Querying Semantic Web Resources Using TRIPLE Views

Zoltán Miklós[1], Gustaf Neumann[1], Uwe Zdun[1], and Michael Sintek[2]

[1] Vienna University of Economics, Department for Information Systems
{zoltan.miklos, gustaf.neumann, uwe.zdun}@wu-wien.ac.at
[2] DFKI GmbH, Kaiserslautern, sintek@dfki.de

Abstract. Resources on the Semantic Web are described by metadata based on some formal or informal ontology. It is a common situation that casual users are not familiar with a domain ontology in detail. This makes it difficult for such users (or their user tools) to formulate queries to find the relevant resources. Users consider the resources in their specific context, so the most straightforward solution is to formulate queries in an ontology that corresponds to a user-specific view. We present an approach based on multiple views expressed in ontologies simpler than the domain ontology. This allows users to query heterogeneous data repositories in terms of multiple, relatively simple, view ontologies. Ontology developers can define such view ontologies and the corresponding mapping rules. These ontologies are represented in Semantic Web ontology languages such as RDFS, DAML+OIL, or OWL. We present our approach with examples from the e-learning domain using the Semantic Web query and transformation language TRIPLE.

1 Introduction

On the Semantic Web, resources are annotated with ontology-based metadata. Ontologies are formal, explicit specifications of a shared conceptualization. Ontologies play a central role on the Semantic Web because they represent the relations between semantic information, hence all query or reasoning services have to be based on ontologies. Several ontology languages have been proposed and investigated recently, such as RDFS [18], DAML+OIL [4], or OWL [17].

There are several problems that make it hard to query Semantic Web resources:

- *User Perspective:* Users who are interested in finding resources formulate queries related to some ontology. The user's (or their agent's) view of a domain is often different from an ontology developer's view. On the other hand, users are expected be be familiar enough with the domain ontology to efficiently formulate queries. Users would not have to learn the domain ontology if they could formulate their queries in an ontology that corresponds to their domain-specific view.

- *Application Integration:* In the current state of the Semantic Web, ontologies are developed from scratch, which means that many ontologies exist that describe the same domain. It is widely believed, that in the future development of the Semantic Web, this heterogeneity of ontologies will remain and multiple ontologies for many particular domains will coexist. Applications face the problem of obtaining information from sources which are described by different, independently developed ontologies.
- *Performance Overhead:* In many Semantic Web domains there are large or very large data sets. Queries can produce a considerable performance overhead. Problem-specific views of the resources could potentially minimize this problem.
- *Lack of Formal Definitions:* Various standardized vocabularies for metadata exist, such as IEEE LOM [11] for learning resources, but these are often only informally defined and do not allow for deeper reasoning.

In this paper, we propose an approach to overcome the problem of heterogeneous ontologies. Rather than defining a single large ontology, we propose to distinguish one or more "source ontologies" and a number of "target ontologies". The source ontologies describe the resource instances and are usually rather complex. The target ontologies individually cover certain aspects of the application domain. A user who wants to query heterogeneous data sources can formulate queries in terms of (multiple) relatively simple target ontologies. In other words, a view of the resources described in the source ontologies is created. In this view the resources are expressed using the target ontologies. For expressing ontologies, resources, and views, we use the Semantic Web query and transformation language TRIPLE [19].

We present our approach in Section 2 using an introductory example from the e-learning domain. In Section 3 we outline our solution. In Section 4 we give a short overview of the language TRIPLE. Section 5 describes our approach using the previously introduced scenario in TRIPLE. In Section 6 we give a description logic specification for our mappings. Section 7 presents related work, and Section 8 concludes the paper with our plans for future research.

2 Example: E-learning Scenario

In this section we describe a typical e-learning scenario that we use for the examples in this paper. The Semantic Web is used in our examples to support a search for relevant resources. These resources include online courses, online books, different kinds of exercises, etc.

The Learn@WU system[1] is an e-learning system that supports these kinds of resources and presents them on the Web. A subsystem of Learn@WU allows students to interactively try out exercises in tests which are randomly chosen from the learning resources.

[1] see `http://learn.wu-wien.ac.at/`

Students are not aware of the learning resource ontology and are not interested in how e-learning experts organize the learning resources; instead, they have their own view of exercises. The interactive test system only requires this limited view of exercises, not the whole learning resource ontology.

