(S)CRUD Pattern Support for Semantic Web Applications

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 (and ontological representations such as OWL) has an increasing tendency in web-based applications development that employ semantics for automatic information processing. One of the most important operations in software applications are the those related with persistent store of the real world entities in repositories, commonly known as CRUD operations (Create-Retrieve-Update-Delete). Most of the applications are in present developed using object-oriented paradigm. In applications taking the advantage of the Semantic Web technologies, the CRUD operations are performed over ontological repositories extended with search operation. In this paper we present an approach to carry out these operations. It is based on the automatic object ontology mapping with bean generation and (semi)automatic form generation. Our approach is evaluated in two domains – online labor market and scientific publication within a portal offering information in particular domain.

1 Introduction

A typical software application deals with processing of data entities to realize the task for which it was designed. These data entities are commonly real word entities described in the computer readable form – goods, reservations, bank account and others. In most cases it is necessary to store them persistently to be able to work with them later. This implies that one of the most required and most important tasks of software applications is the ability to store, and when required, to retrieve data about real world entities.

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 [5]. In presence the relational algebra and relational databases stand 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. In spite of the advantages of relational databases, the use of upper level ontologies for meta-data representation may bring many benefits.

These come from their higher expressivity, better flexibility to changes, possible reasoning over the ontology to acquire additional information, shareability between different groups working in the same domain.

When concerning ontologies we mean an ontology represented in OWL, the Web Ontology Language (`www.w3.org/TR/owl-features/`) that is designed for use by applications that need to process the content of information instead of just presenting information to humans. Employing OWL for applications for the Semantic Web has increasing tendency mainly due its greater machine interpretability of Web content than that supported by XML, RDF, and RDF-S by providing additional vocabulary along with a formal semantics.

In applications developed using object-oriented approach, we are working with objects holding data to be processed. When persistent storage of the data of these objects is required, we need to perform the persistent operations over objects, commonly known as CRUD pattern (Create-Retrieve-Update-Delete). The content-based web applications require an extension of the CRUD pattern by the search operation to SCRUD.

The Semantic Web is still in its early stages. Current applications for the Semantic Web are developed in traditional way, enriched by a semantic layer. This layer includes the domain ontologies, inference engines and tools supporting the access and processing of the semantically rich data or services. The Semantic Web brings new challenges in the field of (S)CRUD operations. The situation with (S)CRUD is a little bit different comparing with object-oriented approach, because here we work with ontologies instead of relational databases. In the case that application is developed in object-oriented way, we need to realize a transformation between objects and ontological instances.

In this paper we deal with the (S)CRUD pattern in the context applications for the Semantic Web developed using object-oriented paradigm. Our approach is based on automatic object ontology mapping and (semi)automatic form generation (forms are used to bring the (S)CRUD operations to the user interface level). In the paper we use examples from two domains – online labor market and scientific publication in which our approach was tested and evaluated.

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 that can be considered as a motivation and inspiration for our work. OntoPortal uses ontologies and adaptive hypermedia principles to enrich the linking between resources [11]. The AKT project aims to develop technologies for knowledge processing, management and publishing, e.g. OntoWeaver-S [12], which is a comprehensive ontology-based infrastructure for building knowledge portals with support for web services. The SEAL [14] 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 [15] 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.

However, while various support for creation of adaptive web-based portal solutions is currently 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 considering two major goals: to be able to support different target domains in a single portal instance, and to set up a platform environment where the ontology models and adaptivity will be among first-class features [2]. It is used as an integration and presentation platform for software tools that realize various methods for information acquisition, analyzing, categorizing and presenting. For example, a set of web crawling tools collects (structured) domain-specific documents from the Web. To support information retrieval, approaches like clustering and criteria and top-k ordering are employed. The data and search results presentation is performed by personalized faceted navigation and cluster visualization that employ user characteristics estimated according user clicks while navigating in the information space.

Application domains where we experiment with proposed methods are job offers [13] and scientific publications [6]. In both domains mentioned framework for portal building is used. *Job Offer Portal* (JOP) offers its users ways of semantic-based navigation through the information space of job offers using several personalized presentation tools, which present job offers stored in ontological base. Employers have the possibility to submit new job offers using a set of forms automatically 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 publications and supports users by personalized navigation within the publications information space.

