MDE for publishing Data on the Semantic Web

Guillaume Hillairet, Frédéric Bertrand, Jean Yves Lafaye

{guillaume.hillairet01, fbertran, jylafaye}@univ-lr.fr
Laboratoire Informatique Image Interaction,
University of La Rochelle, FRANCE

Abstract. Publishing local data on the Semantic Web entails providing a shareable semantic data representation. We present a complete MDE approach that allows importing data sources into an RDF repository. At a preliminary stage, the object domain model is mapped to an ontology and to a persistence model. Specifying mappings requires a model annotation performed by a domain expert. All other processes are automated and data transformations are generated from the mappings via model weaving techniques.

Keywords: MDE, Semantic Web, Ontology, RDF

1 Introduction

Due to the expansion of the Semantic Web [2], three technical spaces presently rise up in the information system design landscape. The object space, which evolves into the Model Driven Engineering (MDE) space, allows implementing software applications and uses several languages and libraries. The second space concerns data persistence, mainly through relational databases management systems. The third space deals with resource description and is the core of the Semantic Web. It allows publishing RDF data on the Web as well as OWL ontologies, and ensures that machines can interpret and combine data.
Publishing existing data on the Semantic Web supposes defining bridges between technical spaces. Data persistency is mainly achieved through relational databases systems. In order to ease the use of relational data in enterprise application, several approaches, mainly based on the Active Record pattern [9], propose an object relational mapping (ORM) solution [1] [4]. This is an example of such a bridge, tending to cope with the so-called 'impedance mismatch' between distinct formalisms. More recently, some academic and industrial tools started offering quite homologous solutions for bridging relational and RDF data representations [3] [5] [10] [14].
Our opinion is that the gap between relational and RDF data models, is too wide for being crossed over in a single step, i.e. with a direct mapping. In fact, we advocate for introducing an object-oriented domain model as an intermediate between the relational and RDF layers. Despite the differences existing between the object and RDF model [16], their similarities (notion of classes and class hierarchy) remain higher than those existing between the relational and the RDF model. We assume that

the first step which maps the relational and object models is already conveniently addressed by the literature (as evoked above). So, we can focus on the second step that aims at filling the gap between object and RDF data models. The core of our proposal consists of specifying the mapping between the object-oriented domain model and one or more ontologies. The general use case for our work is the following: within the context of Software and Information System design, we assume that an object-oriented domain model does exist, which stands as a reference model for both software applications and database management systems. In order to raise interoperability and data sharing, we propose an object-ontology mapping tool that allows two kinds of data access. The first one follows the ETL [13] (extract, transform and load) pattern for publishing objects (possibly loaded from relational database) as RDF triples. The second data access offer by our tool is an on-demand mapping that translates SPARQL [19] queries into HQL [1] (Hibernate Query Language) that can be executed over the object-oriented domain model.
The remainder of the paper is organized as follows. Section 2 presents our motivation for developing this work and provides a small example that will be kept as an illustration until the end of the paper. Section 3 describes the overall architecture of our proposal. Section 4 outlines the mapping language between the object and ontology modelling spaces and gives some examples. Section 5 presents the ETL approach for publishing objects and relational data as RDF resources. Section 6 presents the on-demand mapping implementation. Related works are quoted in Section 7 before concluding in Section 8.

2 Motivation

Developing the Semantic Web entails publishing existing relational database content in an OWL/RDF format. In this paper, we present a model driven approach for publishing data that have been created, or loaded, by object-oriented applications and that are finally stored in RDF repositories. We show how MDE allows a better coping with such an architectural complexity. Most of the computer applications presently rely on object-oriented modeling, while most of the data are stored in relational databases. We take account of this situation and propose to use the object-oriented domain model as the basis for defining a mapping between object-oriented data and RDF data sources.
Before going further, let’s clarify our insight on the usage of domain ontologies and object-oriented domain models. Despite one common objective which is to capture the main concepts of a domain, and in addition to technical and syntactic discrepancies, the viewpoints and final use are differing. The object-oriented domain model is a basis for designing both robust software application and persistent data layers. The matter is not so to give unambiguous definitions of terms (which are usually shared by domain users) as to list the prominent elements and specifies the constraints and mutual relationships [15]. Domain ontology, also capture main concepts and terminology but are fitted to reasoning, browsing and querying which are the facilities required in the semantic web context. Conversely, they generally are not adapted to software and database design. The solutions brought to ontology data

persistency are efficient for semantic queries but lack integrity constraint definition and checking; they are not convenient for update insert and overall delete operations.
Practically, the question of creating RDF data (individuals) conforming to a special ontology arises in two contexts

– In case an existing ontology is available that fits the data, direct data sharing is facilitated. Since existing ontologies essentially are consensual, then local data then prove to be presented through a shared knowledge view and vocabulary.
– In case no existing ontology is satisfactory, a special new ontology has to be created so as to account for the local data semantics. However, in order to ensure that this new ontology semantics is not ambiguous and can practically be shared, all genuine concepts should extend actual concepts of an existing ontology. This is easily achieved by using such ontology matching facilities as, for instance, those provided in OWL.