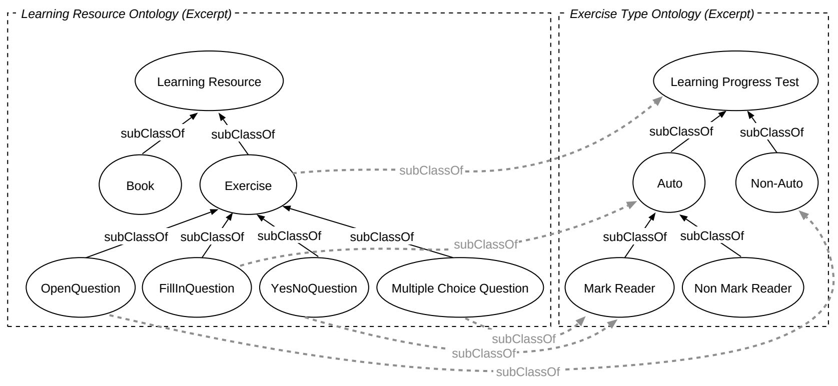

Fig. 1. The learning resource ontology, the exercise ontology and possible mappings

An excerpt of the learning resource ontology is depicted on the left hand side of Figure 1. The ontology of exercise types, as depicted on the right hand side of the figure, is a typical smaller ontology that only consists of a part of the learning resources in a particular context. There is a set of correspondences between these two ontologies that have to be considered during a mapping, as depicted by dotted lines in the figure.

3 Creating Views in a Target Ontology

Our approach relies on the distinction between two kinds of ontologies, *source ontologies* and *target ontologies*. Source ontologies are developed by domain experts for representing the Semantic Web resources in a particular domain. In contrast, target ontologies reflect a user's perspective or a domain-specific or problem-specific view of Semantic Web resources. Note that this distinction between source and taget ontologies is only conceptual; of course, any ontology can be a source ontology and target ontology as well.

Throughout this paper we use the following terms:

Source ontology: A (pre-existing) ontology that was developed specifically for a certain application.

Target ontology: This is an ontology that covers a small aspect of an application domain. Examples of a target ontology from the e-learning domain are various kinds of online exercises that can be automatically corrected or are suited for a mark reader. Another example is an ontology that contains the formats suitable for various delivery machines.

Metadata repository: Resource metadata, defined in terms of some (source) ontologies. There might be multiple repositories using the same source ontology.

Mapping model and view model: The specification to map the resources in the source ontologies to one or more target ontologies. The mapping model is specific to its source ontologies and target ontologies. Different view models can be defined on top of a mapping.

View: A set of resources, expressed in the target ontology. The view is created according to a mapping model and a view model.

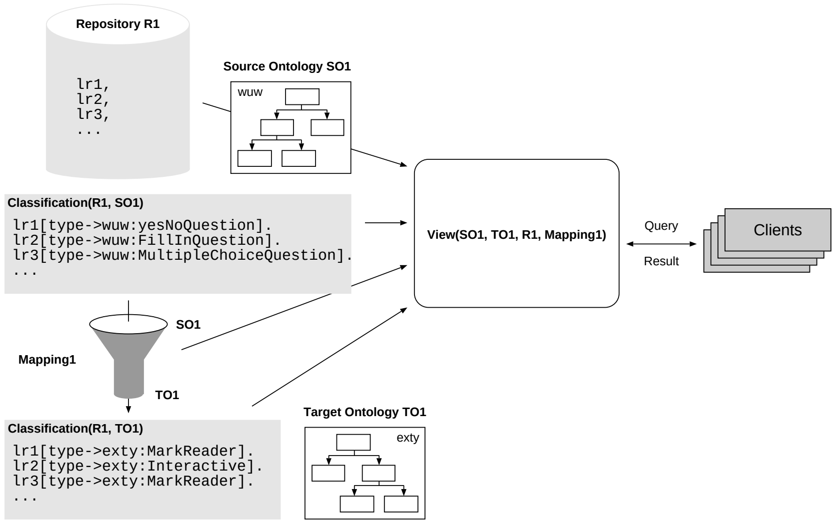

Fig. 2. Mapping Ontologies into Views

The ontology editors provide a set of *mapping rules* between the resources and their properties in the source ontologies and target ontologies. The *view rules* are applied using the ontologies involved, mapping rules, and instance resources. The result is a context-specific view, expressed in the application ontologies. Clients can query the view using the application ontology only, without further knowledge of the more complex source ontologies (see Figure 2).

More than two ontologies can also be involved in a view. We depicted this situation in Figure 3. In the examples of Section 5 we use only two ontologies for reasons of simplicity and brevity. The examples use the language TRIPLE, which is briefly explained in the next section.

