Ontology Mapping and SPARQL Rewriting for Querying Federated RDF Data Sources

Konstantinos Makris[1], Nektarios Gioldasis[1], Nikos Bikakis[2], and Stavros Christodoulakis[1]

[1] TUC/MUSIC Lab, Technical University of Crete, Greece
{makris, nektarios, stavros}@ced.tuc.gr
[2] KDBS Lab, National Technical University of Athens, Greece
bikakis@dblab.ntua.gr

Abstract. The web of data consists of distributed, diverse (in terms of schema adopted), and large RDF datasets. In this paper we present a SPARQL query rewriting method which can be used to achieve interoperability in semantic information retrieval and/or knowledge discovery processes over interconnected RDF data sources. Formal mappings between different overlapping ontologies are exploited in order to rewrite initial user SPARQL queries, so that they can be evaluated over different RDF data sources on different sites. The proposed environment is utilized by an ontology-based mediator system, which we have developed in order to provide data integration within the Semantic Web environment.

1 Introduction

Information access from federated resources, distributed over the internet, is extremely important for Semantic Web applications and end users. In this paper, we introduce an environment that provides transparent access to federated RDF data sources. A set of mappings between OWL ontologies is defined in order to integrate the access to the federated resources. The queries of the Semantic Web users are expressed in SPARQL over an OWL ontology. The mappings are used to transform the original SPARQL query, through rewriting, to a set of SPARQL queries which are used to access the federated RDF data sources.

We focus on the following research issues: (a) determination of the ontology mapping types, which can be used in the context of SPARQL query rewriting, (b) modeling of the mappings between a source ontology and the target ontologies, (c) rewriting of the SPARQL queries posed over a source ontology in terms of the target ontologies.

Ontology mapping in general is a task that has received tremendous attention by the Semantic Web community [6], [7], [3]. In this work we are only interested in the specification of the kinds of mappings between OWL ontologies, which can be exploited by the SPARQL query rewriting process. Despite the extensive studies, to the best of our knowledge, there is no work addressing the problem of ontology mappings in the context of SPARQL query rewriting.

In the field of query rewriting, limited studies examine the problem of posing a SPARQL query over different RDF datasets. An approach [4] which comes closer to ours, with some of its parts based on a preliminary description of our work [8], proposes a method that exploits transformations between RDF structures (i.e. graphs) in order to perform SPARQL query rewriting. Compared to our approach, where mappings rely on Description Logic semantics, this choice seems to restrict the mappings expressivity and also the supported query types.

Paper Outline. Section 2 describes the mapping model which has been developed in order to express mappings between OWL ontologies. Sections 3 and 4 provide an overview of the SPARQL query rewriting process. Section 5 presents an illustrative query rewriting example, while Sect. 6 concludes our work.

2 Ontology Mapping Model

In order for SPARQL queries posed over a source ontology to be rewritten in terms of a target ontology, mappings between the source and target ontologies should be specified. In Fig. 1, we show the structure of two overlapping ontologies. The source ontology describes a store, while the target ontology describes a bookstore. Between these two ontologies, various kinds of mappings can be identified.

In this section we introduce a model for the expression of mappings between OWL ontologies in the context of SPARQL query rewriting. In this context, some mapping types may not be useful, and therefore not examined here, due to the lack of specific features (e.g. aggregates) in the current specification of SPARQL. Such mapping types are described in [5] and some of them could be useful for post-processing the query results but not during the query rewriting and query answering process.

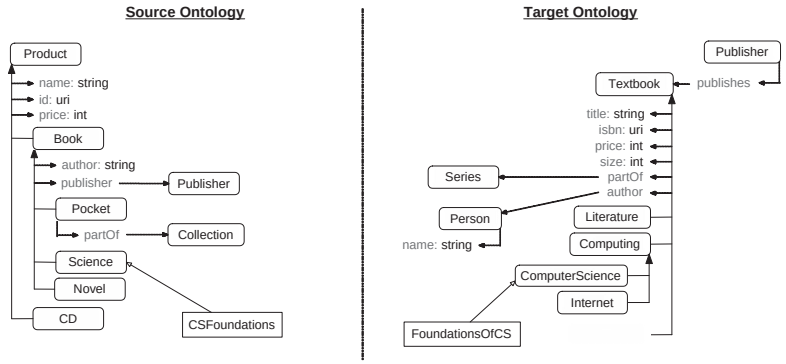

Fig. 1. Semantically Overlapping Ontologies. The rounded corner boxes represent classes. They are followed by their object/datatype properties. The rectangle boxes represent individuals. The arrows represent relationships between basic constructs.

