Generating SPARQL Executable Mappings to Integrate Ontologies * **

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

University of Sevilla, Spain
{carlosrivero, inmahernandez, druiz, corchu}@us.es

Abstract. Data translation is an integration task that aims at populating a target model with data of a source model by means of mappings. Generating them automatically is appealing insofar it may reduce integration costs. Matching techniques automatically generate uninterpreted mappings, a.k.a. correspondences, that must be interpreted to perform the data translation task. Other techniques automatically generate executable mappings, which encode an interpretation of these correspondences in a given query language. Unfortunately, current techniques to automatically generate executable mappings are based on instance examples of the target model, which usually contains no data, or based on nested relational models, which cannot be straightforwardly applied to semantic-web ontologies. In this paper, we present a technique to automatically generate SPARQL executable mappings between OWL ontologies. The original contributions of our technique are as follows: 1) it is not based on instance examples but on restrictions and correspondences, 2) we have devised an algorithm to make restrictions and correspondences explicit over a number of language-independent executable mappings, and 3) we have devised an algorithm to transform language-independent into SPARQL executable mappings. Finally, we evaluate our technique over ten scenarios and check that the interpretation of correspondences that it assumes is coherent with the expected results.

Keywords: Information Integration, Data Translation, Semantic-web Ontologies, SPARQL Executable Mappings

1 Introduction

Data in current databases are usually modelled using relational or nested relational models, which include relational and semi-structured schemata [4]. However, there is an increasing shift towards representing these data by means of ontological models due to the popularity and maturity of the Semantic Web [35].

* 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).

** See [31] for a demo regarding this paper.

In this paper, we focus on semantic-web ontologies that are represented using RDF, RDF Schema and OWL ontology languages to model structure and data, and their data are queried by means of the SPARQL query language [3,10,25].

Existing databases comprise a variety of models, created by different organisations for different purposes, and there is a need to integrate them [4,19,22]. Mediators provide a well-known solution to the problem of integrating models since they can help bridge the semantic gap amongst them [11,33]. A mediator relates a source model, which contains the data of interest, to a target model, which usually contains no data. Mediators can perform two tasks: data integration and data translation [19,22]. The former deals with answering queries posed over the target model, which is virtual, using the source model only [15,19,37]. The latter, which is the focus of this paper, aims at populating a target model with data of a source model [8,12,24,28,29].

Mappings, which are the cornerstone components of mediators, relate source and target models in different ways [8,9,11,18,19]. Building and maintaining mappings automatically is appealing insofar this relieves users from the burden of writing them, checking whether they work as expected or not, making changes if necessary, and restarting this cycle [4,27]. Mappings can be of various types but, in this paper, we focus on two: correspondences and executable mappings.

On the one hand, correspondences may be handcrafted with the help of a visual tool [1], or generated automatically using matching techniques [9,11,30]. They are hints that specify which elements from the source and target models are related in some unspecified way [5]. Therefore, they must be interpreted to perform the data translation task. However, this interpretation is far from trivial since it is not unique, i.e., different approaches interpret correspondences in different ways [2,5,28]. Consequently, it is mandatory to check whether the resulting target data are coherent with the expected results.

On the other hand, executable mappings, a.k.a. operational mappings, encode an interpretation of correspondences in a given query language [14,28,29]. These mappings are executed by means of a query engine to perform the data translation task. The main benefit of using these mappings is that the data translation task is simplified, making it more efficient and flexible: thanks to executable mappings, instead of relying on ad-hoc programs that are difficult to create and maintain, the query engine is used as the transformation engine [14]. Furthermore, these engines incorporate a vast knowledge on query manipulation, from which it is derived that the executable mappings are automatically optimised for better performance of the data translation task.

In the bibliography, there are a number of techniques to automatically generate executable mappings. Unfortunately, none of them can be straightforwardly applied to semantic-web ontologies in the context of the data translation task due to the following reasons, namely (for further details, see Section 2):

- Some of them are based on instance examples of the target model [29], which are suitable in scenarios in which the target model is already populated. However, we focus on scenarios in which the target model has no instances at all, which seems to be quite usual in practice [2,4,28].

– Others focus on nested relational models, which represent trees [13,20,28,36]; however, they cannot be straightforwardly applied to ontologies, which represent graphs, due to a number of differences between them [17,21,23,32].

