An approach to object-ontology mapping

Peter Bartalos and Mária Bieliková

Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava
Ilkovičova 3, 842 16 Bratislava, Slovakia
{bartalos, bielik}@fiit.stuba.sk

Abstract. The use of ontologies has an increasing tendency in new web-based application development. Although the ontological repositories offer APIs, a more convenient way is a use of objects which handle data. To take the advantage of the use of objects in the application with the ontological domain model, it is necessary to perform the mapping between ontological instances and objects. In this paper we present an approach dealing with the automatic creation of the class model (in the sense of object-oriented paradigm), which handles data. We also discuss two approaches of mapping between ontological instances and objects. Our approach is evaluated in two domains – online labor market and scientific publications within a portal offering information based on particular domain.

1 Introduction

The explosive development of the Web has brought forward the need for machine processable representations of semantically rich information: a vision at the heart of the Semantic Web [3]. In presence the relational algebra and relational databases stands as the most popular data representation and storage. This follows from the huge theory behind relational calculus and relational model design, good support on application level (available APIs, mappers, etc.), good performance of relational databases. In spite of the advantages of relational databases, the use of upper level ontologies may bring many benefits. These come from their higher expressivity, flexibility to changes, shareability between different groups working in the same domain, possible reasoning over the ontology to acquire additional information.

Having software application where ontologies are used, we often need to work with data of concepts stored in an ontological repository (i.e., instances of ontological classes). In such case the use of APIs of existing ontological repositories is pretty difficult. In applications developed with object-oriented approach, it is more convenient to work directly with objects holding data to be processed. In such a case an object ontology mapper is necessary to perform transformations between these two representations.

The use of an object-ontology mapper is useful in applications developed using object-oriented approach, where ontological data model is created and

ontological repositories are used for persistent data store (instances of classes). In this paper we deal with automatic mapping ontologies to object-oriented representation and vice versa. We discuss the creation of classes which can store data of ontological entities and present a method for generation such classes automatically from the ontology. We also discuss two approaches of the mapping between ontology instances and objects (instances of classes of an object-oriented programming language).

The rest of the paper is structured as follows. Section 2 presents overview of portal solutions including the framework for the ontology-mapping described in this paper. In section 3 we give an overview of the object ontology mapping. Section 4 is devoted to the creation of the object-oriented model of an ontology, i.e., the bean generation. In section 5 we deal with the mapping between ontological instances and objects. Finally, section 6 contains the evaluation of proposed approach and conclusions.

2 Portal solutions for the Semantic Web

The use of ontologies plays an important role in applications based on the Semantic Web technologies. We name some of the most known recent approaches developed in the course of research projects and can be considered as a motivation and inspiration of our work. OntoPortal uses ontologies and adaptive hypermedia principles to enrich the linking between resources [8]. The AKT project aims to develop technologies for knowledge processing, management and publishing, e.g. OntoWeaver-S [9], which is a comprehensive ontology-based infrastructure for building knowledge portals with support for web services. The SEAL [11] framework for semantic portals takes advantage of semantics for the presentation of information in a portal with focus on semantic querying and browsing. SOIP-F [12] describes a framework for the development of semantic organization information portals based on "conventional" web frameworks, web conceptual models, ontologies as well as additional metadata.

Our work has in common with most of mentioned approaches that we work on developing methods for information acquisition, analyzing, organizing and personalized presentation in specific domains employing ontologies in portal solutions. However, while various support for creation of adaptive web-based portal solutions is provided, ontological representation is not supported in appropriate level of abstraction. Issues concerning the changeability of open information spaces should be addressed with respect to effective portal development and maintenance. We have developed a framework for the creation of adaptive web-based portal solutions that considers mentioned issues [2]. It is used as an integration and presentation platform for our tools, developed for the support of the Semantic Web.

Application domains where we experiment with proposed methods are job offers ([10], `nazou.fiit.stuba.sk`) and scientific publications ([4], `mapekus.fiit.stuba.sk`). In both domains a framework for portal building is used. *Job Offer Portal* (JOP) offers its users ways of navigation through the information

space of job offers using several presentation tools, which present job offers stored in ontological base acquired by a chain of data harvesting tools that acquire and process data from the Internet. Employers also have the possibility to submit new job offers using a set of forms generated by the framework based on the currently used ontology. *Publication Presentation Portal* (P3) serves for personalized presentation layer for digital libraries. It uses meta-data about scientific publications and aids users in personalized navigating within the publications information space.