Since an ontology may capture some aspects of the data but fails to capture others, one domain model may need to be mapped to several ontologies. Whatever the choice, defining a correspondence between the ontologies and the data to be published is mandatory. We claim that the domain model is a key representation, standing as a pivot model halfway between the persistence and ontology models. Let's notice that the persistence layer may indifferently be implemented in any manner, say relational or XML, with no special impact upon the overall approach.

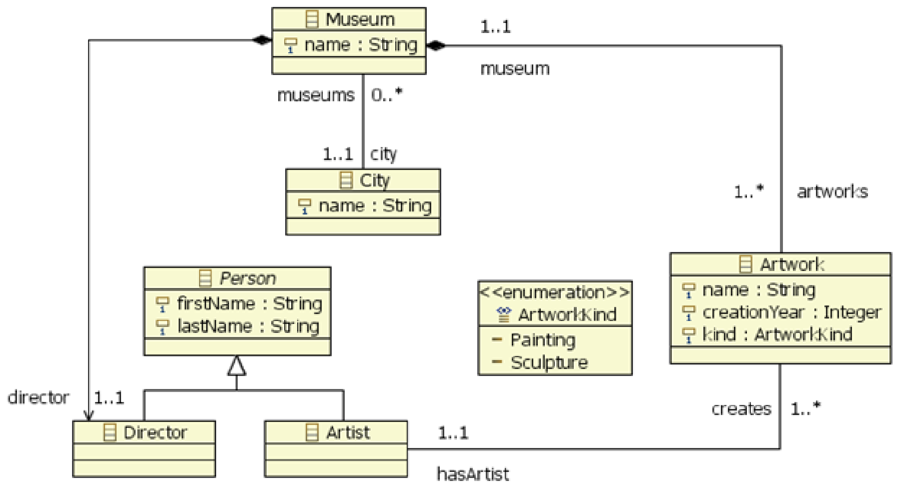

Fig. 1: An object-oriented domain model for a cultural application *(excerpt).*

A short example depicted in Fig. 1 shows the domain model of a museum while Fig. 2 gives an excerpt of the object-ontology mapping between the domain model and several ontologies. Let's now consider the museum example, and illustrate our overall approach. Using appropriate functions provided by modeling tools, the object-oriented domain model may lead to provide a relational schema which is made of tables that are associated to domain model classes. The resulting database is populated by domain data. In order to publish the data on the Semantic Web, an ontology is needed that will account for the concepts underlying the domain model. Fig. 2 shows what links can be specified when weaving the domain model and the ontology to be generated. In this example, the resulting ontology merges three existing ontologies *(e.g. foaf, dbpedia, geonames)* and a novel one *(museum)*.