In this paper, we present a technique to automatically generate SPARQL executable mappings to perform the data translation task between OWL ontologies. To illustrate it, we use an example that is contextualised in the domain of films and reviews, using DBpedia and Revyu as data sources. The original contributions of our technique are that it is based on restrictions and correspondences, instead of instance examples, which makes it appealing in many practical cases. Furthermore, we have devised an algorithm to generate language-independent executable mappings that makes restrictions and correspondences explicit. Finally, we have devised an algorithm to transform language-independent into SPARQL executable mappings by creating triple patterns and linking variables of these patterns.

This paper is organised as follows: Section 2 presents the related work; in Section 3, we present the algorithms to automatically generate SPARQL executable mappings; Section 4 presents the evaluation of our technique; and, finally, Section 5 recaps on our conclusions.

2 Related work

In this section, we study the techniques to automatically generate executable mappings in both the semantic-web and database research fields.

In the semantic-web research field, Qin et al. [29] devised a technique to generate executable mappings between ontologies in a semi-automatic way. Their technique is divided into five modules. The first module deals with the automatic discovering of correspondences. The second module determines whether two instances in different ontologies represent the same real-world entity. The third module deals with the clustering of correspondences that are related. The fourth module takes a set of source and target instances as input and generates a set of frequent queries of interest between them. Note that target instances have to be provided by the user when the target is not populated. Finally, the fifth module generates executable mappings based on frequent queries and correspondences. This technique generates a set of executable mappings that can be specified in Web-PDDL (an ontology language devised by the authors), Datalog or SWRL.

Regarding the database research field, Popa et al. [28] proposed a technique to automatically generate executable mappings for performing the data translation task between nested relational models. A nested relational model defines a tree that comprises nested nodes with attributes. The first step of their technique consists of computing primary paths, each of which is the unique path from the tree root to a node. Furthermore, this step comprises the identification of referential constraints that relate two primary paths by an equality expression between two attributes. The second step consists of applying an extension of the relational chase algorithm to compute logical relations, each of which is an

enumeration of the logical joins specified by referential constraints. Therefore, a logical relation comprises a number of primary paths that are related by referential constraints. This step is applied in both source and target models. The third step computes the executable mappings by performing the Cartesian product between source and target logical relations. For each pair, the technique analyses the correspondences that relate source and target elements in this pair. Note that this technique takes only one type of correspondence into account: source attribute to target attribute. Finally, the fourth step deals with the transformation of the previous executable mappings, which are represented in an intermediate nested-relational query language, into XQuery or XSLT queries.

The technique devised by Popa et al. [28] was the starting point to a number of subsequent approaches: Fuxman et al. [13] proposed the use of nested mappings that are generated by correlating Popa et al's mappings, which are called basic mappings. Basic and nested mappings can produce redundant target instances, which motivated the research on the generation of laconic/core mappings that produce target instances with no redundancy [20,36]. Mappings systems like Clio or Clip are also based on this technique [1,14].

Finally, ontology and schema matching approaches focus on the automatic generation of correspondences. Choi et al. [9], Euzenat and Shvaiko [11], and Rahm and Bernstein [30] are good surveys on the state of the art of schema and ontology matching techniques. However, it is important to notice that, in this paper, we assume that correspondences already exist, so we make no contribution to the schema or ontology matching research fields.

As a conclusion, Qin et al. [29] is suitable to generate executable mappings in scenarios in which the target model is already populated. In the rest of scenarios, the user must provide an adequate set of instance examples to ensure that the technique works properly, not only with the aforementioned examples, but also with new instances. If the set of instance examples does not capture variability well-enough, the technique may fail to cover some cases. Furthermore, the technique may suffer from overfitting, i.e., an overfitted technique works well with training examples, but may not be able to generalise to adapt to new instances from the real world.

Regarding the techniques based on nested relational models [13,20,28,36], they cannot be straightforwardly applied to ontologies due to the following differences:

- Structure: a nested relational model comprises a number of nodes, which may be nested and have a number of attributes. This model represents a tree in which there is a unique path from the root to any node or attribute. Contrarily, an ontology comprises three types of nodes: classes, data properties and object properties. This ontology represents a graph in which there may be zero, one or more paths connecting two arbitrary nodes. Note that these graphs do not have a unique root and may contain cycles.
- Instances: an instance in a nested relational model has a unique type that corresponds to an existing node. Contrarily, an instance in an ontology may have multiple types that correspond to a number of existing classes, which