Fig. 1 depicts an overview of the common architecture of portals developed using our framework (for more details see [2]).

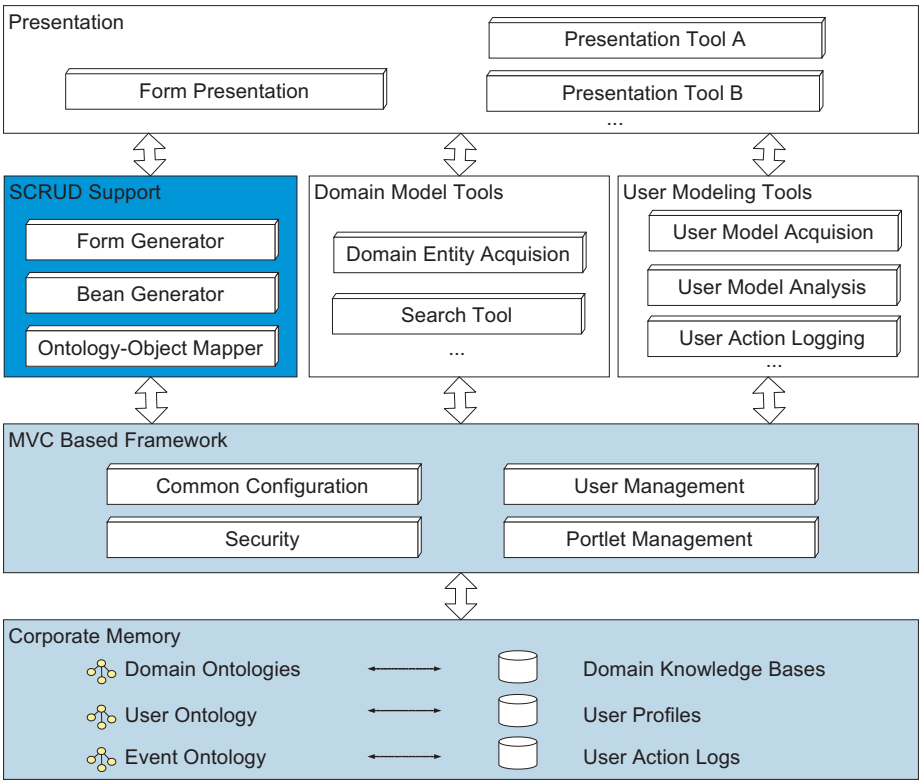

Fig. 1. Portal architecture

At the bottom the corporate memory is placed which stores the domain, user and event ontologies [5]. The second layer includes the common modules for configuration, security, user management and portlet management. The next layer contains different functional modules. First, the *SCRUD support* component, which employs the methods we deal with in this paper. Second, the domain model tools with various functionality required in the particular domain. Third,

the user modeling tools for the acquisition, analysis of the user model. The top layer contains the presentation tools.

The first two layers together with the *SCRUD support* module are common for each portal developed with the framework. The remainder parts (not highlighted in the figure) include software tools which are tailored (or just reconfigured) for the given domain.

The *SCRUD support* component performs the persistent operations over the data entities of the domain. It includes the bean generation module, which creates the class model based on the ontology, see Fig.2. Instances of these classes are used to manipulate with data stored in the ontology.

The *SCRUD support* offers methods realizing the persistent storage and retrieval of the objects in the ontological repository. During the storage of an object to the repository, the object to ontology transformation method is employed. The ontology to object transformation is called when an already stored instance is retrieved from the repository.