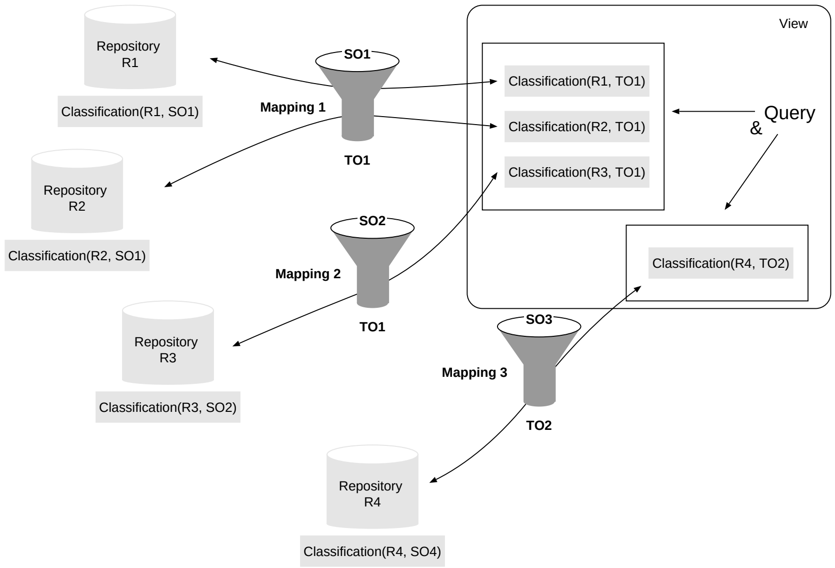

Fig. 3. View with Multiple Ontologies

4 Overview of TRIPLE

TRIPLE [19] is a rule language for the Semantic Web which is based on Horn logic and borrows many basic features from F-Logic [13] but is especially designed for querying and transforming RDF models.

TRIPLE can be viewed as a successor of SiLRI (Simple Logic-based RDF Interpreter [5]). One of the most important differences to F-Logic and SiLRI is that TRIPLE does not have fixed semantics for object-oriented features like classes and inheritance. Its modular architecture allows such features to be defined for object-oriented and other data models like UML, Topic Maps, or RDF Schema. Description logic extensions of RDF (Schema) like OIL, DAML+OIL, and OWL that cannot be fully handled by Horn logic are provided as modules that interact with a description logic classifier, e.g., FaCT [10], resulting in a hybrid rule language.

Namespaces and Resources: TRIPLE has special support for namespaces and resource identifiers. Namespaces are declared via clause-like constructs of the form *nsabbrev* := *namespace*., e.g.:

```
rdf := "http://www.w3.org/1999/02/22-rdf-syntax-ns#".
```

Resources are written as *nsabbrev:name*, where *nsabbrev* is a namespace abbreviation and *name* is the local name of the resource.

Statements and Molecules: Inspired by F-Logic object syntax, an RDF statement (triple) is written as: *subject*[*predicate* → *object*]. Several statements with the same subject can be abbreviated as “molecules”:

```
stefan[hasAge->33;isMarried->yes;...].
```

RDF statements (and molecules) can be nested, e.g.:

```
stefan[marriedTo->birgit[hasAge->32]].
```

Models: RDF models, i.e., sets of statements, are made explicit in TRIPLE ("first class citizens").[2] Statements, molecules, and also Horn atoms that are true in a specific model are written as *atom*@*model* (similar to Flora-2 module syntax), where *atom* is a statement, molecule, or Horn atom and *model* is a model specification (i.e., a resource denoting a model), e.g.:

```
michael[hasAge->35]@factsAboutDFKI.
```

TRIPLE also allows Skolem functions as model specifications. Skolem functions can be used to transform one model (or several models) into a new one when used in rules (e.g., for ontology mapping/integration):

```
O[P->Q]@sf(m1,X,Y) <- ...
```

If all (or many) statements/molecules or Horn atoms in a formula are from one model, the following abbreviation can be used: *formula*@*model*. All statements/molecules and Horn atoms in *formula* without an explicit model specification are implicitly suffixed with @*model*.

Logical Formulae: TRIPLE uses the usual set of connectives and quantifiers for building formulae from statements/molecules and Horn atoms, i.e., $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$, etc.[3] All variables must be introduced via quantifiers, therefore marking them is not necessary (i.e., TRIPLE does not require variables to start with an uppercase letter as in Prolog).

Clauses and Blocks: A TRIPLE clause is either a fact or a rule. Rule heads may only contain conjunctions of molecules and Horn atoms and must not contain (explicitly or implicitly) any disjunctive or negated expressions. To assert that a set of clauses is true in a specific model, a model block is used: @*model* $\{$*clauses*$\}$, or, if the model specification is parameterized: $\forall$ *Mdl* @*model*(*Mdl*) $\{$*clauses*$\}$.

5 Examples of Querying Using TRIPLE Views

In this section we provide examples for resolving typical problem scenarios in TRIPLE using the view concepts presented in Section 3.

[2] Note that the notion of *model* in RDF does not coincide with its use in (mathematical) logics.

[3] For TRIPLE programs in plain ASCII syntax, the symbols AND, OR, NOT, FORALL, EXISTS, `<-`, `->`, etc. are used.