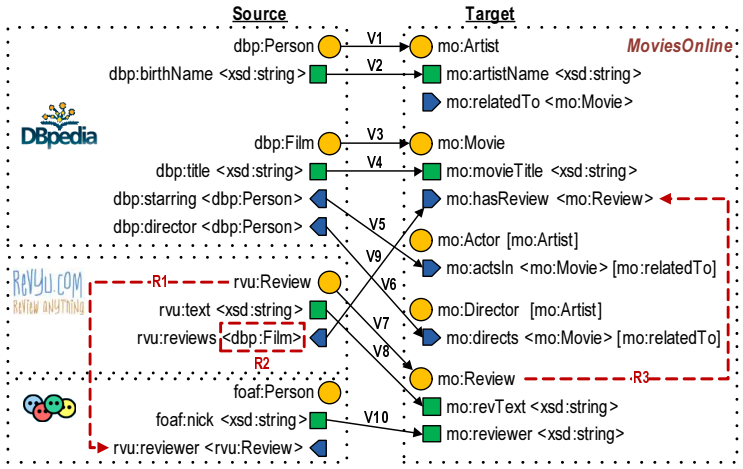


Fig. 1. Data translation scenario of our running example

need not be related by specialisation. Furthermore, in a nested relational model, an instance of a nested node exists as long as there exists an instance of the corresponding parent node. Contrarily, in an ontology, classes, data properties and object properties can be instantiated independently from each other by default.

– Queries: in the context of nested relational models, queries to perform the data translation task are encoded using XQuery or XSLT, which depend on the structure of the XML documents on which they are executed. Contrarily, in an ontology, these queries must be encoded in a language that is independent of the structure of XML documents, since the same ontology can be represented by different XML documents.

3 Mapping Generation

In this section, we present our technique to automatically generate SPARQL executable mappings based on restrictions and correspondences, which is divided into two steps: Kernel Generation and SPARQL Transformation.

To illustrate it, we use a running example in the domain of movies and reviews. Our running example builds on a fictitious video-on-demand service called Movies Online. It provides information about the movies it broadcasts and reviews of these movies. Movies Online have a number of knowledge engineers in their staff, and they devised the target ontology of Figure 1, which models movies and reviews. Instead of performing a handcrafted population of the ontology, Movies Online decided to translate information of movies and reviews from DBpedia [7] and Revyu [16], which are the source ontologies. Note that the Revyu ontology references the Friend of a Friend ontology, following the principles of Linked Data [6]. Note also that, throughout this paper, ‘dbp’, ‘rvu’,

'foaf' and 'mo' are the prefixes of DBpedia, Revyu, FOAF and Movies Online, respectively.

To represent ontologies, we use a tree-based notation in which classes are represented as circles, data properties as squares and object properties as pentagons. Furthermore, the domain of a data or object property is represented by nesting the property into the domain class, e.g., *dbp:birthName* is a data property whose domain is *dbp:Person*. The range of a data or object property is represented by '<' and '>', e.g., *dbp:starring* is an object property whose range is *dbp:Person*. Finally, class and property specialisations are represented by '[' and ']', e.g., *mo:Actor* is subclass of *mo:Artist*, and *mo:actsIn* is subproperty of *mo:relatedTo*.

In the following subsections, we present the restrictions and correspondences that our technique is able to process, and the algorithms to automatically generate SPARQL executable mappings.

3.1 Restrictions and correspondences

In this section, we present the restrictions and correspondences of our technique. Regarding restrictions, we assume that source and target ontologies pre-exist, so they contain a number of inherent restrictions, e.g., "*foaf:Person* is the domain of *foaf:nick*". Furthermore, it is possible to specify user-defined restrictions to adapt existing source and target ontologies to the requirements of a specific scenario, e.g., *R1* restricts reviews to have, at least, one reviewer.

Our technique is able to process six types of restrictions, namely:

- Domain(x, y): data or object property x is domain of class y. For instance, in Figure 1, Domain(*foaf:nick*, *foaf:Person*).
- Range(x, y): object property x is range of class y, e.g., *R2* corresponds to Range(*rvu:reviews*, *dbp:Film*).
- StrongDomain(x, y): class x is domain of data or object property y, whose minimal cardinality is one, e.g., StrongDomain(*foaf:Person*, *foaf:nick*). Note that this restriction is equivalent to a minimal cardinality of one over the domain class.
- StrongRange(x, y): class x is the range of object property y, whose minimal cardinality is one, e.g., *R1* corresponds to StrongRange(*rvu:Review*, *rvu:reviewer*). Note that this restriction is equivalent to a minimal cardinality of one over the range class.
- Subclass(x, y): class x is subclass of class y, e.g., Subclass(*mo:Director*, *mo:Artist*).
- Subproperty(x, y): data or object property x is subproperty of data or object property y, e.g., Subproperty(*mo:actsIn*, *mo:relatedTo*).