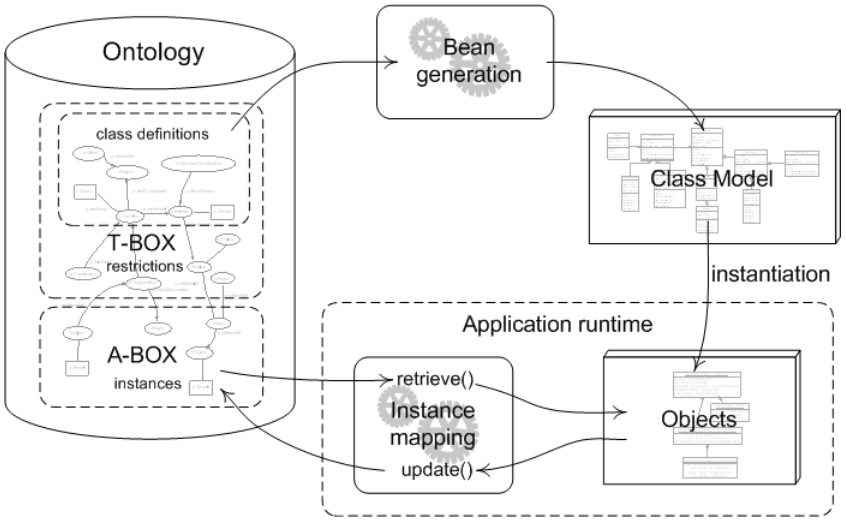

Fig. 2. Mapping utilization.

3 Overview of Object-Ontology mapping

The mapping of an ontology into object-oriented model is a technique, where we create a model of the ontology in object-oriented paradigm. The motivation is to manipulate with data stored in an ontology using objects. Majority mapping principles is independent of the used programming language. Some differences may occur depending on the support of multiple inheritance. The multiple inheritance problem is solved using interfaces, when the given object-oriented language does not support it.

Ontology is a conceptualization of a domain. It consists of concepts, relations between concepts, restrictions and is described in one of the ontological languages such as the upper level OWL. It can be also considered as a data model. In this case the T-box of the ontology stands as a schema and data of the concrete entities are caught in the A-box. Ontological and object-oriented representations differ in their expressivity. Ontologies have higher expressivity [6]. For example, in ontologies a membership of an entity to the class can be expressed using restrictions but in object-oriented paradigm not. In OWL it is also possible to define hierarchies of properties but in object-oriented environment not. There are other situations where we see that object-oriented approach is more limited.

In some sense object-oriented representation is similar to the ontological. In object-oriented approach we have objects with their behavior. Between objects exist various relations similarly to ontology classes. If we only need to store the values of the properties of ontological classes, we do not need to represent the whole complexity of ontological representation (whole schema, restrictions etc.). In such case we can afford to represent the ontology by object-oriented paradigm to work with the A-box. In this case we do not cover those aspects of ontologies that are intended for example for reasoning, we just hold information about instances in the objects to be able to manipulate them in comfortable way.

The basic idea of the object-ontology mapping is to create a set of classes in such a way that each ontological class has its equivalent in a class of selected object-oriented programming language (the use of interfaces can be necessary when the language does not support multiple inheritance). One example of such a mapping is shown in Fig. 3 (see Section 4).

In Table 1, the relations between elements of object-oriented paradigm and ontologies are shown. The creation of the class model can be performed automatically, i.e., generated. Classes aggregate other classes which correspond to object-type properties of ontological classes (e.g., the *JobOffer* class has attribute *hasSalary* of type *Salary*, see Fig. 3). Data-type properties of an ontological class have their corresponding attributes in the program class (e.g., *xmls:float* is mapped to the *Float* Java class), see attribute *amount* of the *Salary* class in Fig. 3. An instance of the ontological class is mapped to an instance of the corresponding program class (the object of that class). One example of the mapping is shown in Fig. 4 (see Section 5).

Table 1. Relation between elements of the object-oriented paradigm and ontologies

Ontology		**OO environment**
class	$\longleftrightarrow$	class
class instance	$\longleftrightarrow$	object
property	$\longleftrightarrow$	attribute
subject	$\longleftrightarrow$	ID property[a]
predicate	$\longleftrightarrow$	attribute name
object	$\longleftrightarrow$	attribute value

[a] explained in section 5

An object-ontology mapper is similar to an object-relational mapper [1] (one known implementation is *Hibernate* `www.hibernate.org`). It performs the transformation of the instances of the ontological classes into objects and vice versa, i.e., fills up the values of the corresponding properties (see example in Fig. 4).

The idea of object-ontology mapping is not new. In [7] the basic ideas of creation of a class model in Java, which encompass the OWL ontology are described. We have extended this approach to employ the object-ontology mapping.

4 Bean generation

We have proposed a method for a Java bean generation, which automatically generates required classes (beans) and interfaces. For each ontological class it generates a source code of the equivalent class (and necessary interfaces) in Java programming language. The class contains attributes equivalent to ontological class properties with *get*, *set*, *add* methods (*add* method is necessary if the property has multiple cardinality, to add an item to a list representing the property).