2.1 Abstract Syntax and Semantics

The basic constructs of OWL are the classes c, the object properties op, the datatype properties dp and the individuals i. In order to define the mapping types which are useful for the rewriting process, we use Description Logics (DL). We treat OWL classes as DL concepts, OWL properties as DL roles and OWL individuals as DL individuals. Following our convension, let C, D be OWL classes (treated as atomic concepts), R, S be OWL object properties (treated as atomic roles) and K, L be OWL datatype properties (treated as atomic roles). Similarly, let a, b, c, v_{op} be individuals and v_{dp} be a data value.

An interpretation $\mathcal{I}$ consists of a non-empty set $\Delta^{\mathcal{I}}$ (the domain of the interpretation) and an interpretation function, which assigns to every atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, to every atomic role B a binary relation $B^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and to every individual k an element $k^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ (based on [1]).

In Tables 1, 2 and 3 we present the set of class and property constructors that we use for the mapping definition. In these tables we introduce some new constructors (preceded with asterisk) which should not be confused with the basic DL constructors defined in [1]. In addition to the constructors, a DL knowledge base consists of common assertional axioms, i.e. inclusion ($\sqsubseteq$, $\sqsupseteq$) and equality ($\equiv$). The semantics of concept/role inclusion and equality are available in [1].

Table 1. Class constructors used in the definition of mappings.

Name	Syntax	Semantics
Intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
*Class Restriction	$C.(R\ \overline{\texttt{cp}}\ v_{op})$	$\{\alpha \in C^{\mathcal{I}} \mid \exists b.\ (\alpha, b) \in R^{\mathcal{I}} \wedge b\ \overline{\texttt{cp}}\ v_{op}\}$
	$C.(K\ \texttt{cp}\ v_{dp})$	$\{\alpha \in C^{\mathcal{I}} \mid \exists b.\ (\alpha, b) \in K^{\mathcal{I}} \wedge b\ \texttt{cp}\ v_{dp}\}$
	$C.(R\ \overline{\texttt{cp}}\ S)$	$\{\alpha \in C^{\mathcal{I}} \mid \exists b, \exists c.\ (\alpha, b) \in R^{\mathcal{I}} \wedge (\alpha, c) \in S^{\mathcal{I}} \wedge b\ \overline{\texttt{cp}}\ c\}$
	$C.(K\ \texttt{cp}\ L)$	$\{\alpha \in C^{\mathcal{I}} \mid \exists b, \exists c.\ (\alpha, b) \in K^{\mathcal{I}} \wedge (\alpha, c) \in L^{\mathcal{I}} \wedge b\ \texttt{cp}\ c\}$
	$\overline{\texttt{cp}} \in \{\neq, =\}$, $\texttt{cp} \in \{\neq, =, \leq, \geq, <, >\}$	

Definition 1 (Class Expression). *A class expression is a class or any complex expression between two or more classes, using union or intersection operations. A class expression is denoted as CE and is defined recursively in (1). Any class expression can be restricted to the values of one or more object property expressions OPE (Definition 2) or datatype property expressions DPE (Definition 3), using the comparators $\overline{\texttt{cp}} \in \{\neq, =\}$ and $\texttt{cp} \in \{\neq, =, \leq, \geq, <, >\}$, respectively. Moreover, it is possible for a class expression to be restricted on a set of individuals having object/datatype property values with a specific relationship between them.*

$$\begin{aligned} CE := c \mid CE \sqcap CE \mid CE \sqcup CE \mid CE.(OPE\ \overline{\texttt{cp}}\ v_{op}) \mid CE.(DPE\ \texttt{cp}\ v_{dp}) \\ \mid CE.(OPE_1\ \overline{\texttt{cp}}\ OPE_2) \mid CE.(DPE_1\ \texttt{cp}\ DPE_2) \end{aligned} \tag{1}$$

Table 2. Object property constructors used in the definition of mappings.