5.1 Simple Views

At first, we have to define the ontologies under consideration. The excerpt from the Learn@WU ontology for learning resources introduced in Section 2 can be defined by the following TRIPLE statements:

```
@wuw:ont {
   wuw:LearningResource[rdfs:subClassOf -> rdfs:Resource].
   wuw:Book[rdfs:subClassOf -> wuw:LearningResource].
   wuw:Exercise[rdfs:subClassOf -> wuw:LearningResource].
   wuw:OpenQuestion[rdfs:subClassOf -> wuw:Exercise].
   wuw:FillInQuestion[rdfs:subClassOf -> wuw:Exercise].
   wuw:YesNoQuestion[rdfs:subClassOf -> wuw:Exercise].
   wuw:MultipleChoiceQuestion[rdfs:subClassOf -> wuw:Exercise].
   ...
}
```

Here we define a class hierarchy of learning resources which is expressed in a TRIPLE model `wuw:ont`. The RDF Schema [18] semantics (for `rdfs:subClassOf`, `rdf:type` etc.) are defined as shown in the appendix (model `rdfschema(Mdl)`).

Next, we define a sample metadata repository using this ontology:

```
question1_1[rdf:type -> wuw:OpenQuestion; wuw:difficulty -> low].
question1_2[rdf:type -> wuw:FillInQuestion;
            wuw:difficulty -> medium].
question1_3[rdf:type -> wuw:YesNoQuestion; wuw:difficulty -> high].
question2_1[rdf:type -> wuw:YesNoQuestion; wuw:difficulty -> low].
question2_2[rdf:type -> wuw:MultipleChoiceQuestion;
            wuw:difficulty -> medium].
question3_1[rdf:type -> wuw:MultipleChoiceQuestion;
            wuw:difficulty -> high].
book1[rdf:type -> wuw:Book].
...
```

Now consider a second ontology which defines exercise types and which is used by the web test subsystem of Lean@WU. This ontology is much simpler than the learning resource ontology and uses fewer resources and properties. For instance, all `difficulty` properties and all resources that are not exercises (like the book resource in the above example) can be omitted. The ontology is expressed as a second TRIPLE model:

```
@exty:ont {
    exty:LearningProgressTest[rdfs:subClassOf -> rdfs:Resource].
    exty:Auto[rdfs:subClassOf -> exty:LearningProgressTest].
    exty:NonAuto[rdfs:subClassOf -> exty:LearningProgressTest].
    exty:MarkReader[rdfs:subClassOf -> exty:Auto].
    exty:NonMarkReader[rdfs:subClassOf -> exty:Auto].
}
```

The method of integrating these two ontologies is to use the `wuw` ontology as source ontology and the `exty` ontology as target ontology. Clients only use views expressed in the target ontology for the queries. We first define a mapping between the two ontologies and then define view models based on the mapping. A very simple mapping uses only the `subClassOf` relationship to model the relationships between the two ontologies. The mapping is placed in a separate mapping model:

```
@exty:mappings {
   wuw:MultipleChoiceQuestion[rdfs:subClassOf -> exty:MarkReader].
   wuw:YesNoQuestion[rdfs:subClassOf -> exty:MarkReader].
   wuw:FillInQuestion[rdfs:subClassOf -> exty:Auto].
   wuw:Exercise[rdfs:subClassOf -> exty:LearningProgressTest].
   wuw:OpenQuestion[rdfs:subClassOf -> exty:NonAuto].
}
```

For any given set of models, we can now—in a next step—define the rules for creating a view model `view(Ont1, Ont2, Mappings)`:

```
FORALL Ont1, Ont2, Mappings  @view(Ont1, Ont2, Mappings) {
  FORALL R,P,O  R[P -> O] <- R[P -> O]@Ont2.
  FORALL R,P,O  R[P -> O] <-
      R[P -> O]@rdfschema(view(Ont1, Ont2, Mappings)).
  FORALL R,C1,C2  R[rdf:type -> C1]
      R[rdf:type -> C2]@rdfschema(Ont1) AND
       C2[rdfs:subClassOf -> C1]@Mappings.
}
```

The view model is parameterized by taking two ontologies and a mapping between the ontologies as parameters. In this example `Ont1` is the source and `Ont2` the target ontology. The first rules states that everything in `Ont2` holds. Then, in the second rule, the RDF schema semantics defined before are applied to the view model. Finally we use the mapping model with its `subClassOf` relationships to create a view of the two ontologies according to the mapping.

Note that the definitions above are typically implemented by different roles, such as providers of ontologies and providers of view models. The actual resources are usually produced by the users of the system. The users (or the user tools) are usually interested in creating queries as well, but here it is important to be able to generate queries by very simple means. For instance, a user might compose queries in GUI-based tools. The view concept helps us to limit the query syntax and yet produce usable and user-customizable results.