The name of the bean and corresponding interface is the same as the name of the corresponding ontological class, see part 1 on Fig. 3. It is necessary to take into consideration the fact that there may exist two classes with the same name varying in the prefix (namespace). To ensure that each ontological class name is mapped to a unique bean name, we include the prefix into it – hash value of the prefix (in examples the hash is not included to preserve simplicity). The result is used as the bean name and also as the file (containing the source code) name.

To be able to determine the type of the instance (i.e., the class of the instance) we added a *static final* attribute *type* to each class. Its value is set to the ontological class name including prefix. It is hard coded during the generation process and is used to set the instance type when an object to ontology mapping is performed (i.e., a triple $\langle$ *value of attribute ID, rdf:type, value of attribute type* $\rangle$ is added to the created RDF graph), see part 2 in Fig. 3.

The bean attributes names are created similarly from the hash of the prefix and the property name, see part 3 in Fig. 3. The type of the attribute is in the case of object-type property determined as a bean which represents the property (part 4). In the case of data-type property, the classes of the Java framework are used (class Integer, Float, etc.). For each attribute there exists a subsistent *get*, *set* and if the property is multiple also the *add* method (part 5). These methods are included also in interfaces (part 6).

Different class relations are reflected in the object-oriented model such as inheritance. The detailed description how these relations are considered in the object-oriented model can be found in [7]. In our example in Fig. 3 it is shown how the *rdfs:subClassOf* relation between classes *jo:Offer* and *jo:JobOffer* is reflected in the object-oriented model, part 7. In this case, the interface *JobOffer* extends the interface *Offer*. If a united interface is created (for example when *owl:equivalentClass* relation is mapped its name is a composition of names of the corresponding classes including prefix.

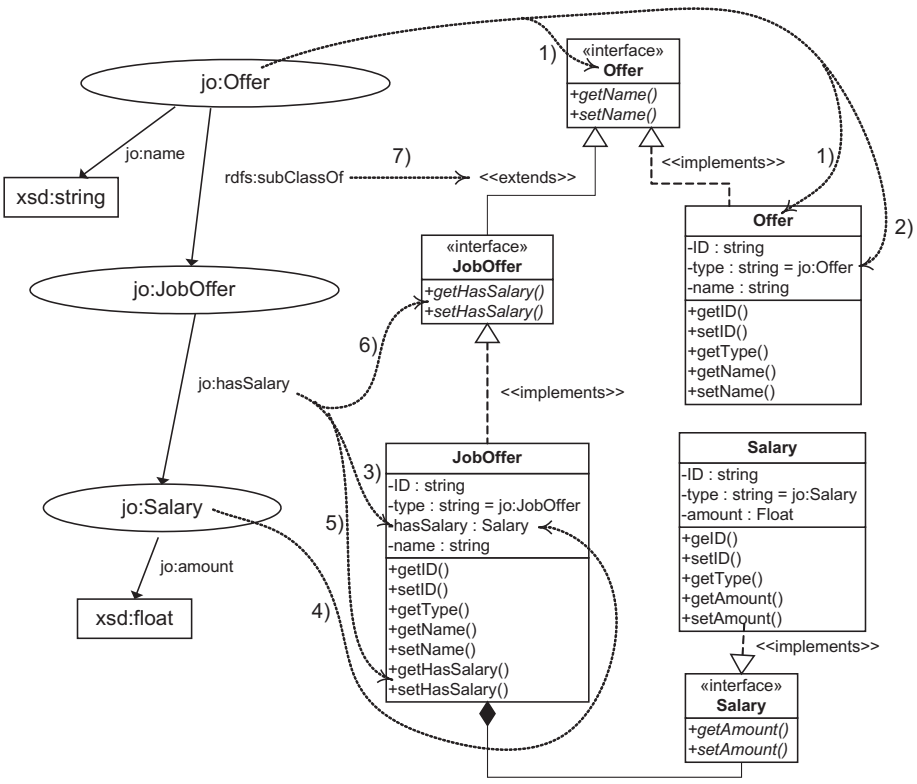

Fig. 3. Mapping of the T-box to class model

The OWL language defines the *owl:Thing* superclass. This class is a superclass of each class defined in the ontology. To bring this aspect into the generated class model, we define an super–interface which must be extended by each other generated interface and implemented by each generated class. This is then used in the case when some property has not defined its range. In this case, the corresponding attribute in the class is of type of the super–interface. Because of polymorphism, any instance of the generated classes can stand as that attribute value. In the case that a property has not defined domain, each class has a corresponding attribute.