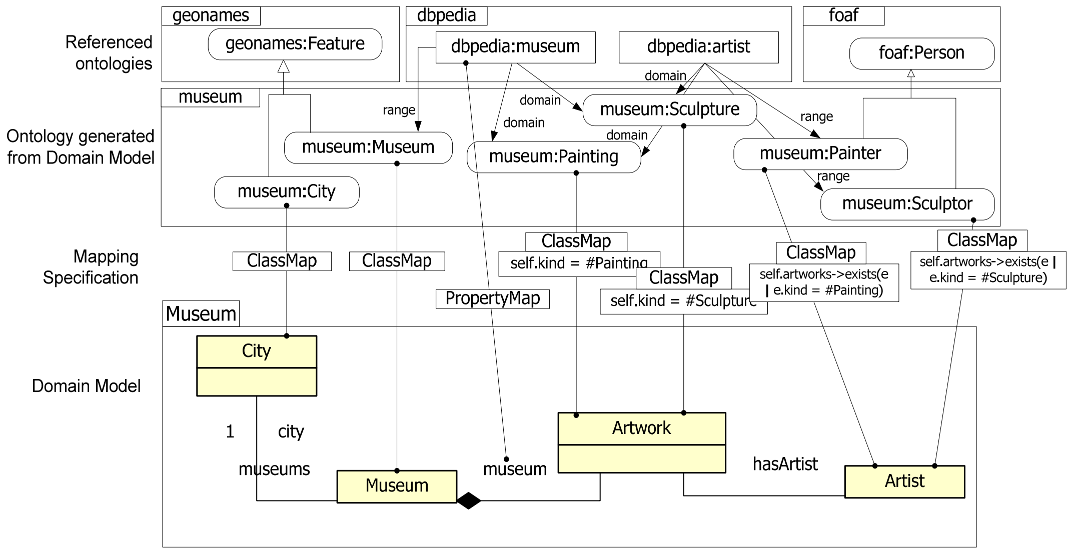

Fig. 2: Mapping the object-oriented domain model with ontologies.

3 Overall Approach

We already evoked the object-relational mapping that links the domain model to the corresponding database. We pointed out the so-called impedance mismatch. We take advantage of the important academic work on this subject and do not account for how the object-relational mapping is achieved. Conversely, we focus on the object-ontology mapping which is quite similar and has less been studied. Our approach can be split into three steps:

1. Specifying the correspondence between domain model elements and ontology concepts by means of the mapping language we developed. This step requires a human domain expert who operates in a specially designed software environment that aids and controls his actions. The role of the expert is mainly to select, constrain and combine elements in the domain model, using operators such as restriction, intersection, union, subsumption, equivalence… and map or create corresponding concepts in the ontology. The result is a mapping specification.
2. Actual generation – on the basis of the mapping specification - of the ontology that describes the data semantics.

3. Populating the knowledge base which is associated to the ontology (individuals) from the relational data. This step involves both the object-relational and object-ontology mappings.

According to our approach, the object-relational mapping also is represented by a weaving model. It links the domain model to the database. We choose to consider the domain model as a metamodel (EMF model) at the M2 level (MDE). Doing so, the domain data may be represented at the M1 level. Then, domain model instances appear as M1 model elements that comply to their M2 metamodel. In contrast, using plain UML class diagrams with instances and classes appearing at a same level would have led to an unnecessary complication of the weaving and transformation processes.
Our mapping language between an object model and several ontologies is defined by a textual grammar. It is then processed and translated into weaving models, themselves being specified with the AMW model weaver [6] (Atlas Model Weaver).
Importing and transforming the data from the database so as to populate the knowledge base is achieved via a series of model transformations, implemented in ATL [11] (Atlas Transformation Language), as shown in Fig. 3.

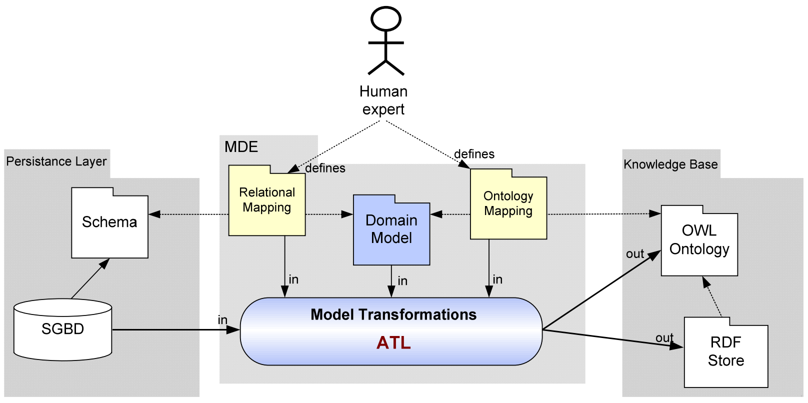

Fig. 3: Linking the modelling spaces to one another

4 Defining ontological views over domain models

This section presents the implementation and use of the mapping language we defined so as to specify the correspondence between the domain model and one (or more) ontology. For sake of conciseness, the complete language grammar cannot be presented here.

4.1 The Domain model / Ontology mapping Language by example

We use the object-oriented domain model in Fig. 1 as an example that outlines our mapping language. The ontology in Fig. 2 appears as a view being defined upon the

domain model thanks to the mapping language. The mapping language implementation we outline hereunder links an EMF metamodel to several OWL ontologies.
A mapping specifies correspondences between compatible elements, such as: a package and an ontology; an EMF class and an OWL class; an EMF attribute and an OWL datatypeProperty, *etc*. More complex mappings are allowed, in order to express 1-N or N-1 correspondences between compatible elements. In the following, we give examples of such simple and complex mappings.