For instance, a typical query using the above ontology view definition might be that the web test system requires those resources that are of type `MarkReader` (i.e., exercises that it can automatically test in web forms). The system simply has to query the view for resources of the particular type and receives all resources in the view that are exercises of type `MarkReader`. Note that the simplicity of the query is due to the view definition described above.

```
FORALL R  <- (R[rdf:type -> exty:MarkReader])@view(wuw:ont,
  exty:ont, exty:mappings).
```

The output of the TRIPLE engine for the example above is:

```
R = question3_1
R = question2_2
R = question2_1
R = question1_3
```

5.2 Constraining the Views with Property Values

As already mentioned in Section 2, the simple mapping rules used in the previous section might not be enough. Sensible views often have to be created under consideration of the (property) values of the data. Consider a simple example as an extension of the scenario in the previous section: the web test system should create a shuffled mix from the set of exercises that are the basis for test questions. Further assume that the test questions are defined as being either of type `FillInQuestion`, `YesNoQuestion`, or `MultipleChoiceQuestion`, and are not of the difficulty `low`. Moreover, the web test system should only use those questions from the result set that it can automatically correct in web forms (i.e., those of type `MarkReader`).

We have to use a more complex mapping rule to implement the definition of test questions:

```
@exty:mappings {
  ...
  FORALL R  R[rdf:type -> exty:TestQuestion] <-
     ((R[rdf:type -> wuw:MultipleChoiceQuestion] OR
       R[rdf:type -> wuw:YesNoQuestion] OR
       R[rdf:type -> wuw:FillInQuestion])  AND
      (R[wuw:difficulty -> medium] OR
       R[wuw:difficulty -> high]))@rdfschema(wuw:ont).
}
```

Note that the `exty:TestQuestion` definition is simply composed of `AND` and `OR` expressions, which means it can easily be created, say, by a tool that allows users to compose ontologies graphically.

The query for the view created with this mapping again is quite simple. It simply adds the new test question type definition to the previous `MarkReader` query by composition with `AND`:

```
FORALL R  <- (R[rdf:type -> exty:TestQuestion]
              AND R[rdf:type -> exty:MarkReader])
             @view(wuw:ont, exty:ont, exty:mappings).
```

The output of the TRIPLE engine for this query is:

```
R = question2_2
R = question3_1
R = question1_3
```

6 Description Logic Specification of Mappings

An interesting class of mappings is those that can be specified with a standard description logic like $\mathcal{SHIQ}$ [9] or its Semantic Web variants (OIL, DAML+OIL, OWL).

The advantage of this approach is that users can create relatively interesting mappings with a simple point-and-click interface, since description logic expressions do not use any variables and therefore only class and property (role) names plus a small set of connectives (conjunction, disjunction, negation, etc.) have to be dealt with.

The mappings are specified by connecting class expressions of the source (Ont_S) and target (Ont_T) ontology with the usual implication:

$$C^{Ont_S} \sqsubseteq C^{Ont_T}$$

The most simple case is where C^{Ont_S} and C^{Ont_T} are class names ("primitive concepts"), e.g.:

$$\text{wuw:MultipleChoiceQuestion} \sqsubseteq \text{exty:MarkReader}$$

The corresponding RDFS/OWL expression (in TRIPLE syntax) uses the `rdfs:subClassOf` to relate the two classes:

```
wuw:MultipleChoiceQuestion[rdfs:subClassOf -> exty:MarkReader].
```

The view definition `@view(Ont1, Ont2, Mappings)` contains the following rule to map instances from C^{Ont_S} to C^{Ont_T}:

```
FORALL R,C1,C2  R[rdf:type -> C2] <-
      R[rdf:type -> C1]@rdfschema(Ont1) AND
      C1[rdfs:subClassOf -> C2]@Mappings.
```

For the case of complex class expressions, it is much simpler to create TRIPLE rules (via an automatic mapping) instead of writing rules that handle these class expressions directly.[4] For expressions of the form $C^{Ont_S} \sqsubseteq C^{Ont_T}$, C^{Ont_S} becomes the body and C^{Ont_T} becomes the head of a TRIPLE rule:

$$\text{TRIPLE}(C^{Ont_T}) \leftarrow \text{TRIPLE}(C^{Ont_S}).$$

As a consequence, C^{Ont_T} can only be a very simple class expression (i.e., no disjunction or negation is allowed).

[4] An alternative would be to access a description logic classifier from within the TRIPLE engine.

As the following example shows, the mapping from the description logic expression to the corresponding TRIPLE rule is straightforward:

$$\left.\begin{array}{l}(\text{wuw:MultipleChoiceQuestion} \\ \quad\sqcup \text{ wuw:YesNoQuestion} \\ \quad\sqcup \text{ wuw:FillInQuestion}) \\ \sqcap\ \exists\text{wuw:difficulty}.\{\text{medium}, \text{high}\}\end{array}\right. \sqsubseteq \text{exty:TestQuestion}$$

```
FORALL R  R[rdf:type -> exty:TestQuestion] <-
     ((R[rdf:type -> wuw:MultipleChoiceQuestion] OR
       R[rdf:type -> wuw:YesNoQuestion] OR
       R[rdf:type -> wuw:FillInQuestion])  AND
      (R[wuw:difficulty -> medium] OR
       R[wuw:difficulty -> high]))@rdfschema(wuw:ont).
```