The bean generation process proceeds in two phases:

1. The first phase – analysis – is necessary to find out the relations between ontological classes (*rdfs:subClassOf*, *owl:equivalentClass*, etc.). This information is then used to build correctly the necessary interfaces. Also the properties of the ontological classes are identified in the analysis phase.
2. In the second phase – file building – the Java interface and class source code files are generated. Firstly, it is determined which interfaces must the class

implement and then for each property, an attribute with access methods is added to the class. For each class, also the related interfaces are generated. Finally, serialization into source code is performed.

5 Instances mapping

The Java beans generated by the algorithm described in Section 4 are used to store data about the ontological entities. To make a use of these beans effective, we need a mechanism performing transformation between instances of Java beans (objects) and ontological instances. This process can be performed in two ways:

- the first approach is a development of an add hoc mapper (coupled to the concrete ontology) and
- the second one is a use of universal mapper (universal in the sense of independency of the ontology).

Both approaches perform the transformation between ontological instances and corresponding objects, see Fig. 4. This transformation includes the creation of an object with filled attributes (based on the mapped RDF graph) when the ontology instance is mapped to the object. On the other hand, the transformation includes the creation of an RDF graph describing the entity whose object representation was mapped to ontological.

A special property *ID* is used to store the URI of the ontological instance (this URI is also the URI of the subjects of each triple describing the entity) – for example *jo:jobOffer_0123* in Fig. 4. When object to ontology mapping is performed with new entity, it is necessary to create an URI for it. This URI contains the common prefix of the ontology and a string of randomly generated GUID. If an entity is only updated (it already exists in the ontology), it is necessary to remove the old data from the ontology, which creates a need to delete the RDF graph related with the URI of the instance (this has to be done recursively to object properties).

5.1 Add hoc mapping method

An add hoc mapper performs transformations between objects and instances of ontological classes using hard coded methods. Such a method realizes the mapping between all data-type properties of an entity and calls other similar methods to map the object properties (note that object properties represent other entities, so it calls a method for mapping that entity).

Disadvantage of this add hoc approach is the coupling to the given ontology and the need of specify the applicable transformation method when mapping is required. The advantage is that it is more effective (in the sense of performance) than a universal method. Although for each ontology we need a special mapper, its creation can be automated, i.e., the add hoc mapper can be generated.

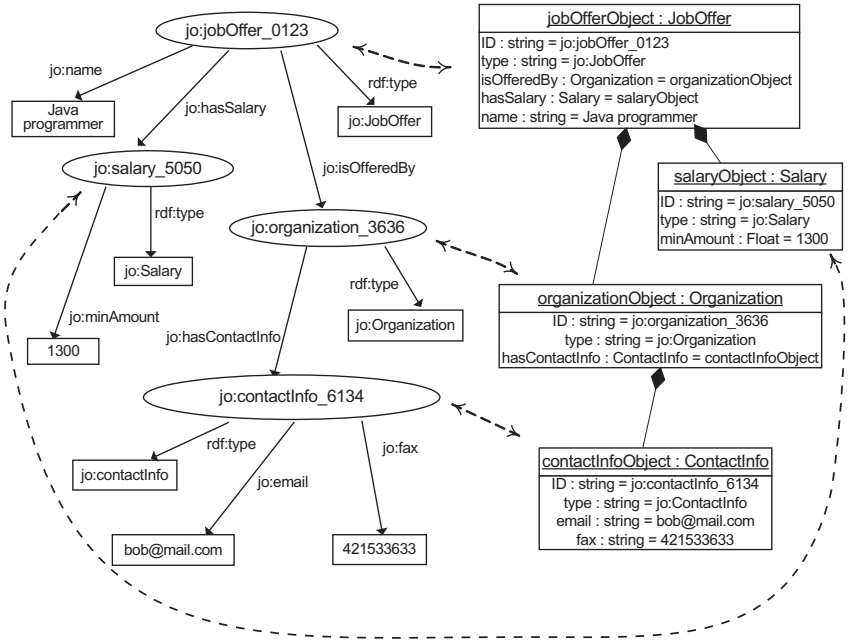

Fig. 4. Mapping between ontological instances and Java objects.

Mapping ontology to objects. The pseudo code of the method which maps RDF triples of job offer entity to job offer object (see Fig. 4), is presented as Algorithm 1. It contains simple loop mapping each RDF triple to corresponding attribute values. In each round, based on the predicate it finds the corresponding attribute and sets its value. If the property is an object property, it calls the proper method to map it into aggregated object.