4.1.1 Simple Mapping

A simple mapping accounts for a 1-1 correspondence between a domain model element and an ontology concept, as shown in the listing below. A mapping instance contains the keyword *map* followed by the element to be mapped (*package, class, attribute or reference*). The mapping body is similar to the object language structure, i.e. a class mapping should be nested in a package mapping, a property mapping within a class mapping etc…

```
prefix foaf: "http://xmlns.com/foaf/0.1/";
prefix geonames: "http://www.geonames.org/ontology#";

map package Museum with museum: "http://museum#" {
  map class Museum with museum:Museum {
        uriPattern = "http://museum/" + self.name;
        subClassOf = {geonames:Feature}
        properties = {
          map attr name with museum:name;
          map ref artworks with museum:artworks;
          map ref city with geonames:locatedIn;
  }
}
  map class Director with foaf:Person {
        properties = { map attr lastName with foaf:family_name; }
  }
}
```

Listing 1: An example of simple mappings

The package mapping specifies which the corresponding target ontology is. In the *museum* example, the target ontology is given the same name as the source object model (*museum*), and a special URI. Let's notice that the museum ontology does not yet exist. It is still an abstract ontology in wait of being generated at a further step, in accordance to the specified mapping. When the package is to be mapped to an existing ontology, the latter should be declared via its namespace in the prefix header of the mapping. In the example above, the FOAF ontology[1] is referenced that way, and all FOAF concepts hence are made available for further mappings. EMF class mapping examples are given, such as for *Museum* and *Director*, which are respectively mapped to the museum class in the novel ontology and to Person in FOAF. A class mapping comprises the following clauses:

[1] http://foaf-project.org

- **uriPattern:** specifies the URI of the RDF resource corresponding to the ontology class target. The URI definition is specified via an OCL expression that returns a String. OCL allows browsing the domain model for pruning relevant attributes on which OCL functions can be applied for eventually build the required URI pattern.
- **subClassOf** is an OWL keyword (from the ontology matching language) and here indicates that the mapped element in the new ontology refers to a yet existing ontology concept. Other OWL keywords can be used to express other appropriate kinds of relationship. (e.g. : *museum* in our *museum* ontology is a kind of *Feature* in the *geonames* ontology)
- **properties**: specifies the correspondences between properties of the model class (attribute, reference) and the ontology properties. The property mapping clause distinguishes between *map attr* that links an *attribute* to a *datatypeProperty* and *map ref* that links a *reference* (association) to an *objectProperty*.

4.1.2 Complex Mapping

A complex mapping represents a 1-N or N-1 correspondence between model and ontology elements. Our language, accepts these mappings in a simple way. Complex mappings appear as a series of simple mappings being defined within a same context (see Listing 2 where museum is mapped twice).

```
map class Artwork (self.kind = #Painting) with museum:Painting {
  uriPattern = "http://museum/painting/" + self.name;
  properties = {
      map attr name with dbpedia:title;
      map attr creationYear with museum:creationYear;
      map ref museum with dbpedia:museum;
      map ref museum with geonames:locatedIn;
      map ref hasArtist with dbpedia:artist;
  }
}
map class Artwork (self.kind = #Sculpture) with museum:Sculpture {…}
map class Artist(self.creates->forAll(e | e.kind = #Painting))
with museum:Painter {
  uriPattern = "http://museum/person/"
             + self.firstName + self.lastName;
  subClassOf = {foaf:Person}
  properties = {
      map attr firstName with foaf:firstName;
      map attr lastName with foaf:family_name;
  }
}
map Artist(self.creates->forAll(e | e.kind = #Sculpture))
 with museum:Sculptor {…}
```

Listing 2: A complex mapping example

All Class mapping clauses with source *Artwork* belong to the same context and then define a complex 1-N mapping. More precisely, the Artwork domain class is split into two ontology target classes that distinguish between *painting* and *sculpture*. The opposite mapping is consequently typed N-1. In such complex mappings of classes, the selection of the subsets that define the concrete classes in the ontology is achieved by means of OCL constraints whose context is the source class in the domain model. Complex mappings may also concern properties. For instance, in Listing 2, the *museum* reference in the *Artwork* class is mapped to both *dbpedia:museum* and *geonames:locateIn* properties. This 1-N mapping allows setting two distinct views (cultural *vs* geographical) upon the same domain model property.