Fig. 1 depicts an overview of the common architecture of portals developed using our framework (for more details on the framework itself see [2]). At the bottom the corporate memory is placed which stores the domain, user and event ontologies [7]. 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 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 include software tools which are tailored (or just reconfigured) for the given domain.

3 (S)CRUD pattern support

Most software systems are in present developed using object-oriented paradigm. In these systems, the data entities are encapsulated in objects during runtime

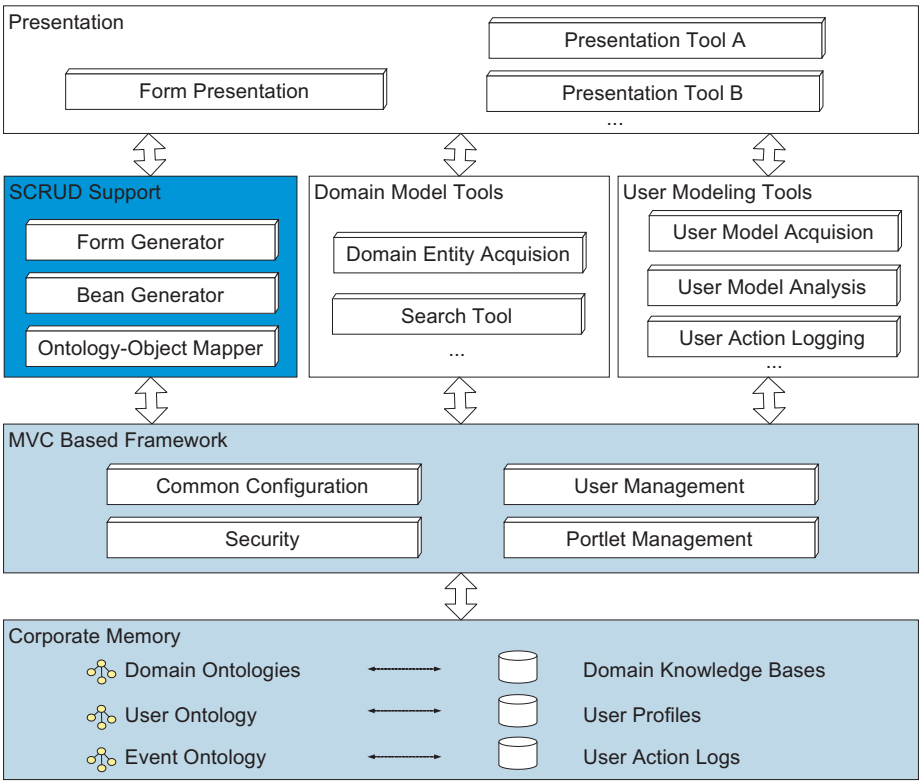

Fig. 1. Portal framework architecture

when they are processed. The persistent store of the data is usually realized employing relational databases. There exist tools supporting the persistent operations in object-oriented applications using relational databases (one of the most known is *Hibernate*, `www.hibernate.org`). In applications for the Semantic Web, which use ontological repositories, there is a lack of such a tool. The persistency is realized using ad hoc ways.

Frequently one of the main tasks in applications for the Semantic Web is searching for data or services, which satisfy posed requirements. Hence, this is a demand of an extension the basic CRUD pattern with the search possibility. Unlike the retrieve operation, which returns an exactly determined instance based on some ID, the search operation should return a set of instances, which suit the constrains. During the search, methods for identifying those instances which match the requirements are used. So the CRUD pattern is extended with the search operation to *SCRUD Search–Create–Retrieve–Update–Delete*.

Our solution supports the (S)CRUD pattern on two levels. First, it offers methods (in the sense of object-oriented paradigm) performing (S)CRUD operations which are called from the application code or by software agents. Second,

since (S)CRUD is also relevant at the user interface level of most applications, we create an access point through which the user can communicate with the application when the operations are performed.

In Fig. 2, the place of the (S)CRUD pattern in the applications for the Semantic Web and our contribution to this field is shown. The beans (simple classes with attributes and corresponding access methods) are generated by the *Bean generator* (right-center in Fig. 2). Instances of these classes are created using the *SCRUD realizer* when (S)CRUD operations are performed (right-bottom in Fig. 2). When

- `create()` method is called, the *SCRUD realizer* takes new instance, transforms the object to RDF graph and stores it in the ontological repository.
- `retrieve()` operation acquires the RDF graph of the given instance and then transforms it into the object and returns it.
- `update()` operation is a composition of `delete()` and `create()` operation. It would by possible to analyze which properties were changed and alter only these, but this is time consuming, hence it is more effective to go the composition way.
- `delete()` operation removes the instance from the ontological repository based on the given object.
- `search()` method returns a list of instances (including relevances), which are related with the string containing key words, given as a input.