Mapping objects to ontology. The pseudo code of the method mapping job offer object to RDF triples is shown in Algorithm 2. It consists of a loop creating an RDF triple for each attribute. If the attribute corresponds to an object property, the proper method is called to create the RDF graph of entity (entity which is the value of the property), which is added to the resulting graph.

5.2 Universal mapping method

The universal mapping method performs a mapping between any given ontology (any RDF graph) and corresponding objects. As each time the same method is used, it is not needed for the mapper to specify which entity is mapped (like in the case of add hoc mapper). The universality is possible because of the reflection. Using the reflection it is possible to invoke *get*, *set* and *add* methods over the objects.

Algorithm 1 MapJobOffer *Input*: rdf_graph *Output*: object

```
object.setID(root_node.URI)
for all triple related to root_node of rdf_graph do
  if triple.predicate is isOfferedBy then
    object.setIsOfferedBy(MapOrganization(triple.object))
  else if triple.predicate is hasSalary then
    object.setHasSalary(MapSalary(triple.object))
  else if triple.predicate is name then
    object.setName(triple.object)
  end if
end for
return object
```

Algorithm 2 MapJobOffer *Input*: object *Output*: rdf_graph

```
for all attribute of object do
  if attribute is isOfferedBy and not null then
    triple = createTriple(object.getID(), jo:isOfferedBy, attribute.value.getID())
    graph.addTriple(triple)
    graph.addGraph(MapOrganization(attribute.value))
  else if attribute is hasSalary and not null then
    triple = createTriple(object.getID(), jo:hasSalary, attribute.value.getID())
    graph.addTriple(triple)
    graph.addGraph(MapSalary(attribute.value))
  else if attribute is name and not null then
    triple = createTriple(object.getID(), jo:name, Java programmer ^ xsd:String)
    graph.addTriple(triple)
  end if
end for
return graph
```

A disadvantage of this approach is that it is slower than add hoc mapping (just because of the reflection). The advantage is that each time the same mapper with the same transformation method is used.

Mapping ontology to objects. The mapping of the RDF graphs into Java objects is a recursive process, which begins at the root node of the RDF graph, see pseudo code in Algorithm 3 (the root node is the URI node of the mapped entity). In one call of recursion, each property related to the root node is processed. First, the attributes of a Java object which represent a data-type property of the ontological class are set to the corresponding value that is retrieved from the object of the corresponding RDF triple. Also those attributes which represent object properties of an ontological class are set to newly created Java objects. The attributes of these new objects are then set in the recursive call, where the root node is represented by a node representing the object property.

Algorithm 3 RDF2Object *Input*: rdf_graph *Output*: object

```
object.getID() = rootNode.URI
for all triple related to root_node of rdf_graph do
  if propertyTypeOf(triple.predicate) is data-type property then
    object.setAttribute(triple.object)
  else {property is object-type property}
    object.setAttribute(RDF2Object(property))
  end if
end for
return object
```

Mapping objects to ontology. The mapping of the Java objects to the RDF graph is also recursive process, see pseudo code in Algorithm 4. In one call of recursion, attributes of one Java object are transformed to corresponding RDF triples. Those attributes which represent object properties of ontological classes are transformed in recursive calls. Here a corresponding subgraph is created and then added to the existing one.

Algorithm 4 Object2RDF *Input*: object *Output*: rdf_graph

```
triple = createTriple(object.getID(), rdfs:type, object.getType())
graph.addTriple(triple)
for all attribute of object do
  if propertyTypeOf(attribute) is data-type property then
    triple = createTriple(object.getID(), attribute.name, attribute.value)
    graph.addTriple(triple)
  else {property is object-type property}
    subgraph = Object2RDF(attribute)
    graph.addGraph(subGraph)
  end if
end for
return graph
```

6 Evaluation and conclusions

In this paper we have presented an approach to object-ontology mapping. Our work is a step forward to make the ontologies more usable in new web-based applications for the Semantic Web. The relational model and storages have currently undisputable benefits towards ontological approach and will not be replaced by them. There is no doubt of existence of situations when the use of upper level ontologies can be very useful. Those applications can benefit from the higher expressivity power, semantic enrichment of data, shareability and reasoning possibilities.