7 Related Work

Our work follows the mediator architecture suggested by Wiederhold [21]. He addresses the problem in the database context that for end users the lack of abstraction and the need to understand the representation of data hinders the access to the available data. Another motivation for mediator architectures is the problem of combining information from multiple databases or other information sources where the representation and structure is different. In mediator architecture, a mediator plays the role of a virtual database: Users might formulate queries as if the data were available in a mediator database and the mediator translates the queries to the data sources. The mediator then synthesizes the answers from the sources and returns the answer to the user. In our approach it is possible for users to formulate their queries only in terms of the target ontology.

Several approaches have been proposed and implemented to build mediators over relational databases, for example [3].

Calvanese et al. [1] analyze the problems of ontology integration. They describe the ontologies with Description Logics. They argue that for capturing the mappings between ontologies the use of Description Logic is not sufficient and suggest a notion based on queries which is similar to our approach using rule-based views.

The subset of possible mappings we can define in description logics, as described in section 6, is still relevant in many practical situations, especially since they are easier to specify via a graphical user interface.

Mitra et al. [15] investigate the integration of heterogeneous sources (UML, DAML+OIL). They define articulation rules that establish correspondence between concepts in different ontologies. These articulation rules are used to rewrite queries. They use a common ontology format called the ONION conceptual model.

The currently available ontology editors for the Semantic Web also support some form of ontology mapping. Noy et al. [16] surveys ontology mapping support of these tools.

Recently several researchers investigated different aspects of ontology mappings for the Semantic Web. The mappings in these investigations are used not to query for resources using views but for information integration, where essentially the same problems have to be solved.

Halevy et al. [8] developed a data management infrastructure for Semantic Web applications. The main focus of their work was to enable interoperability of XML sources. They developed a mapping language based on XQuery. The mappings are also used to support query answering, which is possible in both directions using their query-answering algorithm. We use a specific class of Horn rules for expressing the mapping rules.

On the Semantic Web the data from heterogeneous sources is described by different ontologies. Doan et al. [6] argue that, in the case of information processing, the mapping between ontologies should happen (semi-)automatically. They developed a system that employs machine learning techniques to find the mappings.

Catarci and Lenzerini [2] described interschema knowledge using description logic. Their work focuses on how to use this interschema information for reasoning tasks, for example, to check the consistency of integrated information systems.

Data integration problems and possible solutions are extensively analyzed in database literature. Wache [20] provides a classification of the problems. The schematic and data heterogeneity conflicts are described in [14]. Semantic conflicts are analyzed by Kashyap et al. [12] and Goh [7]. Structural or semantic heterogeneity or data inconsistency can cause integration conflicts:

- Structural conflicts occur if we compare the structure of the data models of different information sources. In the case of structural conflicts, the semantic structure of the data is the same, the difference is in the representation structure. Structural conflicts are, for example, label conflicts, where the data attributes are equivalent but different labels are used.
- In the case of a semantic conflict, the equivalent structural elements have to be interpreted with different semantics. A typical semantic conflict is the unit and scale conflict, when two numeric values have the same semantics, but the values have a different meaning. An example would be "price", if it is represented in one information system implicitly in Euros, while in other systems in Dollars. Another typical semantic conflict is when two information sources use different data representation types.

Because of the large number of possible conflicts, individual treatment of the problems is often necessary. We have only concentrated on a subset of these problems that can be resolved with mapping rules between resources and their properties in two (or more) ontologies.

8 Conclusion and Future Work

On the Semantic Web, resources are described with metadata related to some ontology. Using our approach, users or applications can express their queries for resources with respect to an ontology that reflects a user-specific or application-specific view of the domain (i.e., in the application ontologies). We demonstrated with some typical application cases from the e-learning domain how mapping rules and views can be defined using the language TRIPLE. These views allow clients to formulate queries only in the target ontologies. The mapping rules connect the instances described by the "source ontology" with the terminologies (classes and properties) of the "target ontology". The ontologies, the mappings, and the queries are specified in TRIPLE. Simple mappings can also be formulated in Description Logic; we demonstrated a specific class of mappings which is relevant in many practical situations.

Our approach has several advantages:

- Users can formulate their queries in the target ontology in a more natural way than in the source ontology. The mappings in our method are developed by ontology developers and usually not by the users who submit queries. Our objective was not to require the user to be familiar with the domain expert's ontology; mapping rules can be created in any case only by persons who are familiar with the source ontology.
- If the source ontology evolves, the application ontology often remains the same, and only the mappings have to be updated or rewritten. In this way the users are not affected by changes in the source ontology.
- As in other mediator approaches, our approach can be used to integrate data described in different, independently developed ontologies.
- Integration with other software systems is possible.
- Queries based on views can potentially be more efficient than queries based on the source ontologies.
- Because our implementation uses TRIPLE, the ontologies we connect with views can be represented in different ontology languages like RDFS or DAML +OIL.
- The method could also be used for integrating information sources. Our paper demonstrates the opposite direction: In the examples we have only one information source and several (possible) views. When integrating information sources the automatic creation of mappings is much more important, in our examples the mappings are created manually.