4.2 Weaving between Domain Models and Ontologies

The language presented here, conforms to a textual syntax that aids an expert in specifying correspondences between the domain model and an ontology. We ground our proposal upon MDE principles. The mapping is viewed as a special model, namely a weaving model that records the set of correspondences. Building the weaving model involves several model transformations.

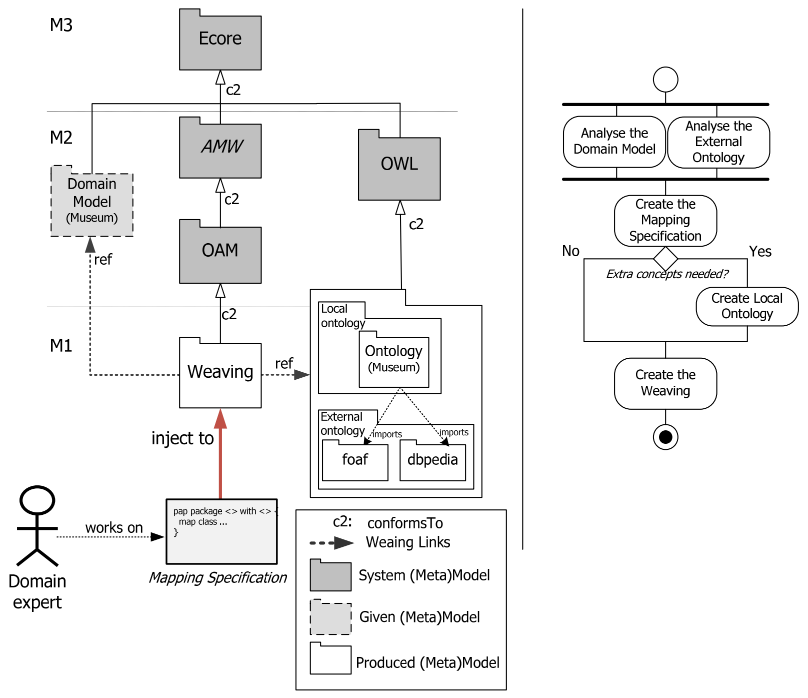

Fig. 4: Several processes are combined for building the mapping file.

1. The mapping possibly is fully defined by the expert. Nevertheless, a basic default mapping may be provided on request. It results from applying standard transformation rules from UML to OWL (UML2OWL[2]), as given in the ODM specification [17]. We implemented these rules *via* a model transformation

[2] http://www.eclipse.org/m2m/atl/usecases/ODMImplementation

process. Starting from this basic original mapping model, the expert is then only in charge of validation and extension tasks, so as to build the final mapping he wants.

2. In case existing ontologies are not sufficient to provide all needed concepts underlying the domain model referenced in the mapping and of interest for data publication, a novel local ontology should be generated. This is achieved by a model transformation that takes both the domain model and the mapping model as inputs and provides the desired OWL ontology as an output.
3. The last step consists in translating the mapping model to a weaving model representation. We use the AMW model weaver environment to implement the weaving process. So, we had to build a weaving metamodel specific to AMW. This process merges a series of transformations (text to model, model to model).

5 Populating the knowledge base

Up to now, we worked at an abstract level, dealing with classes and concepts. From now on, in order to effectively publish data on the Semantic Web, concrete data should be made reachable through a web interface. The natural solution is to make an import from the data source into an RDF repository so that the information to publish could be accessed *via* a web browser. We argued for putting the domain model as a pivot in our method, in consequence, the importation splits up in several consecutive operations we sketch hereafter. First we provide an object view upon the data in the data source according to the domain model, and then we give an ontology view upon the object data. Views are defined as mappings at the data level that lead to weaving models. The weaving models permit to generate transformation rules that can perform the importation process. Using such a high-order transformation is mandatory since the object model at hand cannot be known *a priori*.
The two following subsections respectively address one of the two steps described above. The main benefit of MDE is that it allows linking transformations and dealing in a modular way with the numerous models we need here, with no significant increase of the complexity. Besides, it supplies traceability of the transformations. The example deals with relational persistent data, but dealing with – say – XML data would only require fitting the mapping of the domain model to the new persistent format, all other concerns remaining unchanged.