These methods can be called from the application to perform the persistent operations. Also a wrapper which acquires entities from web sites can use them to store the data. Even a software agents performing an information processing task can exploit the SCRUD realizer to access the data entities.

To bring the (S)CRUD operations on the user interface level, we have developed a *Form generator* (left-center in Fig. 2). It is used to create (based on ontological description of the entities) the description of forms, which are used to present the data entities for a user who can edit them. Accordingly the user can fill in information of the real world entities and store them. This invokes the binding of the filled data into objects and the call of the `create()` operation of the *SCRUD realizer*. The user can also update the already existing data entities. Then, the `retrieve()` and `update()` operations are exploited. The `search()` operation is called when the user fills keywords in the relevant text field and presses the search button. The found instances are presented for the user employing faceted semantic browser (configured for the domain). User can then choose any instance and edit it in the forms.

3.1 Object-ontology mapping

Mapping of an ontology into object-oriented model is technique, where we create a model of the ontology by taking advantage of the object-oriented paradigm. The basic idea of the ontology to objects mapping is to create a set of classes in such a way that each ontological class has its equivalent in a class of the selected

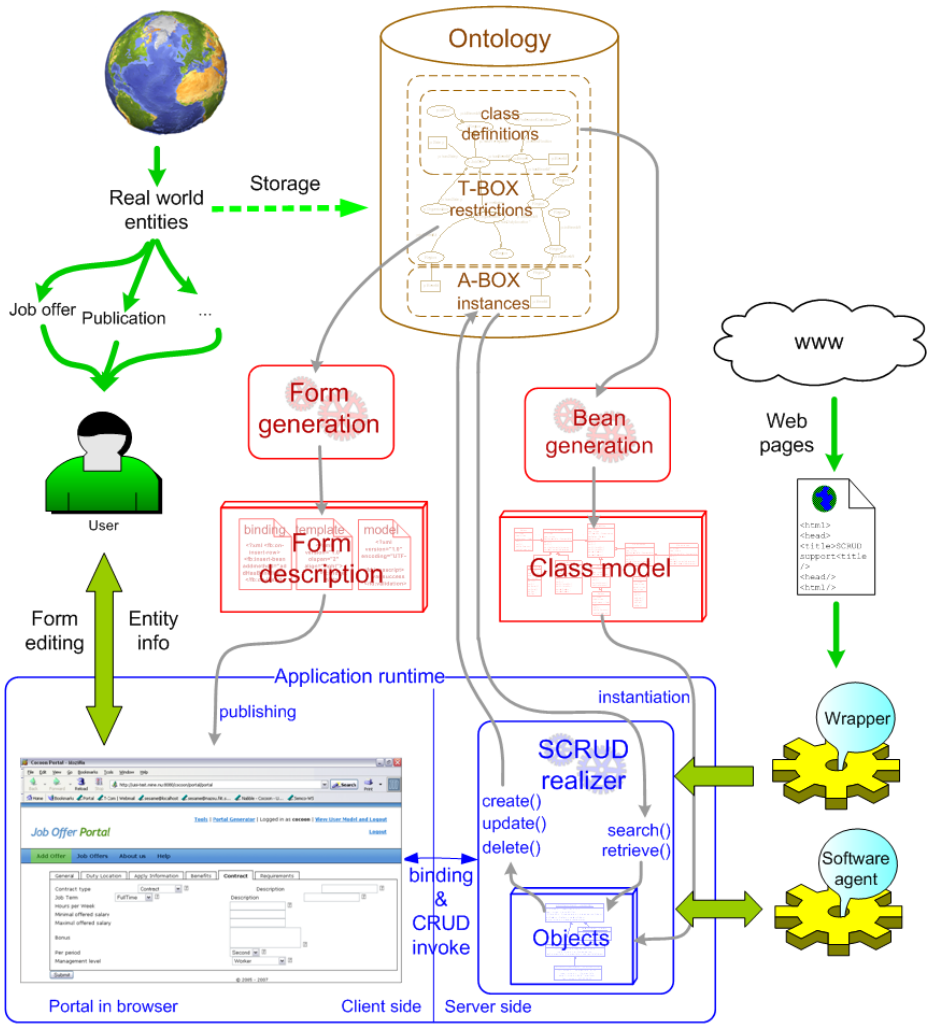

Fig. 2. (S)CRUD pattern support.