Of course, our approach has some limitations:

- Our approach assumes that some mapping between different ontologies can be found. If it is not (easily) possible, other data integration approaches (e.g., imperative approaches) may be more appropriate. Specifically if programmatic specification of data integration is constantly required (e.g., by the end user), rule-based approaches might be cumbersome.
- The possibly increased efficiency using a view has to be contrasted with the effort of creating the view. Specifically, when the resources in source ontology often change, creating views can cause problems. This problem can be resolved by introducing caching structures in the view computation.

In our future research we plan to address more mapping conflicts, e.g., semantic conflicts. We also want to investigate how to integrate our approach with other data integration approaches and how to optimize query execution.

Acknowledgement

This work was supported by the Elena project and is partly sponsored by the European Commission (IST-2001-37264).

References

1. Calvanese, D., De Giacomo, G., Lenzerini, M.: Ontology of Integration and Integration of Ontologies. In *Proceedings of the 2001 Description Logic Workshop (DL 2001)*.
2. Catarci, T., Lenzerini, M.: Representing and Using Interschema Knowledge in Cooperative Information Systems. *Journal of Intelligent and Cooperative Systems*, 2(4):375-398, 1993.
3. Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J.: The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *Proceedings of IPSJ Conference*, pp. 7-18, Tokyo, Japan, October 1994.
4. DAML+OIL. http://www.w3.org/TR/daml+oil-reference.
5. Decker, S., Brickley, D., Saarela J., Angele, J.: A query and inference service for RDF. In *The Query Languages Workshop, QL'98*, WorldWideWeb Consortium (W3C), Boston, USA, 1998., http://www.w3.org/TandS/QL/QL98/.
6. Doan, A., Madhavan, J., Domingos P., Halevy, A.: Learning to Map between Ontologies on the Semantic Web. In *11th International World Wide Web Conference (WWW'2002),* Hawaii, USA, 2002.
7. Goh, C.: Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems. Ph.D. Thesis, MIT Sloan School of Management, 1996.
8. Halevy, A., Ives, Z., Tatarinov, I., Mork, P.: Piazza: Data Management Infrastructure for Semantic-Web Applications. In *Proceedings of the Twelfth International World Wide Web Conference*, Budapest, Hungary, May 2003.
9. Horrocks, I., Sattler, U., Tobies, S.: Practical Reasoning for Expressive Description Logics. In *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, 1999.
10. Horrocks, I.: The FaCT System. 2001., http://www.cs.man.ac.uk/∼horrocks/FaCT/
11. IEEE Learning Technology Standards Committee: *IEEE LOM Draft 6.1*, http://ltsc.ieee.org/doc/index.html, 2001.
12. Kashyap, V., Sheth, A. P.: Semantic and Schematic Similarities Between Database Objects: A Context-Based Approach. In *VLDB Journal* 5(4): 276-304(1996).
13. Kifer, M., Lausen, G., Wu, J.: Logical Foundations of Object-Oriented and Frame-Based Languages, In *Journal of the ACM*, Jul. 1995., vol 42, pp. 741–843.
14. Kim, W., Seo, J.: Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Computer*. Vol 24. No. 12. December, 1991.
15. Mitra P., Wiederhold G., Decker S.: A scalable framework for the interoperation of information sources. In *Semantic Web Working Symposium*, pp. 317–329, 2001.

16. Noy, N. F., Musen, M. A.: Evaluating Ontology-Mapping Tools: Requirements and Experience. In *Workshop on Evaluation of Ontology Tools at EKAW'02 (EON2002)*. 2002. http://www.smi.stanford.edu/pubs/SMI_Reports/SMI-2002-0936.pdf.
17. OWL. http://www.w3.org/TR/owl-ref/.
18. RDF Schema. http://www.wcatarci93representing3.org/TR/rdf-schema/.
19. Sintek, M., Decker, S.: TRIPLE–A Query, Inference, and Transformation Language for the Semantic Web. In *Proceedings of the First International Semantic Web Conference (ISWC)*, Sardinia, June 2002.
20. Wache, H.: Semantische Mediation fur heterogene Informationsquellen. Akademische Verlagsgesellschaft AKA GmbH, Berlin, Reihe "Dissertationen zur Künstlichen Intelligenz (DISKI)", Nr. 261, 2003. In German.
21. Wiederhold, G.: Mediators in the Architecture of Future Information Systems. In *IEEE Computer*, Vol. 25, No. 3. March, 1992.