We deal with three types of correspondences, namely:

- ClassCorrespondence(x, y): instances of the y target class are copied from instances of the x source class, e.g., *V1* that corresponds to ClassCorrespondence(*dbp:Person*, *mo:Artist*).

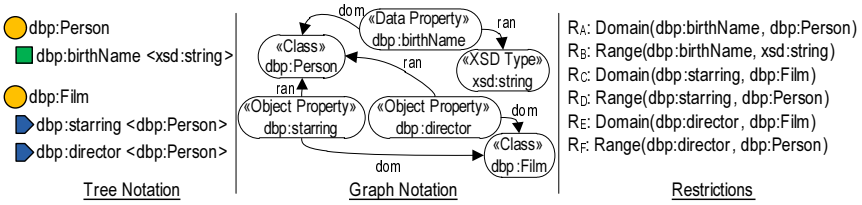


Fig. 2. An example of the restrictions of an ontology

- DataPropertyCorrespondence(x, y): instances of the y target data property are copied from instances of the x source data property, e.g., *V2* that corresponds to DataPropertyCorrespondence(*dbp:birthName*, *mo:artistName*).
- ObjectPropertyCorrespondence(x, y): instances of the y target object property are copied from instances of the x source object property, e.g., *V5* that corresponds to ObjectPropertyCorrespondence(*dbp:starring*, *mo:actsIn*).

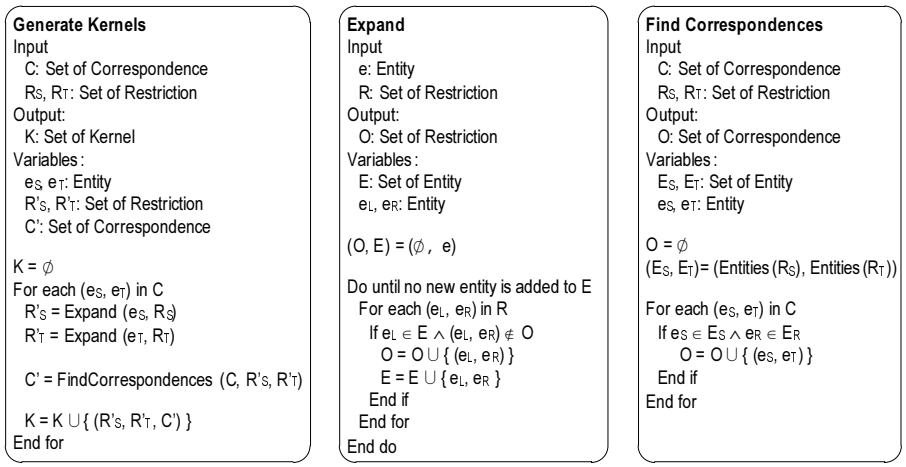
Therefore, for a particular scenario, we have a number of source and target restrictions, and a number of correspondences. Figure 2 presents a part of the source ontology in our running example that is represented in our tree-based notation, in a graph-based notation, and the restrictions that are associated to this part of the ontology, respectively. Note that each of these restrictions models a directed edge in the ontology graph that has a left and a right entities, e.g., the left entity of R_A in Figure 2 is *dbp:birthName*, and the right part is *dbp:Person*.

Finally, it is important to notice that correspondences in isolation are usually not suitable enough to perform the data translation task. For instance, assume that we use *V5* in isolation to translate instances of *dbp:starring* into *mo:actsIn*. In this context, we translate the domain and range of *dbp:starring* into the domain and range of *mo:actsIn*, respectively. Unfortunately, by doing this, we are translating *dbp:Film* into *mo:Actor*, and *dbp:Person* into *mo:Movie*, which is obviously incorrect. This is the reason why we must take both restrictions and correspondences into account to generate executable mappings.

3.2 Kernel Generation

In this section, we present the algorithm to automatically generate kernels. Intuitively, a kernel of a correspondence is a language-independent executable mapping that comprises those source restrictions, target restrictions and correspondences that we must take into account to produce coherent target data. The algorithm is shown in Figure 3(a) and it takes a set of correspondences C, and a set of source and target restrictions as input, R_S and R_T, respectively. For each correspondence, it creates a new kernel that is added to the output set K. A kernel is a three-element tuple (R'_S, R'_T, C') that comprises a set of source restrictions R'_S, a set of target restrictions R'_T, and a set of correspondences C'.