5.1 From Object data to RDF data

The first step concerns the translation of objects (represented at the M1 layer) into RDF resources. Here we see the benefits of putting the domain model at the M2 layer. Doing so, we have a clear separation between objects (M1 layer) and domain classes (M2 layer). This leads to a simplification of the transformation rules specification. Otherwise, we would have been faced to dealing with UML domain classes and objects at the same level (M1 layer).
The implementation is done using a high-order transformation technique, in order to generate the required transformation. Let *MM2RDF.atl* be the transformation that

converts a M1 model into an RDF model. As depicted in Fig 5, *MM2RDF.atl* is generated by the high-order transformation *OAM2ATL.atl*. The latter takes the mapping model and its related models (domain model and ontology model) as input. It is here made clear that the use of a high-order transformation is inescapable since *MM2RDF.atl* rules depend of the domain model which is a variable model.
The role of *OAM2ATL.atl* is to screen the weaving model, and for each class mapping clause (map class), it generates a specific ATL rule in the *MM2RDF.atl* file. This resulting ATL transformation extracts an RDF data representation from a model conforms to the domain model.

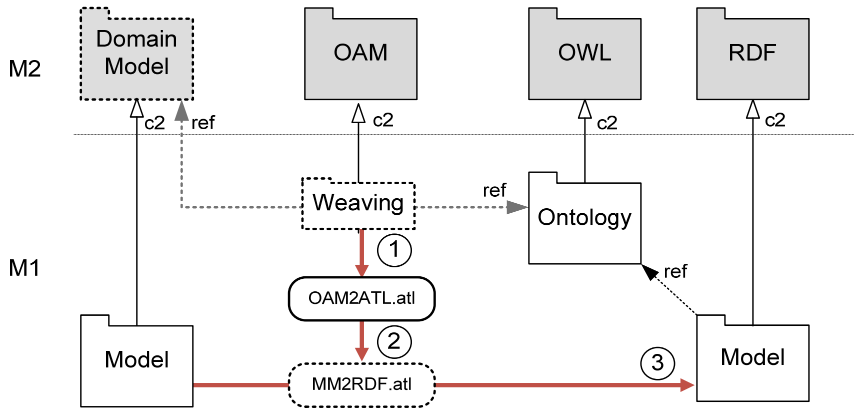

Fig. 5: Model Transformation process for obtaining an RDF repository from a model.

The following listings show some significant excerpts of the *MM2RDF.atl* transformation. Listing 3 shows how an instance of the Museum class is translated into an RDF Resource. For each domain model class that appears in the mapping, an equivalent rule is generated. The RDF metamodel used in this transformation is derived from the ODM specification, and has been extended in order to ease transformation specifications. Each RDF resource is the subject of some statements (see Line 7). For instance, the lazy rules, *makeDataStatement* and *makeObjectStatement*, respectively create the desired statement from a class attribute, and from an association (see Listing 4).

```
1    rule Museum2Resource {
2      from m : Museum!Museum
3      to r :
4        RDF!Resource (
5          uri <- m.getURI(),
6          subjectOf <- Sequence { type,
7            thisModule.makeDataStatement(m, m.name, 'name'),
8            m.artworks->collect(e |
                     thisModule.makeObjectStatement(m, e, 'artworks')),
9            thisModule.makeObjectStatement(m, m.city, 'city'))}
10       ),
        type : RDF!Statement ( ... )
```

Listing 3 Generated ATL code excerpt for translating a Museum instance into an RDF Resource.

```
lazy rule makeObjectStatement {
  from s : OclAny, o : OclAny, pname : String
  to r : RDF!Statement (
          subject <- s,
          predicate <- p,
          object <- o
         ),
      p : RDF!Property ( ... ),
 }
```

Listing 4: Statements and properties are created by lazy rules.

5.2 From Relational data to Object data

In this paper, we assume that a relational database is *in situ* available, whose schema has been obtained by applying an object-relational mapping upon the domain model and the database schema. In our current implementation, we rely on the Hibernate[3] framework, for the object-relational mapping definition (more precisely the TENEO[4] project which is dedicated to EMF persistency). The weaving model that represents the object-relational mapping is thus obtained from the Hibernate mapping definition.
The database schema is captured by applying a schema discovery process with the help of the Hibernate framework. It allows defining a projector from the database onto the MDE technical spaces. The concrete data are modelled accordingly, by sending queries to the database. We then apply a model transformation similar to the one presented in section 5.1. The last step that imports relational data into an OWL knowledge base can now be achieved through exploiting the sole object-ontology mapping.