Appendix: Complete example

```
// 0. rdf schema semantics

FORALL Mdl @rdfschema(Mdl) {
  FORALL O,P,V   O[P->V] <- O[P->V]@Mdl.
  FORALL O,P,V   O[P->V] <- EXISTS S
      (S[rdfs:subPropertyOf->P] AND O[S->V]).
  FORALL O,P,V   O[rdfs:subClassOf->V] <-
      EXISTS W   (O[rdfs:subClassOf->W] AND W[rdfs:subClassOf->V]).
  FORALL O,P,V   O[rdfs:subPropertyOf->V] <-
      EXISTS W   (O[rdfs:subPropertyOf->W] AND W[rdfs:subPropertyOf->V]).
  FORALL O,T   O[rdf:type->T] <-
      EXISTS S   (S[rdfs:subClassOf->T] AND O[rdf:type->S]).
}

// 1. learning object ontology plus some resources at WUW

@wuw:ont {

   // ontology
   wuw:LearningResource[rdfs:subClassOf -> rdfs:Resource].
   wuw:Book[rdfs:subClassOf -> wuw:LearningResource].
   wuw:Exercise[rdfs:subClassOf -> wuw:LearningResource].
   wuw:OpenQuestion[rdfs:subClassOf -> wuw:Exercise].
   wuw:FillInQuestion[rdfs:subClassOf -> wuw:Exercise].
   wuw:YesNoQuestion[rdfs:subClassOf -> wuw:Exercise].
   wuw:MultipleChoiceQuestion[rdfs:subClassOf -> wuw:Exercise].

   // some instances
   question1_1[rdf:type -> wuw:OpenQuestion; wuw:difficulty -> low].
   question1_2[rdf:type -> wuw:FillInQuestion; wuw:difficulty -> medium].
   question1_3[rdf:type -> wuw:YesNoQuestion; wuw:difficulty -> high].
   question2_1[rdf:type -> wuw:YesNoQuestion; wuw:difficulty -> low].
   question2_2[rdf:type -> wuw:MultipleChoiceQuestion; wuw:difficulty -> medium].
   question3_1[rdf:type -> wuw:MultipleChoiceQuestion; wuw:difficulty -> high].
   book1[rdf:type -> wuw:Book].
}

// 2. exercise types ontology

@exty:ont {
    exty:LearningProgressTest[rdfs:subClassOf -> rdfs:Resource].
    exty:Auto[rdfs:subClassOf -> exty:LearningProgressTest].
    exty:NonAuto[rdfs:subClassOf -> exty:LearningProgressTest].
    exty:MarkReader[rdfs:subClassOf -> exty:Auto].
```

```
    exty:NonMarkReader[rdfs:subClassOf -> exty:Auto].
}

// 3. mapping definitions

@exty:mappings {

   wuw:MultipleChoiceQuestion[rdfs:subClassOf -> exty:MarkReader].
   wuw:YesNoQuestion[rdfs:subClassOf -> exty:MarkReader].
   wuw:FillInQuestion[rdfs:subClassOf -> exty:Auto].
   wuw:Exercise[rdfs:subClassOf -> exty:LearningProgressTest].
   wuw:OpenQuestion[rdfs:subClassOf -> exty:NonAuto].

   FORALL R  R[rdf:type -> exty:TestQuestion] <-
       ((R[rdf:type -> wuw:MultipleChoiceQuestion] OR
         R[rdf:type -> wuw:YesNoQuestion] OR
         R[rdf:type -> wuw:FillInQuestion])  AND
        (R[wuw:difficulty -> medium] OR
         R[wuw:difficulty -> high]))@rdfschema(wuw:ont).

}

// 4. view definition

FORALL Ont1, Ont2, Mappings  @view(Ont1, Ont2, Mappings) {

  FORALL R,P,O  R[P -> O] <- // everything in Ont2 holds
      R[P -> O]@Ont2.

  FORALL R,P,O  R[P -> O] <- // everything in mappings holds
      R[P -> O]@Mappings.

  FORALL R,P,O  R[P -> O] <- // apply rdf schema semantics on "self"
      R[P -> O]@rdfschema(view(Ont1, Ont2, Mappings)).

  FORALL R,C1,C2  R[rdf:type -> C1] <- // rdfs:subClassOf mapping
      R[rdf:type -> C2]@rdfschema(Ont1) AND
       C2[rdfs:subClassOf -> C1]@Mappings.
}

// 5. queries

// give me the types of all resources from the WUW ontology
// wrt. view ontology (i.e., in terms of exty ontology)
FORALL R,O  <- R[rdf:type -> O]@view(wuw:ont, exty:ont, exty:mappings).

// query Mark Readers for online system
FORALL R  <- (R[rdf:type -> exty:MarkReader])@view(wuw:ont, exty:ont, exty:mappings).
```