Name	Syntax	Semantics
Intersection	$R \sqcap S$	$R^{\mathcal{I}} \cap S^{\mathcal{I}}$
Union	$R \sqcup S$	$R^{\mathcal{I}} \cup S^{\mathcal{I}}$
Composition	$R \circ S$	$\{(\alpha, c) \mid \exists b.\ (\alpha, b) \in R^{\mathcal{I}} \wedge (b, c) \in S^{\mathcal{I}}\}$
*Inverse	$inv(R)$	$\{(b, \alpha) \mid (\alpha, b) \in R^{\mathcal{I}}\}$
*Domain Restriction	$R.domain(C)$	$\{(\alpha, b) \mid (\alpha, b) \in R^{\mathcal{I}} \wedge \alpha \in C^{\mathcal{I}}\}$
*Range Restriction	$R.range(C)$	$\{(\alpha, b) \mid (\alpha, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$

Definition 2 (Object Property Expression). *An object property expression is an object property or any complex expression between two or more object properties, using composition, union or intersection operations. An object property expression is denoted as OPE and is defined recursively in (2). Inverse property operations are possible to appear inside an object property expression. Any object property expression can be restricted on its domain and/or range using a class expression to define the applied restrictions.*

$$\begin{aligned} OPE := op \mid OPE \circ OPE \mid OPE \sqcap OPE \mid OPE \sqcup OPE \mid inv(OPE) \\ \mid OPE.domain(CE) \mid OPE.range(CE) \end{aligned} \tag{2}$$

Table 3. Datatype property constructors used in the definition of mappings.

Name	Syntax	Semantics
Intersection	$K \sqcap L$	$K^{\mathcal{I}} \cap L^{\mathcal{I}}$
Union	$K \sqcup L$	$K^{\mathcal{I}} \cup L^{\mathcal{I}}$
Composition	$R \circ K$	$\{(\alpha, c) \mid \exists b.\ (\alpha, b) \in R^{\mathcal{I}} \wedge (b, c) \in K^{\mathcal{I}}\}$
*Domain Restriction	$K.domain(C)$	$\{(\alpha, b) \mid (\alpha, b) \in K^{\mathcal{I}} \wedge \alpha \in C^{\mathcal{I}}\}$
*Range Restriction	$K.range(\texttt{cp}\ v_{dp})$ $\texttt{cp} \in \{\neq, =, \leq, \geq, <, >\}$	$\{(\alpha, b) \mid (\alpha, b) \in K^{\mathcal{I}} \wedge b\ \texttt{cp}\ v_{dp}\}$

Definition 3 (Datatype Property Expression). *A datatype property expression is a datatype property or any complex expression between object and datatype properties using the composition operation, or between two or more datatype properties, using union or intersection operations. A datatype property expression is denoted as DPE and is defined recursively in (3). Any datatype property expression can be restricted on its domain values using a class expression to define the applied restrictions. In addition, the range values of a datatype property expression can be restricted on various data values v_{dp}, using a comparator $\texttt{cp} \in \{\neq, =, \leq, \geq, <, >\}$.*

$$\begin{aligned} DPE := dp \mid OPE \circ DPE \mid DPE \sqcap DPE \mid DPE \sqcup DPE \\ \mid DPE.domain(CE) \mid DPE.range(\texttt{cp}\ v_{dp}) \end{aligned} \tag{3}$$

2.2 Ontology Mapping Types

In this section we present a rich set of 1:N cardinality mapping types, in order for mappings of these types to be used for the rewriting of a SPARQL query.

Class mapping (c_s ***rel*** CE_t, ***rel*** $:= \equiv \mid \sqsubseteq \mid \sqsupseteq$)**.** A class from a source ontology s can be mapped to a class expression from a target ontology t.

Object property mapping (op_s ***rel*** OPE_t, ***rel*** $:= \equiv \mid \sqsubseteq \mid \sqsupseteq$)**.** An object property from a source ontology s can be mapped to an object property expression from a target ontology t.

Datatype property mapping (dp_s ***rel*** DPE_t, ***rel*** $:= \equiv \mid \sqsubseteq \mid \sqsupseteq$)**.** A datatype property from a source ontology s can be mapped to a datatype property expression from a target ontology t.

We note here that the equivalence/subsumption between two different properties or between a property and a property expression, denotes equivalence/ subsumption between the domains and ranges of those properties or property expressions. The proof for the above statement is available in [9].