6 Querying the RDF Store

The process we presented above publishes data as an OWL knowledge base. It can be viewed as implementing a data warehouse *via* the Extract-Transform-Load principle (ETL) [13], in which resident data are translated in order to fit the target system format. In our approach, the target system is an OWL ontology whose content is made of RDF triples. Data access is achieved by use of a RDF query language, e.g.: SPARQL [19]. An example of such a SPARQL query is given hereafter. Let's suppose the user wants to list artworks present in "Le Louvre".

```
prefix dbpedia: <http://dbpedia.org/property/>
```

[3] http://www.hibernate.org
[4] http://www.elver.org/

```
prefix museum: <http://museum#>
select ?art
where {
   ?art dbpedia:museum ?museum .
   ?museum museum:name "Le Louvre"
}
```

Listing 5: A SPARQL query on the museum ontology.

Here, we assume that the knowledge base has been generated and completely populated from an existing database. Performing a query then only involves the knowledge base that finally can be a RDF file or a Sesame[5] RDF database.

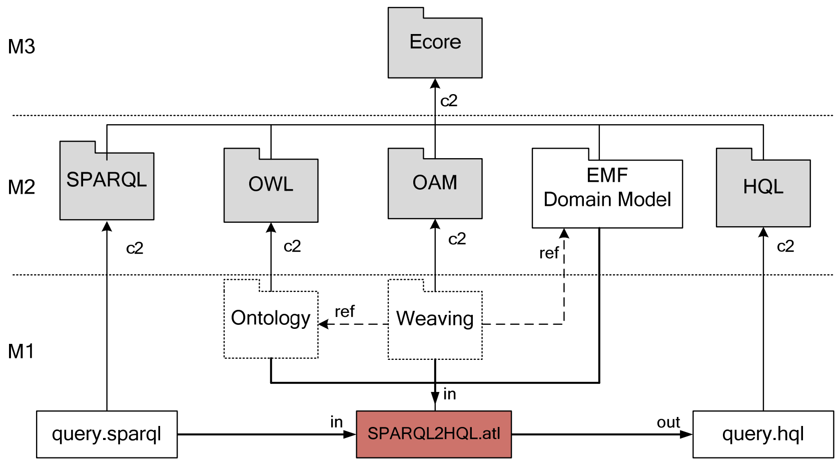

Fig. 6: Model Transformation process for Query rewriting

However, translating and importing the whole data base into an RDF repository, does not seem - in most cases - the best solution, since it implies to replicate data and carry out a synchronization process with heavy redundancy. We advocate for a better practice, which directly answers the ontology query by querying the original data source. Implementing this approach led us to specify an automated rewriting process that relies on the specification of the chain of mappings and finally translates a SPARQL query into a set of SQL statements.
The rewriting process naturally is developed within the MDE paradigm and potentially applies to any persistent data source format. Fig. 6 gives an overview of the series of model transformations that lead to the result. More precisely, on the museum example, the initial SPARQL query is first translated into an equivalent HQL query upon the domain model, which in turn is rewritten as a SQL query on the relational database. The model transformations that are responsible for that rewriting process take both the object-ontology and object-relational mappings and weaving models into account, in order to exploit the element correspondences for translating

[5] http://www.openrdf.org

the terms of one query into its counterpart in the target language. For example, the SPARQL query in Listing 3 gives the corresponding HQL query in Listing 4.

```
select art
from Museum museum, Artwork art
where art.museum = museum and museum.name = 'Le Louvre';
```

Listing 6: HQL query resulting from a rewriting process from SPARQL.

The resulting HQL query can be rewritten by Hibernate according to the data source format (e.g.: SQL).

7 Related Works

The works connected to our approach belong to various but complementary domains. Those that treat of the UML and RDF(S) correspondence [7] [8] [18], and especially the ODM specification from the OMG [17] that sets up basic useful although limited transformation rules, helped us in defining the abstract syntax of our mapping language between an object model and an ontology. Linking metamodels and ontology through weaving models has also been proposed by Klapper [12] who suggests the use of ontology alignment techniques in order to automatically build the weaving model. This seems of interest for us in case we try to map a model to an existing ontology, but is out of concern when creating a new ontology is the goal.
Another kind of related works treats of data integration by means of an ontological view, and more generally of the semantic web. The question still is to implement RDF knowledge bases providing access to existing data source. Bizer with D2RQ [3], Chen [5] and de Laborda [14] come with works having goals very close to ours. They propose a direct mapping of the knowledge base to the data source. Conversely, a part of our contribution is to rely on the domain model (and not on the persistence model) as a pivot element in the process of correspondence design.

