exchange problems.

Evaluation process. Our evaluation process comprises four steps, namely: 1) We used our tool to automatically generate a set of schema mappings based on the single exchange sample and the set of correspondences of a specific data exchange problem. 2) We transformed each schema mapping into a query mapping in SPARQL. This step is mandatory since our goal was to use a query engine to perform data exchange. We used the target patterns of the schema mapping as the CONSTRUCT clause, and the source patterns as the WHERE clause. 3) We exchanged the source data using both the automatically-generated and the handcrafted queries by executing them over the source dataset to produce a target dataset. 4) We validated if both the target data output by the automatically-generated and handcrafted schema mappings were equivalent to each other.

We implemented this process in a script that uses Java 1.6, Jena TDB 0.9.3, Sesame 2.6.10, and OWLIM Lite 4.2.

Validity results. Table 4 summarises our experimental results. The columns represent the data exchange problems of our repository, and the rows a number of measures; the first group of measures provides an overall idea of the size of each data exchange problem, i.e., the number of source and target triples of the single exchange samples, the correspondences between the entities, and the number of entities involved in the correspondences. The second group of measures provides information about our schema mapping generation, i.e., the time that our tool took to generate them in seconds, the number of handcrafted schema mappings, the number of automatically-generated schema mappings, and the number of completions that our tool performed. The third group provides an overall idea about the exchange of data using these schema mappings, i.e., the number of source triples in millions, the number of target triples generated in millions, and the time the generated schema mappings took to exchange data in minutes.

Note that the output schema mappings do not necessarily correspond to the number of input correspondences. This is due to the fact that some of the generated schema mappings are redundant and we are able to prune them in our fourth step. In the worst case, our tool generates the same number of schema mappings as the input correspondences. This occurs in the DF-U, FD-U, and DG data exchange problems.

The target data generated by our automatically-generated schema mappings were equivalent to the target data generated by handcrafted schema mappings in every data exchange problem. This reveals that the schema mappings that our tool generates agree with the schema mappings that domain experts expect to be generated. The time our tool took to generate them was less than one second in every case; since timings are imprecise in nature, we repeated each experiment 25 times and selected the maximum value. We also measured the time that the schema mappings that we generated automatically took to exchange data. Although these timings depend largely on the technology being used, we think that presenting them is appealing insofar they suggest that the queries can be executed on reasonably-large datasets in a sensible time. Note that these timings depend on the size of the source data to be extracted, and the target data to be generated.

5.2. *Scalability evaluation*

Repository. To the best of our knowledge, little effort has been paid to evaluating the scalability of schema mapping proposals in the context of RDF. MostoBM (Rivero et al., 2013a) provides seven data exchange patterns that are instantiated into a number of data exchange problems using some parameters, and we decided to use them to evaluate our proposal. In this article, we focus on the following data exchange patterns: 1) Lift Properties: the data properties of a set of subclasses are moved to a common superclass. 2) Sink Properties: the data properties of a superclass are moved to a number of subclasses. 3) Extract Subclasses: a source class is split into several subclasses and the domain of the target data properties is selected amongst the subclasses. 4) Extract Superclasses: a class is split into several superclasses, and data properties are distributed amongst them. The other data exchange patterns in MostoBM rely on transformation functions or require reasoning on the datasets, which does not apply to our tool.

Data exchange patterns are instantiated into problems using seven parameters that allow to scale both the entities and the data of a dataset. Since our intention was to evaluate the behaviour of our tool when entities and correspondences scale, we focused on the following subset: 1) Levels of classes (L): number of relationships (specialisations or object properties) amongst one class and the rest of the classes in the source or target datasets; 2) Number of related classes (C): number of classes related to each class by specialisation or object properties; 3) Number of data properties (D): of the source and target datasets.

We instantiated 500 data exchange problems for each data exchange pattern. Each of these data exchange problems com-