object-oriented programming language [10] (the use of interfaces can be necessary when the language does not support multiple inheritance). The creation of such classes can be performed automatically, i.e. the interfaces and classes can be generated. Classes can aggregate other classes which corresponds to object-type properties of ontological classes. Data-type properties of an ontological class have their corresponding attributes in a programmed class. An instance of the ontological class is then mapped to an instance of the corresponding programmed class (the object of that class).

An object-ontology mapper is then similar to an object-relational mapper [1]. It performs the transformation of the instances of the ontological classes into objects and vice versa, i.e. copies the values of the corresponding propertiesMore details about our object-ontology mapping approach is presented in [4].

3.2 Form generation process

Our method for automatic form building from an ontology is based on the structure of the domain ontology (entity description) and form specific information (FSI) stored in the FSI ontology [3]. We identified ontological patterns as repeating structures (sets of concepts and their relations) in ontological representations and defined consecutive data and processing around these patterns [3]. Our method is based on the assumption that similar patterns are shared between ontologies with the patterns themselves being defined using different types of ontological concepts – classes, relations between them, properties, instances and restrictions.

Definition of the form layout. Ontological patterns contain enough information to create a model of the form. However, they do not contain visual description including information about a layout of the form. Therefore we introduced an extension of the domain model – *FormSpecificInformation*. It allows to create user friendly forms and bring configuration possibilities. This extension binds a specific property of selected class with its visual description (see Fig. 3).

We define whether the property will be displayed on the form or not. We allow a user to choose from existing instances related to the property or force her to create a new one. For better usability, we define a pre-defined value of the property. The FSI model allows for dividing of a complex entity form into multiple tabs. Each *FormSpecificInformation* occurs in a list, which defines its order in a form tab. Similarly, we define order of tabs in a list.

This solution allows for personalized layout of forms. The user either edits directly the FSI model (which can be done using automated CRUD pattern support on the FSI entities) or FSIs are adjusted according to user habits (e.g., if the user often uses the same value in the form, the system pre-fills this value automatically). System administrators can also define specialized forms for specific groups of users, hiding irrelevant fields and emphasizing the important ones.

Form generation. The pseudo-code of the form generation is shown in Algorithm 1. The input for the algorithm is a list of ontological entities that are subject of the form generation. If the list has more than one element, also a *Selection Form* is generated. The *Selection Form* gives a user the possibility to select the entity for which the form will be displayed, so he can edit it.

For each entity from the list a corresponding form is generated. First, a blank model of the form is created (this model is an inner representation of the form in the algorithm). Then the list of tabs for the entity is retrieved from the FSI ontology with respect to the chosen template and language (here a

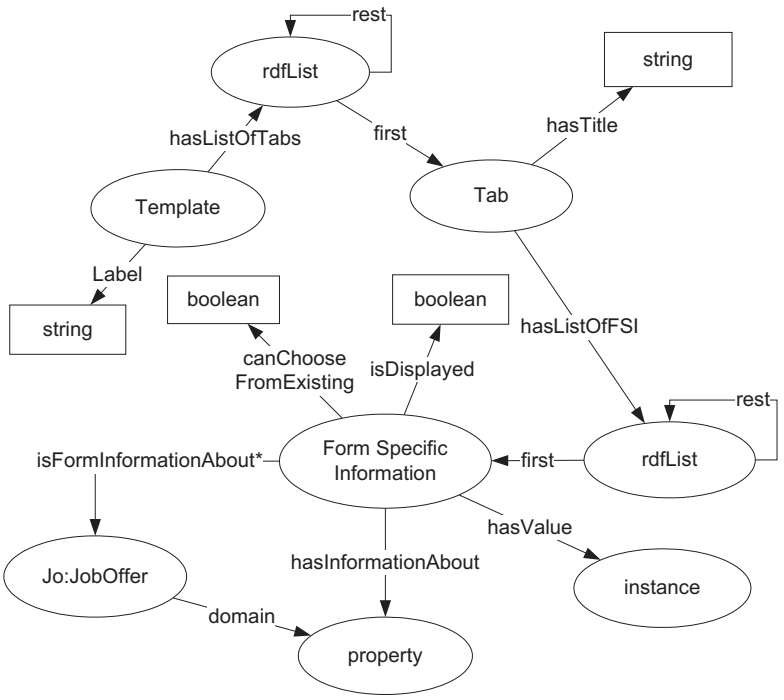

Fig. 3. Domain model extension defining form layout.