To create each kernel, it first calls the *Expand* algorithm (cf. Figure 3(b)), which is responsible for finding all restrictions that have to be explicit regarding



(a) Generate Kernels (b) Expand (c) Find Correspondences

Fig. 3. Algorithms to compute kernels

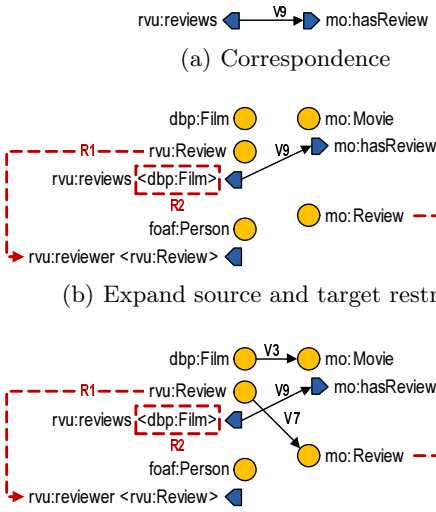
an entity. Recall that an entity may be a class, a data property or an object property. This algorithm takes an entity and a set of restrictions as input, and it finds all restrictions that are related to this input entity. First, this input entity is added to the set of entities E. Then, for each restriction, it is added to the output if it is not already present and the left entity of the restriction belongs to E. Therefore, the *Expand* algorithm computes the maximal connected subsets of restrictions out of all of the source or target restrictions. Note that it is called two times to compute source and target restrictions, respectively.

Finally, the *Find Correspondences* algorithm finds all correspondences C' that relate the entities of source and target restrictions that were found by means of algorithm *Expand* (cf. Figure 3(c)). This algorithm takes a set of correspondences, and a set of source and target restrictions as input. First, it computes all the entities contained in source and target restrictions by means of *Entities* algorithm, which is not described because it is straightforward. Then, for each correspondence, it is added to the output if the entities that are related by this correspondence belong to source and target entities.

The *Generate Kernels* algorithm terminates in $O(c(s^2 + t^2 + c))$ time in the worst case, where c is the total number of input correspondences, and s and t are the total number of input source and target restrictions, respectively. Furthermore, this algorithm generates a total number of c kernels. Note that the proof of this analysis has been omitted due to space restrictions.

Figure 4 illustrates how this algorithm works. It takes correspondence *V9* as input (cf. Figure 4(a)), which is an object property correspondence that relates *rvu:reviews* with *mo:hasReview*.

The first step (cf. Figure 4(b)) is to expand *rvu:reviews* using source restrictions. In this case, the domain of *rvu:reviews* is *rvu:Review* and the range



(a) Correspondence

(b) Expand source and target restrictions

(c) Find related correspondences

Fig. 4. Example of the Generate Kernels algorithm

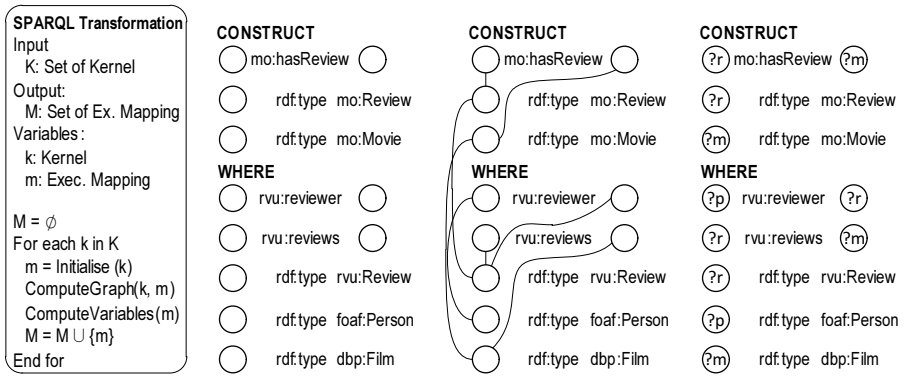
is *dbp:Film*, both restrictions are added. Then, *rvu:Review* has a minimal cardinality restriction with *rvu:reviewer*, so this restriction is added as well. Our technique continues with the expansion until no new source restriction is added.

The second step (cf. Figure 4(b)) is to expand *mo:hasReview* using target restrictions, the domain of *mo:hasReview* is *mo:Movie*, and its range is *mo:Review*, both restrictions are added. Furthermore, *mo:Review* has a minimal cardinality restriction with *mo:hasReview*, this restriction is added too and, in this case, the expansion is finished since no new target restriction is added.