	DF-P	**FD-P**	**DF-TS**	**FD-TS**	**DF-F**	**FD-F**	**DF-U**	**FD-U**	**DG**	**GD**
Input data (single exchange sample and correspondences)										
Source triples	33	36	28	38	19	48	23	46	27	13
Target triples	36	33	38	28	48	19	46	23	13	27
Correspondences	20	20	26	27	17	19	14	13	15	18
Source classes	14	8	27	17	17	7	14	6	18	13
Source data properties	9	9	9	9	5	4	8	5	13	14
Source object properties	9	9	15	25	11	21	6	13	9	3
Target classes	14	13	17	28	17	10	11	8	11	21
Target data properties	9	9	9	9	5	4	8	5	13	14
Target object properties	11	9	24	15	28	13	17	8	1	11
Output data (schema mappings)										
Generation time	0.67s	0.59s	0.53s	0.59s	0.64s	0.58s	0.58s	0.56s	0.52s	0.47s
Number (handcrafted)	18	20	26	28	17	20	14	13	15	11
Number (automatic)	18	16	23	26	15	18	14	13	15	11
Completions	26	22	21	3	19	3	12	2	1	24
Application of our schema mappings (data exchange)										
Source triples	14.28M	56.89M	12.75M	53.23M	12.57M	50.51M	51.91M	60.96M	14.37M	7.48M
Target triples	3.94M	16.00M	0.55M	5.06M	2.49M	2.90M	0.33M	0.84M	3.15M	1.27M
Time (automatic)	2.16m	12.06m	0.70m	23.86m	1.13m	16.32m	0.38m	0.71m	2.12m	0.64m

Table 4: Experimental results of our tool.

prises a number of schema mappings to perform data exchange, a number of correspondences, and a populated source dataset with synthetic data. We added a new functionality to MostoBM to generate a source and a target exchange sample to evaluate our proposal. To do this, we generate the source data by creating a single instance of each source entity and, using the schema mappings output by MostoBM, we get the target data by exchanging the source data.

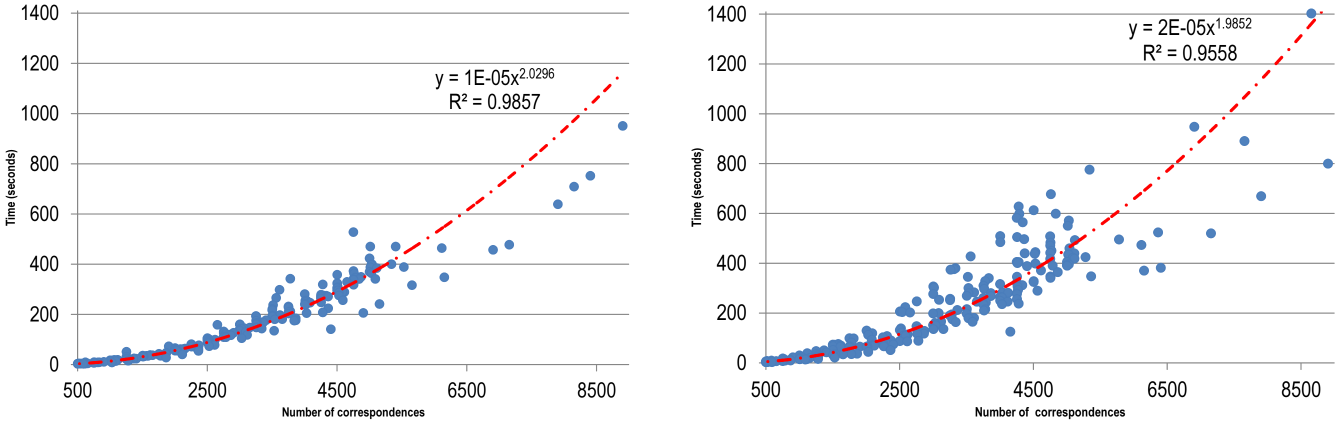
Note that, to provide an idea of the size of these problems, the number of classes ranges from 3 to 7 812, the number of data properties ranges from 250 to 5 000, and the number of object properties ranges from 2 to 7 811.

Evaluation process. This process comprises three steps, namely: 1) We used MostoBM to generate the repository of data exchange problems that result from instantiating a set of data exchange patterns with several values of the input parameters. 2) After generating the repository, it is necessary to run the data exchange problems that it comprises. However, running a single data exchange problem may take hours or even days to complete, since it is executed 25 times (see below). This makes it necessary to select a subset of data exchange problems to execute. Therefore, we used a Monte Carlo method to select 250 data exchange problems for each data exchange pattern, making sure that they combine different values for L, C, and D. 3) We performed the evaluation process of our validation to run the data exchange problems (see Section 5.1).

We implemented this evaluation process in a script that uses Java 1.6 and Jena TDB 0.9.3.