Individual mapping ($i_s \equiv i_t$)**.** An individual from a source ontology s can be mapped to an individual from a target ontology t.

3 SPARQL Query Rewriting Overview

Query rewriting is done by exploiting a predefined set of mappings which is based on the different mapping types described in Sect. 2.2. The SPARQL query rewriting process lies in the query's graph pattern rewriting. The rewritten query is produced by replacing the rewritten graph pattern to the initial query's graph pattern. Consequently the rewriting process is independent of the query type (i.e. `SELECT`, `CONSTRUCT`, `ASK`, `DESCRIBE`) and the SPARQL solution sequence modifiers (i.e. `ORDER BY`, `DISTINCT`, `REDUCED`, `LIMIT`, `OFFSET`).

Graph pattern operators (`AND`, `UNION`, `OPTIONAL`, `FILTER`) remain the same during the rewriting process. Variables, literal constants, operators and built-in functions appearing in a `FILTER` expression, remain also the same. We use 1:1 cardinality mappings for the rewriting of IRIs which may appear inside a `FILTER` expression.

Since a graph pattern consists basically of triple patterns, the most important part of a SPARQL query rewriting is the triple pattern rewriting. Triple patterns may refer either to data (e.g. relationships between instances) or schema (e.g. relationships between classes and/or properties) information. Due to the inability in handling all the different triple pattern types in the same manner, we distinguish triple patterns into Data Triple Patterns (Definition 4) and Schema Triple Patterns (Definition 5). Triple patterns having a variable on their predicate part are not taken into consideration, since they can deal either with data or schema info.

Let L be the set of literals, V the set of variables, I the set of IRIs, I_{RDF} the set containing the IRIs of the RDF vocabulary, I_{RDFS} the set containing the IRIs of the RDF Schema vocabulary and I_{OWL} the set containing the IRIs of the OWL vocabulary.

Definition 4 (Data Triple Pattern). *The triple patterns that only apply to data and not schema info are considered to be Data Triple Patterns, e.g. (?x, rdf:type, src:Product). A tuple $t \in DTP$ (Data Triple Pattern set - shown in (4)) is a Data Triple Pattern.*

$$DTP = (I' \cup L \cup V) \times (I' \cup \{rdf:type, owl:sameAs\}) \times (I' \cup L \cup V) \tag{4}$$

$$I' = I - I_{RDF} - I_{RDFS} - I_{OWL} \tag{5}$$

Definition 5 (Schema Triple Pattern). *The triple patterns that only apply to schema and not data info are considered to be Schema Triple Patterns, e.g. (?x, rdfs:subClassOf, src:Product). A tuple $t \in STP$ (Schema Triple Pattern set - shown in (6)) is a Schema Triple Pattern.*

$$STP = \big((I \cup L \cup V) \times I \times (I \cup L \cup V)\big) - DTP \tag{6}$$

Since a triple pattern consists of three parts (subject, predicate, object), in order to rewrite it we have to follow a three-step procedure by exploiting mappings for each triple pattern's part. The rewriting procedure follows a strict order. Firstly, the triple pattern is rewritten using the mapping which has been specified for its predicate part, resulting to a graph pattern which may contain one or more triple patterns. Then, the resulted graph pattern is rewritten triple pattern by triple pattern, using the mappings of the triple patterns' object parts. Finally, the same procedure is repeated for the triple patterns' subject parts. Variables, blank nodes, literal constants and RDF/RDFS/OWL IRIs which may appear in a triple pattern part do not affect the rewriting procedure. This means that the variables of the initial query appear also in the rewritten query.

We note that the rewriting of a triple pattern, is not dependent on mapping relationships (i.e. equivalence, subsumption). These relationships, affect only the evaluation results of the rewritten query over the target ontology. The complete set of functions that perform triple pattern rewriting, the algorithms that perform graph pattern rewriting, as well as a set of examples is available in [9].

4 Data Triple Pattern Rewriting

In this section, we provide an overview of the set of functions that perform Data Triple Pattern rewriting based on a set of mappings. These functions are actually rewriting steps in the process of Data Triple Pattern rewriting. We have formally shown [9] that each rewriting step that is performed, in order to rewrite a triple pattern, is semantics preserving, in the sense that it preserves the mapping semantics. The complete set of functions, including those that perform Schema Triple Pattern rewriting, as well as a set of examples is available in [9].