Finally, our technique finds correspondences that relate the entities of both source and target restrictions (cf. Figure 4(c)). In this case, correspondences *V3* and *V7* that relate *dbp:Film* with *mo:Movie* and *rvu:Review* with *mo:Review*, respectively. Therefore, the final kernel of correspondence *V9* is shown in Figure 4(c). Note that the kernel for correspondence *V9* groups correspondences *V3* and *V7*. However, the kernel for correspondence *V3* does not group any other correspondences, i.e., the correspondence itself is a kernel. Therefore, this is the reason why our technique must process all input correspondences.

3.3 SPARQL Transformation

In this section, we present the algorithm to transform kernels into SPARQL executable mappings. A key concept of the SPARQL language is the triple pattern, which is a three-element tuple that comprises a subject, a predicate and an object. A SPARQL executable mapping is a two-element tuple (T_C, T_W) that comprises a set of triple patterns of the source and target ontologies, T_C (the CONSTRUCT clause) and T_W (the WHERE clause), respectively.



(a) Algorithm (b) Initialisation (c) Restriction Graph (d) Variables

Fig. 5. Algorithm and example of SPARQL Transformation

The transformation algorithm is shown in Figure 5(a). Note that *Initialise*, *Compute Graph* and *Compute Variables* algorithms are not described in detail due to space limitations. However, we use an example to illustrate them. The transformation algorithm takes a set of kernels K as input and it transforms each kernel into a SPARQL executable mapping, which is added to the output set M. To compute each SPARQL executable mapping, in the first step, the algorithm initialises the source triple patterns and the target triple patterns by means of *Initialise* algorithm. In this initialisation, for each class, it generates a triple pattern with an empty node, e.g., *mo:Review* in Figure 5(b); and for each data property or object property, it generates a triple pattern with two empty nodes, e.g., *mo:hasReview* in Figure 5(b). These empty nodes are assigned with variables in the following steps.

The second step computes the restriction graph that consists of relating the empty nodes previously generated by means of source and target restrictions and correspondences. This restriction graph is computed by the *Compute Graph* algorithm. Note that m is an input/output parameter of this algorithm, i.e., m is modified and returned by the *Compute Graph* algorithm. Finally, the third step computes the variables of the *CONSTRUCT* and *WHERE* clauses by using the same variable for nodes that are connected by an edge. These variables are computed by the *Compute Variables* algorithm, which also has m as an input/output parameter.

An example of the SPARQL Transformation algorithm is shown in Figures 5(b), 5(c), and 5(d). In this case, it transforms the kernel of Figure 4 into a SPARQL executable mapping. The first step (cf. Figure 5(b)) is to create a triple pattern for each entity that appears in the kernel, e.g., for *foaf:Person*, it creates a triple pattern in the *WHERE* clause that specifies a subject node of this type, but it does not assign a variable yet. Furthermore, for *rvu:reviews*, it creates a triple pattern in which subject and object nodes have blank variables. The final result of this step is a template of a SPARQL executable mapping

that comprises a set of empty nodes, which are assigned with variables in the following steps.

The second step (cf. Figure 5(c)) consists of computing the restriction graph for the triple patterns of the previous template. This step is achieved by specifying an edge between two nodes of the triple patterns if there exists a restriction or correspondence that relates those nodes. For example, since *rvu:Review* is the domain of *rvu:reviews*, there exists an edge between the subject of the triple pattern of *rvu:Review* and the subject of the triple pattern of *rvu:reviews*. Furthermore, since *V3* relates *dbp:Film* with *mo:Movie*, there is an edge between the subjects of their triple patterns. Finally, the third step (cf. Figure 5(d)) assigns the same variable for nodes that are connected by edges, e.g., *?m* is the variable used for the subject of *dbp:Film* and the object of *rvu:reviews* triple patterns. The final SPARQL executable mapping for correspondence *V9* is shown in Figure 5(d).

This algorithm terminates in $O(k((e+2)(s+t+c)+e^2))$ time in the worst case, where k is the total number of input kernels, e is the total number of entities in the source and target ontologies, c is the total number of correspondences, and s and t are the total number of source and target restrictions, respectively. Furthermore, this algorithm generates a total number of k executable mappings. Note that the proof of this analysis has been omitted due to space restrictions.