8 Conclusion

This paper grounds on the MDE technique and presents a contribution that aids publishing data on the semantic web. A complete process for building a RDF/OWL knowledge base from existing possibly heterogeneous and distributed databases is specified. Our proposal is semi-automated, but the expert is only involved in the mapping definition step (between the domain model and the ontology). One novel aspect of our work is that we give a prominent role to the domain model in the overall process. It comes as a pivot model standing as a midterm between the data source and the ontology. It embodies a sizeable part of the business knowledge and bears more information about the domain terminology and concepts than – for instance - the flat relational model that generally results from complex normalization and de-normalization processes, far from any semantic concern. It also keeps simpler and focuses on permanent and essential features, while an ontology is intended to also account for subsidiary concerns.

Finally, when splitting the mapping process by introducing the intermediate domain model, we gain in modularity, and robustness. Transformations are more explicit and simpler since the gap is less at each step. We also gain in being less dependent on the persistency technology. In case the database schema is modified, only one transformation step is involved and subject to update.

Our proposal not only permits to create an ontology that describes exported data, but also to create data that conform to an existing ontology. In order to get free from controlling the consistency between the database and the RDF triples, we presently implement an automated 'on the fly' rewriting process that translates SPARQL queries into SQL or XML queries. Consequently, the data can be kept in their original persistence system, with no replication in an RDF repository, while being subject to the queries a user can directly express in terms of the ontology.

References

1. Bauer, C., King, G.: Java Persistence with Hibernate. Manning Publications, 2007.
2. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic Web. Scientific American 284 (2001) 28-37
3. Bizer, C., Seaborne, A: D2RQ-Treating Non-RDF Databases as Virtual RDF Graphs. 3rd International Semantic Web Conference (ISWC2004), Hiroshima, Japan (2004)
4. Castro, P., Melnik, S., Adya, A.: ADO.NET entity framework: raising the level of abstraction in data programming, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2007, pp. 1070–1072.
5. Chen, H., Wang, Y., Wang, H., Mao, Y., Tang, J., Zhou, C., Yin, A., Wu, Z.: Towards a Semantic Web of Relational Databases: a Practical Semantic Toolkit and an In-Use Case from Traditional Chinese Medicine.
6. Del Fabro, M.D., Bézivin, J., Jouault, F., Breton, E., Gueltas, G.: AMW: A Generic Model Weaver. Proc. of the 1ères Journées sur l'Ingénierie Dirigée par les Modèles (2005)
7. Djuric, D., Gasevic, D., Devedzic, V.: Ontology Modeling and MDA. Journal of Object Technology 4 (2005) 109-128
8. Falkovych, K., Sabou, M., Stuckenschmidt, H.: UML for the Semantic Web: Transformation-Based Approaches. Knowledge Transformation for the Semantic Web 95 (2003) 92-107
9. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.
10. Hu, W., Qu, Y.: Discovering Simple Mappings Between Relational Database Schemas and Ontologies. ISWC/ASWC 4825 (2007) 225-238
11. Jouault, F., Kurtev, I.: Transforming Models with ATL. Model Transformations in Practice Workshop at MoDELS Vol. 3844, Montego Bay, Jamaica (2005) 128–138
12. Kappel, G., Kapsammer, E., Kargl, and al., M.: Lifting metamodels to ontologies: A step to the semantic integration of modeling languages. ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, Genova, Italy (2006)
13. Kimball, R., Margy, R.,. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 2nd edition, Wiley, 358-362. (2002)
14. de Laborda, C.P., Conrad, S.: Bringing Relational Data into the SemanticWeb using SPARQL and Relational. OWL. IEEE Computer Society Washington, DC, USA (2006)
15. Larman C., Applying UML and Patterns, An introduction to Object-Oriented Analysis and design and The Unified Process 2nd edition, Prentice Hall, 2001

16. Oren, E., Delbru, R., Gerke, S., Haller, A., Decker, S.: ActiveRDF: object-oriented semantic web programming. Proceedings of the 16th international conference on World Wide Web (2007) 817-824
17. OMG. Ontology Definition Metamodel OMG Adopted Specification, November 2007. http://www.omg.org/docs/ptc/07-09-09.pdf
18. Parreiras, F.S., Staab, S., Winter, A.: On marrying ontological and metamodeling technical spaces. Proceedings of the 6th joint meeting of the european software engineering conference and the 14th ACM SIGSOFT symposium on Foundations of software engineering (2007) 439-448
19. Prud'hommeaux, E., Seaborne, A., others: SPARQL Query Language for RDF. W3C Recommendation (2008)