Let $\mathcal{D}^x_y$ be the function that produces the resulted form of a Data Triple Pattern, after being rewritten by $x \in \{s, p, o\}$ (subject, predicate, object). The subscript $y \in \{c, op, dp, i\}$ shows the type of x (e.g. class, object property, etc.). The function $\mathcal{D}$ takes as arguments a Data Triple Pattern t, as well as a mapping μ. In what follows, we use the subscripts s and t to denote that a class, a property or an individual belongs to the source or target ontology, respectively.

4.1 Rewriting by Triple Pattern's Predicate Part

In order to rewrite a Data Triple Pattern by its predicate part only property mappings can be used, since a class or an individual cannot appear on a triple pattern's predicate part.

Rewriting based on object property mapping. Let op_s be an object property from the source ontology which is mapped to an object property expression from the target. Having a Data Triple Pattern $t = (subject, op_s, object)$ with op_s in its predicate part and anything in its subject and object parts, we can rewrite it by its predicate part, using a predefined mapping μ and the function (7).

$$\mathcal{D}^p_{op}(t, \mu) = \begin{cases} (subject, op_t, object) & \text{if } \mu: \ op_s \rightarrow op_t \\ \\ \mathcal{D}^p_{op}(t_1, \mu_1) \text{ AND} & \text{if } \mu: \ op_s \rightarrow op_{t1} \sqcap op_{t2}, \\ \mathcal{D}^p_{op}(t_2, \mu_2) & \text{where } t_1 = (subject, op_{t1}, object), \\ & t_2 = (subject, op_{t2}, object), \\ & \mu_1: \ op_{t1} \equiv OPE_{t1}, \ \mu_2: \ op_{t2} \equiv OPE_{t2} \\ \\ \mathcal{D}^p_{op}(t_1, \mu_1) & \text{if } \mu: \ op_s \rightarrow inv(op_t), \\ & \text{where } t_1 = (object, op_t, subject) \\ & \text{and } \mu_1: \ op_t \equiv OPE_t \end{cases} \tag{7}$$

Rewriting based on datatype property mapping. Let dp_s be a datatype property from the source ontology which is mapped to a datatype property expression from the target. Having a Data Triple Pattern $t = (subject, dp_s, object)$ with dp_s in its predicate part and anything in its subject and object parts, we can rewrite it by its predicate part, using a predefined mapping μ and the function (8).

$$\mathcal{D}^p_{dp}(t, \mu) = \begin{cases} (subject, dp_t, object) & \text{if } \mu: \ dp_s \rightarrow dp_t \\ \\ \mathcal{D}^p_{op}(t_1, \mu_1) \text{ AND} & \text{if } \mu: \ dp_s \rightarrow op_t \circ dp_t, \\ \mathcal{D}^p_{dp}(t_2, \mu_2) & \text{where } t_1 = (subject, op_t, ?var), \\ & t_2 = (?var, dp_t, object), \\ & \mu_1: \ op_t \equiv OPE_t, \ \mu_2: \ dp_t \equiv DPE_t \\ \\ \mathcal{D}^p_{dp}(t_1, \mu_1) & \text{if } \mu: \ dp_s \rightarrow dp_t.range(\texttt{cp} \ v_{dp}), \\ \text{FILTER}(object \ \texttt{cp} \ v_{dp}) & \text{where } \texttt{cp} \in \{\neq, =, \leq, \geq, <, >\}, \\ & v_{dp} = data \ value, \\ & \text{and } t_1 = (subject, dp_t, object), \\ & \mu_1: \ dp_t \equiv DPE_t \end{cases} \tag{8}$$

4.2 Rewriting by Triple Pattern's Object Part

When a property appears on the object part of a triple pattern, we conclude that the triple pattern deals with schema info, as there is no way for a non RDF/ RDFS/OWL IRI to appear at the same time in the triple pattern's predicate part. Similarly, in case that a class appears on a triple pattern's object part, the only factor which can be used to determine the triple pattern's type (Data or Schema Triple Pattern), is whether the RDF property $rdf : type$ appears on the predicate part or not. Thus, the only cases mentioned for the rewriting of a Data Triple Pattern by its object part concern individuals, as well as classes with the precondition that the RDF property $rdf : type$ appears on the triple pattern's predicate part at the same time.