4 Implementation and evaluation

In this section, we describe the implementation and evaluation of our technique. We have implemented the algorithms described in Section 3 using Java 1.6 and the Jena framework 2.6.3. In this evaluation, we compute the time taken by our technique to automatically generate SPARQL executable mappings. Note that, to compute these times, we ran the experiments on a PC with a single 2.66 GHz Core 2 Duo CPU and 4 GB RAM, running on Windows XP (SP3) and JRE 1.6.0. Furthermore, to make these times more precise, we repeated each experiment 25 times and computed the maximum value.

We have tested our technique with ten different scenarios and the results are shown in Table 1. Note that we measure the number of classes, data properties and object properties of both source and target ontologies, the number of correspondences and source and target restrictions, the total number of generated executable mappings, and the maximum time taken to generate them.

An important issue is whether the interpretation of the correspondences is coherent with respect to the expected interpretation by experts. Regarding the example presented in Section 3 (MO in Table 1), the resulting target instances after performing the data translation task are as expected by experts. Furthermore, another scenario deals with the integration of semantic-web services with successful results. Semantic-web services try to reduce the number of limitations of (non-semantic) web services by enriching them with semantic annotations to improve their discovery and composition. Therefore, each of these services is related to one or more ontologies that describe it. In this context, we integrate

	MO	O2M	C	LP	SP	ERC	ESB	ESP	SRC	SS
Classes	9	72	728	728	728	365	365	365	365	365
Data Properties	8	41	100	100	100	100	100	100	100	100
Object Properties	8	90	726	0	0	363	0	0	363	0
Correspondences	10	11	777	414	414	414	51	51	51	51
Source restrictions	17	695	776	413	413	50	50	50	1502	413
Target restrictions	20	118	776	413	413	1502	170	463	50	50
Executable mappings	10	10	777	414	414	51	51	51	51	51
Time (seconds)	0.08	0.67	0.23	0.19	0.17	18.29	0.22	0.21	27.46	0.25

Table 1. Results of our evaluation

OWL-S as the source ontology and the Minimal Service Model (MSM) as the target ontology [26] (O2M in Table 1). Thanks to our technique, we are able to automatically populate iServe, which comprises a number of semantic-web services represented using MSM, based on a number of semantic-web services represented using OWL-S [26].

Finally, we have tested our technique with a benchmark that provides eight data translation patterns, which are inspired by real-world data translation problems [34]. In this benchmark, the data translation task is performed by a set of queries that are automatically instantiated from a number of query templates that have been devised by experts. These patterns are the following:

- Copy (C): each class, data property and object property source instance is copied into a class, data property or object property target instance.
- Lift Properties (LP): the data properties of a set of subclasses in the source are moved to a common superclass in the target.
- Sink Properties (SP): the data properties of a superclass in the source are moved to a number of subclasses in the target.
- Extract Related Classes (ERC): the data properties of a class in the source are grouped into a number of new classes in the target, which are related to the original one by a number of object properties.
- Extract Subclasses (ESB): a class in the source is split into several subclasses in the target and data properties are distributed amongst them.
- Extract Superclasses (ESP): a class in the source is split into several superclasses in the target, and data properties are distributed amongst them.
- Simplify Related Classes (SRC): source classes, which are related by a set of object properties, are transformed into a target class that aggregates them.
- Simplify Specialisation (SS): a set of specialised classes in the source are flattened into a single target class.

In all of these patterns, our technique generates the same target instances as the benchmark. Therefore, we conclude that the interpretation of the correspondences is coherent with the expected results by experts in these patterns.

5 Conclusions

In this paper, we present a technique to automatically generate SPARQL executable mappings. Our technique has been devised for semantic-web ontologies that are represented using the OWL ontology language. It is based on correspondences and source and target restrictions, and it generates SPARQL executable mappings in two steps: kernel generation and SPARQL transformation.

As a conclusion, we have devised a technique that, building on our experiments, seems promising enough for real-world scenarios: we evaluate it over ten scenarios and, in our evaluation results, executable mappings are generated in less than thirty seconds in the worst case. Furthermore, we also test the interpretation of correspondences that our algorithm assumes, and the result is that it is coherent with the results expected by experts.

The original contributions of our technique are as follows: 1) Instead of relying on instance examples of the target ontology, we automatically generate executable mappings based on restrictions and correspondences, which makes it appealing in many practical cases. 2) We have devised an algorithm to generate kernels, which are language-independent executable mappings that makes restrictions and correspondences explicit. 3) We have devised an algorithm to transform kernels into SPARQL executable mappings by linking the variables of triple patterns in these mappings.