Methods presented in this paper are usable in any object-oriented language. We have evaluated it in Java programming language. It is applicable anywhere in the application, where data of ontological entity is required and processed. This can be done by instantiating the classes generated by the Java bean generator. These instances can be updated and retrieved to/from the repository by the object ontology instance mapper to realize persistent data store.

Java bean generator, which creates class source codes based on the proposed method has been developed. Also a universal mapper was developed which performs described mapping of instances. The bean generator and the instance mapper were successfully tested on ontologies developed within two research projects mentioned in section 2 for job offer domain and domain of scientific publications.

Our job offer ontology is a complex ontology represented in OWL with 740 classes (670 belong to taxonomies). It is filled with some 1 000 instances of manually filled job offers and several thousands instances provided by a wrapper from job offer sites on the Web. The scientific publications ontology contains 390 classes (360 belong to taxonomies) and several thousands of instances. Our approach was employed also with a user model ontology in mentioned projects.

The generator and instance mapper showed that an application can benefit from their use. Without automatized bean generation, the creation of the object-oriented model is hardly realized if the ontology is complex and contains many classes. Also the use of the universal instance mapper or the generation of an add hoc mapper is more convenient than the mapper development usual way.

Object-ontology mapping deals (similarly to the object-relational mapping) with a problem of decision, when to stop the loading process during mapping ontological instance to object (this is because the instance can be indirectly related through properties with a lot of other individuals in the ontology). This problem can be solved using lazy load technique. Our future work is to examine this solution and implement it into our mapper. We also want to further research the polymorphism of the object-oriented paradigm and ontologies. We will focus on the impact of the limitations of the object-oriented paradigm in applications.

Acknowledgments. This work was partially supported by the Slovak Research and Development Agency under the contract No. APVT-20-007104 and the Slovak State Programme of Research and Development "Establishing of Information Society" under the contract No. 1025/04.

References

1. S.W. Ambler. *Mapping Objects to Relational Databases: O/R Mapping In Detail.* J. Wiley & Sons, 2003.
2. M. Barla, P. Bartalos, M. Bieliková, R. Filkorn, and M. Tvarožek. Adaptive portal Framework for Semantic Web applications. In *2nd Int. Workshop on Adaptation and Evolution in Web Systems Engineering at ICWE 2007, Como, Italy*, 2007.
3. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, pages 34–43, 2001.

4. M. Bieliková and P. Návrat. Modeling and acquision, processing and exploiting of knowledge about user activities in the hyperspace of the Internet. In *Znalosti 2007, 6th Annual Conf., Ostrava, Czech republic*, pages 368–371, 2007.
5. M. Ciglan, M. Babik, M. Laclavik, I. Budinska, and L. Hluchy. Corporate memory: A framework for supporting tools for acquisition, organization and maintenance of information and knowledge. In J. Zendulka, editor, *9th Int. Conf. on Inf. Systems Implementation and Modelling, ISIM'06*, pages 185–192, Perov, Czech Rep., 2006.
6. M. Dumas, L. Aldred, M. Heravizadeh, and A. Hofstede. Ontology markup for web forms generation. In *Workshop on Real World RDF and Semantic Web Applications*, 2002.
7. A. Kalyanpur. Automatic mapping of OWL ontologies into Java. In *F. Maurer and G. Ruhe, Proc. of the 17th Int. Conf. on Software Engineering and Knowledge Engineering, SEKE'2004*, 2004.
8. S. Kampa, T. Miles-Board, L. Carr, and W. Hall. Linking with meaning: Ontological hypertext for scholars, 2001.
9. Y. Lei, E. Motta, and J. Domingue. Ontoweaver-s: Supporting the design of knowledge portals. In E. Motta et al., editor, *EKAW*, volume 3257 of *LNCS*, pages 216–230. Springer, 2004.
10. P. Návrat, M. Bieliková, and V. Rozinajová. Methods and Tools for Acquiring and Presenting Information and Knowledge in the Web. In *Int. Conf. on Computer Systems and Technologies, CompSysTech'05*, Varna, Bulgaria, 2005.
11. N. Stojanovic, A. Maedche, S. Staab, R. Studer, and Y. Sure. SEAL: a framework for developing SEmantic PortALs, 2001.
12. E. D. Valle and M. Brioschi. Toward a framework for semantic organizational information portal. In Ch. Bussler et al., editor, *European Semantic Web Symposium, ESWS 2004*, volume 3053 of *LNCS*, pages 402–416. Springer, 2004.