Rewriting based on class mapping. Let c_s be a class from the source ontology which is mapped to a class expression from the target ontology. Having a Data Triple Pattern $t = (subject, rdf : type, c_s)$ with the class c_s in its object part, the RDF property $rdf : type$ in its predicate and anything in its subject part, we can rewrite it by its object part, using a predefined mapping μ and the function (9).

$$\mathcal{D}_c^o(t,\mu) = \begin{cases} (subject, rdf : type, c_t) & \text{if } \mu :\ c_s \rightarrow c_t \\ \\ \mathcal{D}_c^o(t_1,\mu_1) \text{ UNION} & \text{if } \mu :\ c_s \rightarrow c_{t1} \sqcup c_{t2}, \\ \mathcal{D}_c^o(t_2,\mu_2) & \text{where } t_1 = (subject, rdf : type, c_{t1}), \\ & t_2 = (subject, rdf : type, c_{t2}), \\ & \mu_1 :\ c_{t1} \equiv CE_{t1},\ \mu_2 :\ c_{t2} \equiv CE_{t2} \\ \\ \mathcal{D}_c^o(t_1,\mu_1) \text{ AND} & \text{if } \mu :\ c_s \rightarrow c_t.(dp_t\ \texttt{cp}\ v_{dp}), \\ \mathcal{D}_{dp}^p(t_2,\mu_2) & \text{where } \texttt{cp} \in \{\neq, =, \leq, \geq, <, >\}, \\ \text{FILTER}(?var\ \texttt{cp}\ v_{dp}) & v_{dp} = data\ value, \\ & t_1 = (subject, rdf : type, c_t), \\ & t_2 = (subject, dp_t, ?var), \\ & \mu_1 :\ c_t \equiv CE_t,\ \mu_2 :\ dp_t \equiv DPE_t \end{cases} \tag{9}$$

Rewriting based on individual mapping. Let i_s be an individual from the source ontology which is mapped to an individual i_t from the target ontology. Having a Data Triple Pattern $t = (subject, predicate, i_s)$ with i_s in its object part and anything in its predicate and subject parts, we can rewrite it by its object part, using a predefined mapping μ and the function (10).

$$\mathcal{D}_i^o(t,\mu) = (subject, predicate, i_t) \quad \text{if } \mu :\ i_s \equiv i_t \tag{10}$$

4.3 Rewriting by Triple Pattern's Subject Part

Generally, when a class or property appears on the subject part of a triple pattern we conclude that the triple pattern involves schema info, as there is no way for a non RDF/RDFS/OWL IRI to appear at the same time in the triple pattern's predicate part. Thus, the only case mentioned for the rewriting of a Data Triple Pattern by its subject part concerns individuals.

Rewriting based on individual mapping. Let i_s be an individual from the source ontology which is mapped to an individual i_t from the target ontology. Having a Data Triple Pattern $t = (i_s, predicate, object)$ with i_s in its subject part and anything in its predicate and object parts, we can rewrite it by its subject part, using a predefined mapping μ and the function (11).

$$\mathcal{D}_i^s(t, \mu) = (i_t, predicate, object) \quad \text{if } \mu : \ i_s \equiv i_t \tag{11}$$