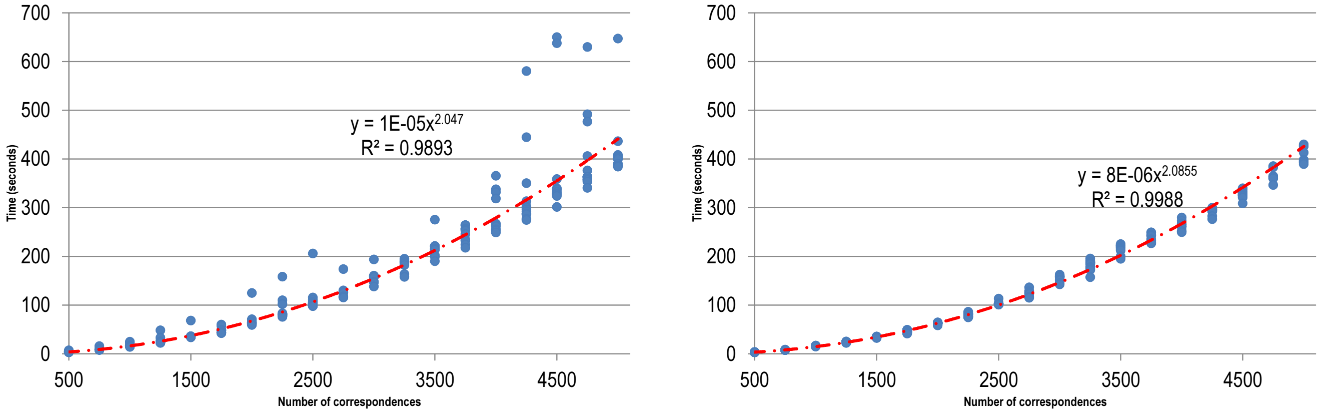
Scalability results. Figure 16 presents our evaluation results; we compared the time our proposal took to generate the schema mappings in the data exchange problems (the Y axis) to the number of correspondences in each data exchange problem (the X axis). Note that, in these problems, the number of sources and target triples of the single exchange sample are equal to the number of source and target entities, and they are also equal to the number of correspondences. Therefore, we only focus on correspondences in this evaluation. Note also that the scaling of correspondences is not linear since the correspondences in the data exchange problems that MostoBM generates are not linear regarding the parameters.

Since timings are imprecise in nature, we repeated each experiment 25 times and averaged the results after discarding roughly 0.01% outliers using the Chebyshev's inequality. For each problem, we checked that the target data that resulted from exchanging data using the schema mappings of MostoBM were equivalent to the results of exchanging data with our automatically generated schema mappings. From our experimental results we can draw the following conclusions: 1) The behaviour of Lift Properties and Sink Properties is similar, as it was also the case for Extract Subclasses and Extract Superclasses. 2) We also computed the minimum squared error tendency line, that is, the one that maximises the R^2 coefficient, and found out that the behaviour is nearly quadratic in every case.



(a) Lift Properties.

(b) Sink Properties.



(c) Extract Subclasses.

(d) Extract Superclasses.

Figure 16: Scalability results of our tool.

5.3. *Limitations*

Despite the fact that we deal with *n*:*m* correspondences, our tool cannot deal with more than one instance of the same class. According to our evaluation results, this limitation does not hinder the applicability of our tool in real-world data exchange problems. A similar problem occurs when the same property has to be used more than once in the same schema mapping. These limitations are due to the fact that correspondences do not state if an entity should appear one or more times in each schema mapping. Neither hinders this limitation its applicability according to our experiments.

Another limitation of our tool is that it does not generate schema mappings that include patterns with regular expressions (Alkhateeb et al., 2009). This implies that we are not able to deal with RDF collections, such as bags, lists, or sequences. However, this limitation does not hinder the applicability of our proposal in practice since these constructs are not recommended when publishing RDF data as Linked Data (Hogan et al., 2012). Furthermore, according to (Glimm et al., 2012), RDF collections do not range amongst the most used constructs in the Web of Data.

Our tool, in its current form, is not able to incorporate literals in the schema mappings, which are mandatory for certain data exchange problems. For instance, if we wish to exchange people that was born in the US, then it is necessary to include US as a literal in the schema mappings. Algorithm *createSchemaMappings* transforms every source literal into a variable (see Figure 14). To deal with this issue, this algorithm can be straightforwardly modified to take a list of constants that must not be transformed.