user personalization come into consideration). Then, for each tab its content is produced. When a set of elements of the tab is retrieved from the FSI, for each element a pattern is identified and the widget is determined, which is added to the tab content. This way a whole form for the entity is produced based on the specific requirements from the FSI. When the whole content of the tab is created, it is added to the model of the form. After processing each tab, a model is serialized into a required form description. The algorithm returns the list of form descriptions including also the *Selection Form*, if it is required.

4 Loading problem

In this paper we deal with the persistency of data entities in applications for the Semantic Web. Till now, we still have not specified what exactly we mean by term data entities. We correctly think about it like the data of an real world entity in the form which the computer "understands". In our case, when we use ontologies, for example RDFS or OWL notation is used to describe real world entities. In this situation, data entities are represented by a graph in which concepts represent nodes and relations represent edges between them. This graph may be quite complex and its nodes are directly or indirectly interconnected.

Algorithm 1 GenerateForm *Input*: entityList *Output*:formList

```
if entityList.size > 1 then
  selectionForm = createSelectionForm(entityList)
  formList.addForm(selectionForm)
end if
for all entity of entityList do
  model = new form model
  tabList = retrieveTabList(template, entity, language)
  for all tab of tabList do
    elementList = retrieveOrderedListOfElements(tab)
    for all element of elementList do
      pattern = identifyPattern(element)
      widget = determineWidget(pattern)
      tab.addWidget(widget)
    end for
    model.addTab(tab)
  end for
  form = serializeModel(model)
  formList.addForm(form)
end for
return formList
```

More precisely, we can use the RDF graph vocabulary and speak about triples *Subject-Predicate-Object*. The subjects and objects correspond to nodes and predicates to edges. One concrete instance of any ontological class is a subgraph, which consists of the node with instance's URI and the nodes interconnected through edges representing properties (edges included). Some of the properties may be object properties, i.e. they represent also instances of some ontological classes. These instances also may have object properties, which can be considered as subproperties of the primary instance and this can be further applied recursively.

The result is that one ontological instance may be interconnected through properties to each instance of the ontology. When the instance is loaded into an object, it is necessary to know which subproperties should be loaded too. Similar situation occurs in object-relational mapping and is known as loading problem, which may cause difficulties with memory weakness.

4.1 Simple solutions to loading problem

The simplest solution for loading problem is to explicitly define the boundaries of the instance, i.e. for each instance type, we must define which subproperties will or will not be loaded. This solution is usable just in the case when we are able to define such a boundary, i.e., it is known which properties need to be processed in the application and which not.

Hence, this approach is not suitable when it is not known whether some property will or will not be processed. In this case better solution is to use the

lazy load technique. Lazy load object does not load all its attributes immediately, but only when it is needed. In this case it is not needed to specify ahead those properties of instances which will or will not be mapped to objects.

These solutions works fine in object-relational mapping and are usable partially also in object-ontology mapping.

4.2 Loading problem and ontologies

We proposed also approaches to loading problem usable specifically for ontologies. Note that ontologies include elements which may be used to explicitly define the properties of instances needed to be loaded. First, we state that the properties of instances of enumerated classes must not be loaded. An enumerated class is fully defined by a list of its members. Hence, when some instance has a property which range is an enumerated class, it is only needed to know concrete instance from the enumeration – it is not necessary to load its properties.

Second, similar situation appears with properties which range is a class from some taxonomy. Here again, we only need to know which item of the taxonomy is related with the instance.

Third approach is inspired by one of the basic elements of the ontological notations RDFS and OWL – *namespaces*. Namespaces are used to specify the vocabulary used to describe entities. The vocabulary is inherently related to the terms included in the domain for which we create the ontology. This means that when we create ontology for the job offer domain, we use for example the *jobOffer* prefix for each class and property defined in this domain. This is used to solve the loading problem. Only those properties with the prefix *jobOffer* will be loaded, when a job offer instance is retrieved.