5 Query Rewriting Example

Consider the query posed over the source ontology of Fig. 1: "Return at most 20 titles of pocket-sized scientific books and optionally their authors. The results should be formed in ascending order based on the title value.". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
SELECT ?name ?author
WHERE{ ?x src:name ?name. ?x rdf:type src:Science.
    ?x rdf:type src:Pocket. OPTIONAL{?x src:author ?author}
} ORDER BY ?name LIMIT 20
```

Let the available predefined mappings be m_1, m_2, m_3 and m_4 (presented below). For their representation we use the abstract syntax presented in Sect. 2.2.

$$m_1 : \ src : name \ \sqsupseteq \ trg : title, m_2 : \ src : author \ \equiv \ trg : author \ \circ \ trg : name,$$
$$m_3 : \ src : Science \ \equiv \ trg : ComputerScience \ \sqcup \ trg : Mathematics,$$
$$m_4 : \ src : Pocket \ \equiv \ trg : Textbook.(trg : size \leq 14)$$

In order to rewrite the initial query's graph pattern GP, every triple pattern of GP should be rewritten by its predicate, object and subject part, as described in Sect. 3. The rewriting procedure is shown in Fig. 2. Firstly, the initial graph pattern is rewritten triple pattern by triple pattern using the mappings of the triple patterns' predicate parts. Triple patterns containing an RDF/RDFS/OWL property on their predicate part do not result in modifications.

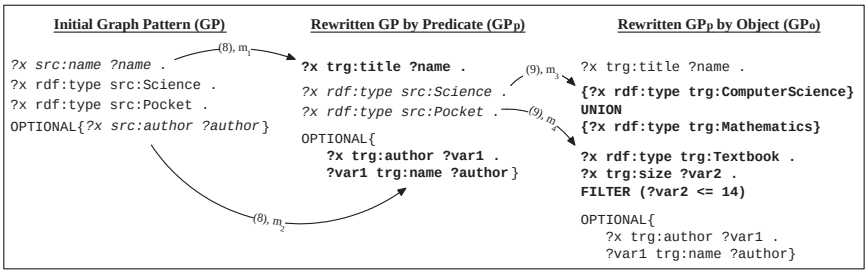

Fig. 2. Triple pattern rewriting by predicate and object part. The parameters upon the arrows denote the rewriting function and the mapping used by the rewriting process.

Similarly, the resulted graph pattern GP_p is rewritten triple pattern by triple pattern using the mappings of the triple patterns' object parts. Triple patterns containing variables on their object part do not result in modifications. The same procedure is repeated for the resulted graph pattern GP_o, using the mappings of the triple patterns' subject parts. However, the graph pattern remains the same since every triple pattern of GP_o contains a variable in its subject part.

Finally, the rewritten SPARQL query, in terms of the target ontology of Fig. 1, is provided by replacing the initial query's graph pattern GP with GP_o.

6 Conclusion

In this paper we presented a method that exploits ontology mappings in order to rewrite SPARQL queries posed over a source ontology, in terms of a target ontology. For this purpose, we also introduced a formal model for describing ontology mappings which can be used in the rewriting process. The proposed SPARQL query rewriting method has been implemented as part of an ontology-based mediator system developed in the TUC/MUSIC Lab.

Our current research focuses on evaluating the system performance, exploiting advanced reasoning techniques during the query rewriting, and developing methodologies for the optimization of the query mediation process. Moreover, this work is going to be integrated with our XS2OWL [10] and SPARQL2XQuery [2] frameworks, in order to allow access to heterogeneous web repositories.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook. Cambridge University Press (2003)
2. Bikakis, N., Gioldasis, N., Tsinaraki, C., Christodoulakis, S.: Querying xml data with sparql. In: DEXA. LNCS, vol. 5690, pp. 372–381. Springer (2009)
3. Choi, N., Song, I.Y., Han, H.: A survey on ontology mapping. SIGMOD Record 35(3), 34–41 (2006)
4. Correndo, G., Salvadores, M., Millard, I., Glaser, H., Shadbolt, N.: Sparql query rewriting for implementing data integration over linked data. In: 1st International Workshop on Data Semantics (2010)
5. Euzenat, J., Polleres, A., Scharffe, F.: Processing ontology alignments with sparql. In: CISIS. pp. 913–917 (2008)
6. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer, Heidelberg (2007)
7. Kalfoglou, Y., Schorlemmer, W.M.: Ontology mapping: The state of the art. In: Semantic Interoperability and Integration. Dagstuhl Seminar Proceedings (2005)
8. Makris, K., Bikakis, N., Gioldasis, N., Tsinaraki, C., Christodoulakis, S.: Towards a mediator based on owl and sparql. In: WSKS. LNCS, vol. 5736 (2009)
9. Makris, K., Gioldasis, N., Bikakis, N., Christodoulakis, S.: Sparql rewriting for query mediation over mapped ontologies. Tech. rep., Technical University of Crete (2010), http://www.music.tuc.gr/reports/SPARQLREWRITING.PDF
10. Tsinaraki, C., Christodoulakis, S.: Interoperability of xml schema applications with owl domain knowledge and semantic web tools. In: ODBASE. LNCS, vol. 4803. Springer (2007)