6. Conclusions

In the general context of the Web of Data, it is not appealing to generate schema mappings building on data models, that is, classes, properties, and constraints, since there exists many data models that comprise very few constraints or no constraints at all. Researchers have proposed an alternative paradigm to generate schema mappings using exchange samples, each of which comprises a subset of source data and a subset of target data, in which the target data is the expected result of exchanging the source data. Unfortunately, some of these proposals require user intervention to handcraft several exchange samples and, if these exchange samples cannot be used to generate schema mappings, the users are responsible for repairing them; or they are hybrid and rely on data models, together with their constraints, which is not appealing in the general context of the Web of Data.

In this article, we present a tool to automatically generate schema mappings amongst RDF datasets using a single exchange sample and a set of *n:m* correspondences. It does not rely on constraints of the source and target data models and does not require any user intervention. We have validated our tool using ten data exchange problems amongst DBpedia, Freebase, and GovWILD datasets. The time to execute this validation never exceeded one second, and the data exchanged were as expected by experts in every case, which suggest that it is very efficient in practice and also that the generated schema mappings are appropriate.

We have also evaluated the scalability of our proposal when data exchange problems scale. We have used four synthetic data exchange patterns proposed by MostoBM, a benchmark for testing data exchange proposals in the context of the Web of Data. The synthetic data exchange patterns were instantiated into 2 000 data exchange problems that we have used to evaluate our proposal. Our evaluation results suggest that it works quite well as the data exchange problems it faces scale.

Acknowledgements

The work that we present in this article was 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, TIN2010-09988-E, TIN2011-15497-E, and TIN2013-40848-R.).

References

Alexe, B., ten Cate, B., Kolaitis, P.G., Tan, W.C., 2011a. Designing and refining schema mappings via data examples, in: SIGMOD Conference, pp. 133–144.

Alexe, B., ten Cate, B., Kolaitis, P.G., Tan, W.C., 2011b. Eirene: Interactive design and refinement of schema mappings via data examples. PVLDB 4, 1414–1417.

Alexe, B., Chiticariu, L., Miller, R.J., Tan, W.C., 2008. Muse: Mapping understanding and design by example, in: ICDE, pp. 10–19.

Alexe, B., Chiticariu, L., Tan, W.C., 2006. SPIDER: A schema mapping debugger, in: VLDB, pp. 1179–1182.

Alkhateeb, F., Baget, J.F., Euzenat, J., 2009. Extending SPARQL with regular expression patterns (for querying RDF). J. Web Sem. 7, 57–73.

Arenas, M., Libkin, L., 2008. XML data exchange: Consistency and query answering. J. ACM 55.

Barceló, P., Pérez, J., Reutter, J.L., 2013. Schema mappings and data exchange for graph databases, in: ICDT, pp. 189–200.

Bellahsene, Z., Bonifati, A., Rahm, E. (Eds.), 2011. Schema Matching and Mapping. Springer.

Bizer, C., Heath, T., Berners-Lee, T., 2009a. Linked Data: The story so far. Int. J. Semantic Web Inf. Syst. 5, 1–22.

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S., 2009b. DBpedia: A crystallization point for the Web of Data. J. Web Sem. 7, 154–165.

Bizer, C., Schultz, A., 2010. The R2R framework: Publishing and discovering mappings on the web, in: COLD.

Böhm, C., Freitag, M., Heise, A., Lehmann, C., Mascher, A., Naumann, F., Ercegovac, V., Hernández, M.A., Haase, P., Schmidt, M., 2012. GovWILD: Integrating open government data for transparency, in: WWW, pp. 321–324.

Bollacker, K.D., Evans, C., Paritosh, P., Sturge, T., Taylor, J., 2008. Freebase: a collaboratively created graph database for structuring human knowledge, in: SIGMOD Conference, pp. 1247–1250.

Bonifati, A., Chang, E.Q., Ho, T., Lakshmanan, L.V.S., Pottinger, R., 2005. HePToX: Marrying XML and heterogeneity in your P2P databases, in: VLDB, pp. 1267–1270.

Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K., 2004. Jena: Implementing the semantic web recommendations, in: WWW, pp. 74–83.

Dou, D., McDermott, D.V., Qi, P., 2005. Ontology translation on the Semantic Web. J. Data Semantics 2, 35–57.

Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L., 2005. Data exchange: Semantics and query answering. Theor. Comput. Sci. 336, 89–124.

Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M., 2013. Binary RDF representation for publication and exchange (HDT). J. Web Sem. 19, 22–41.

Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., Antoniou, G., 2008. Ontology change: Classification and survey. Knowledge Eng. Review 23, 117–152.