Using presented three concepts we help to explicitly define the boundaries of the loading. The last approach we present, is based on an estimation of these boundaries. When loading is performed, there is a strong assumption, that we want to load only properties that are closely related (i.e., similar) to an instance on the semantic level. Since ontologies include this kind of information, we can exploit it. For example, if we are loading a job offer instance, we are not interested in who is the accountant of the company which offers the job.

There has been done work in the field of concept similarity. There exist approaches which are used to determine the semantic distance of concepts [8, 9] and tools realizing the approaches – the Semantic Field Tool[1] (SemFiT). The concept similarity is used to decide whether some property should be loaded comparing the semantic distance of the primary concept (e.g., job offer) and the concept which is the range of the property (e.g., accountant of the company). Since these two concepts have high semantic distance, the property which states that the company's accountant is somebody, will not be loaded. To properly use this concept it is necessary to carefully set the distance, behind which the properties will be considered uninteresting.

[1] Available through a Web Service at `http://khaos.uma.es/SemanticFieldsWS/services/SemFieldsConceptHierarchy?-wsdl`

When dealing with the loading problem in situations where ontological repositories are used the aforementioned approaches are usable to solve it. The idea that the property must not be loaded when its range is an enumerated or taxonomy class, is always applicable. The concept of namespaces is employable regarding how the ontology is constructed. The approach based on semantic distance depends on the used concept similarity method and it requires to determine a threshold distance which decides on loading the property. Proposed approaches can be used separately or in combination. The lazy load technique is suitable if we need to guarantee also the access to the properties behind the supposed boundaries.

5 Conclusions

In this paper we have presented a way of supporting the (S)CRUD pattern support in semantic web applications. Our approach is based on object-ontology mapper and automatic form generator based on the ontology. Our work is a step forward to make the ontologies more usable in new web-based applications. We dealt also with one of the general problem related with retrieval of the data entities from repositories – loading problem that is crucial for realizing search operation in the SCRUD pattern.

Methods presented in this paper are usable generally in applications for the Semantic Web where persistent storage of data entities is required. The object-ontology mapper and form generator were successfully tested on ontologies developed in two research projects, mentioned in section 2, for job offer and scientific publications domains. 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 of publicationss meta-data acquired from the ACM digital library, SpringerLink and CiteSeer.

The form generator was used to create the forms for the job offer and publication entities. Using them and the object ontology mapper the user can fill in new instances in the ontology or edit already existing ones. The mapper was utilized also with the user model ontology. It was used to create a user instance and to store the data acquired through the registration process in the ontology.

Finally, the mapper was used to generate some ontological instances we needed to enrich the ontology with instances of defined qualities. First, an instance was created in the object form based on random generator and then it was stored into the ontological repository using the mapper.

Currently we work on extension of the form generator based on research of the possibilities and usability of integrating personalization into the form generation. The generation process could be influenced by the user model. This way personalized form for the particular user may be created. This includes both layout and content of the form.

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. Accepted.
3. M. Barla, P. Bartalos, P. Sivák, K. Szobi, M. Tvarožek, and R. Filkorn. Ontology as an Information Base for Domain Oriented Portal Solutions. In *15th Int. Conf. on Information Systems Development, ISD'06, Budapest, Hungary*, 2006.
4. P. Bartalos and M. Bieliková. An approach to object-ontology mapping. In *2nd IFIP Central and East European Conference on Software Engineering Techniques CEE-SET 2007*, 2007. Accepted.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, pages 34–43, 2001.
6. 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.
7. 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.
8. Anna Formica. Ontology-based concept similarity in formal concept analysis. *Information Sciences*, 176(18):2624–2641, September 2006.
9. Anna Formica and Michele Missikoff. Concept Similarity in SymOntos: An Enterprise Ontology Management Tool. *The Computer Journal*, 45(6):583–594, 2002.
10. 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.
11. S. Kampa, T. Miles-Board, L. Carr, and W. Hall. Linking with meaning: Ontological hypertext for scholars, 2001.
12. 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.
13. 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.
14. N. Stojanovic, A. Maedche, S. Staab, R. Studer, and Y. Sure. SEAL: a framework for developing SEmantic PortALs, 2001.
15. 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.