References

1. B. Alexe, L. Chiticariu, R. J. Miller, D. Pepper, and W. C. Tan. Muse: a system for understanding and designing mappings. In *SIGMOD*, pages 1281–1284, 2008.
2. B. Alexe, W. C. Tan, and Y. Velegrakis. STBenchmark: towards a benchmark for mapping systems. *PVLDB*, 1(1):230–244, 2008.
3. G. Antoniou and F. van Harmelen. *A Semantic Web Primer, 2nd Edition*. The MIT Press, 2008.
4. P. A. Bernstein and L. M. Haas. Information integration in the enterprise. *Commun. ACM*, 51(9):72–79, 2008.
5. P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD*, pages 1–12, 2007.
6. C. Bizer. The emerging web of linked data. *IEEE Int. Sys.*, 2009.
7. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - a crystallization point for the web of data. *J. Web Sem.*, 2009.
8. C. Bizer and A. Schultz. The R2R framework: Publishing and discovering mappings on the web. In *COLD*, 2010.
9. N. Choi, I.-Y. Song, and H. Han. A survey on ontology mapping. *SIGMOD Record*, 35(3):34–41, 2006.
10. J. Euzenat, A. Polleres, and F. Scharffe. Processing ontology alignments with SPARQL. In *CISIS*, pages 913–917, 2008.
11. J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag, 2007.
12. 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.
13. A. Fuxman, M. A. Hernández, C. T. H. Ho, R. J. Miller, P. Papotti, and L. Popa. Nested mappings: Schema mapping reloaded. In *VLDB*, pages 67–78, 2006.

14. 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.
15. A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.
16. T. Heath and E. Motta. Revyu: Linking reviews and ratings into the web of data. *J. Web Sem.*, 6(4):266–273, 2008.
17. G. Karvounarakis, A. Magkanaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, and K. Tolle. Querying the semantic web with RQL. *Computer Networks*, 42(5):617–640, 2003.
18. D. Kensche, C. Quix, Y. Li, and M. Jarke. Generic schema mappings. In *ER*, pages 132–148, 2007.
19. M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.
20. G. Mecca, P. Papotti, and S. Raunich. Core schema mappings. In *SIGMOD*, pages 655–668, 2009.
21. B. Motik, I. Horrocks, and U. Sattler. Bridging the gap between OWL and relational databases. *J. Web Sem.*, 7(2):74–89, 2009.
22. N. F. Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.
23. N. F. Noy and M. C. A. Klein. Ontology evolution: Not the same as schema evolution. *Knowl. Inf. Syst.*, 6(4):428–440, 2004.
24. P. Papotti and R. Torlone. Schema exchange: A template-based approach to data and metadata translation. In *ER*, pages 323–337, 2007.
25. F. S. Parreiras, S. Staab, S. Schenk, and A. Winter. Model driven specification of ontology translations. In *ER*, pages 484–497, 2008.
26. C. Pedrinaci, D. Liu, M. Maleshkova, D. Lambert, J. Kopecký, and J. Domingue. iServe: a linked services publishing platform. In *ORES*, 2010.
27. M. Petropoulos, A. Deutsch, Y. Papakonstantinou, and Y. Katsis. Exporting and interactively querying web service-accessed sources: The CLIDE system. *ACM Trans. Database Syst.*, 32(4), 2007.
28. L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating web data. In *VLDB*, pages 598–609, 2002.
29. H. Qin, D. Dou, and P. LePendu. Discovering executable semantic mappings between ontologies. In *OTM*, pages 832–849, 2007.
30. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.
31. C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. Mosto: Generating sparql executable mappings between ontologies. In *ER*, 2011.
32. C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. On using database techniques for generating ontology mappings. In *SWWS*, 2011.
33. C. R. Rivero, I. Hernández, D. Ruiz, and R. Corchuelo. A reference architecture for building semantic-web mediators. In *IWSSA*, 2011.
34. C. R. Rivero, D. Ruiz, and R. Corchuelo. On benchmarking data translation systems for semantic-web ontologies (Tech. Report). `http://tdg-seville.info/Download.ashx?id=205`.
35. N. Shadbolt, T. Berners-Lee, and W. Hall. The semantic web revisited. *IEEE Int. Sys.*, 21(3):96–101, 2006.
36. B. ten Cate, L. Chiticariu, P. G. Kolaitis, and W. C. Tan. Laconic schema mappings: Computing the core with SQL queries. *PVLDB*, 2(1):1006–1017, 2009.
37. C. Yu and L. Popa. Constraint-based XML query rewriting for data integration. In *SIGMOD*, pages 371–382, 2004.