Forte, M., de Souza, W.L., do Prado, A.F., 2008. Using ontologies and web services for content adaptation in ubiquitous computing. Journal of Systems and Software 81, 368–381.

Glimm, B., Hogan, A., Krötzsch, M., Polleres, A., 2012. OWL: Yet to arrive on the Web of Data?, in: LDOW.

Google, 2014. Guava: Google core libraries for java 1.6+. `http://code.google.com/p/guava-libraries/`.

Haas, L.M., Hernández, M.A., Ho, H., Popa, L., Roth, M., 2005. Clio grows up: From research prototype to industrial tool, in: SIGMOD, pp. 805–810.

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

Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., Decker, S., 2012. An empirical survey of Linked Data conformance. J. Web Sem. 14, 14–44.

Kruchten, P., 2003. The Rational Unified Process: An Introduction. Addison-Wesley Professional.

Lausen, G., Meier, M., Schmidt, M., 2008. SPARQLing constraints for RDF, in: EDBT, pp. 499–509.

Lenzerini, M., 2002. Data integration: A theoretical perspective, in: PODS, pp. 233–246.

Maedche, A., Motik, B., Silva, N., Volz, R., 2002. MAFRA: A mapping framework for distributed ontologies, in: EKAW, pp. 235–250.

Mallea, A., Arenas, M., Hogan, A., Polleres, A., 2011. On blank nodes, in: ISWC, pp. 421–437.

Marileo, M.C., Bravo, L., Hurtado, C.A., 2012. Repairing inconsistent dimensions in data warehouses. Data Knowl. Eng. 79-80, 17–39.

Marnette, B., Mecca, G., Papotti, P., Raunich, S., Santoro, D., 2011. ++Spicy: An open-source tool for second-generation schema mapping and data exchange. PVLDB 4, 1438–1441.

Mecca, G., Papotti, P., Raunich, S., Buoncristiano, M., 2009. Concise and expressive mappings with +Spicy. PVLDB 2, 1582–1585.

Mocan, A., Cimpian, E., 2007. An ontology-based data mediation framework for semantic environments. Int. J. Semantic Web Inf. Syst. 3, 69–98.

Naveh, B., 2014. JGraphT: a free java graph library. `http://jgrapht.org/`.

Parreiras, F.S., Staab, S., Schenk, S., Winter, A., 2008. Model driven specification of ontology translations, in: ER, pp. 484–497.

Petropoulos, M., Deutsch, A., Papakonstantinou, Y., Katsis, Y., 2007. Exporting and interactively querying web service-accessed sources: The CLIDE system. ACM Trans. Database Syst. 32.

Popa, L., Velegrakis, Y., Miller, R.J., Hernández, M.A., Fagin, R., 2002. Translating web data, in: VLDB, pp. 598–609.

Qian, L., Cafarella, M.J., Jagadish, H.V., 2012. Sample-driven schema mapping, in: SIGMOD Conference, pp. 73–84.

Raffio, A., Braga, D., Ceri, S., Papotti, P., Hernández, M.A., 2008. Clip: A tool for mapping hierarchical schemas, in: SIGMOD Conference, pp. 1271–1274.

Ressler, J., Dean, M., Benson, E., Dorner, E., Morris, C., 2007. Application of ontology translation, in: ISWC, pp. 830–842.

Rivero, C.R., Hernández, I., Ruiz, D., Corchuelo, R., 2013a. Benchmarking data exchange among semantic-web ontologies. IEEE Trans. Knowl. Data Eng. 25, 1997–2009.

Rivero, C.R., Hernández, I., Ruiz, D., Corchuelo, R., 2013b. Exchanging data amongst linked data applications. Knowl. Inf. Syst. 37, 693–729.

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

Rivero, C.R., Hernández, I., Ruiz, D., Corchuelo, R., 2013d. MostoDEx: Research prototype, repositories, scripts, and experimental results . Web page, http://tdg–seville.info/carlosrivero/MostoDEx.

Rivero, C.R., Hernández, I., Ruiz, D., Corchuelo, R., 2013e. Using exchange samples to generate schema mappings amongst RDF datasets . Tech. report, http://tdg–seville.info/Download.ashx?id=382.

Shvaiko, P., Euzenat, J., 2013. Ontology matching: State of the art and future challenges. IEEE Trans. Knowl. Data Eng. 25, 158–176.

Wang, F., Wang, H., Li, J., Gao, H., 2013. Graph-based reference table construction to facilitate entity matching. Journal of Systems and Software 86, 1679–1688.