Towards Semantic Latvia

Janis BARZDINS, Guntis BARZDINS, Rihards. BALODIS, Karlis CERANS,
Audris KALNINS, Martins OPMANIS, Karlis PODNIEKS
Institute of Mathematics and Computer Science
University of Latvia, Riga LV-1459, Latvia

Abstract. Tim Berners-Lee and co-authors in their seminal paper "The Semantic Web", published in 2001, outlined their vision about the future Semantic Web. But today we are still far from the implementation of this vision. Despite fundamental achievements, like definition of OWL (Web Ontology Language) and rapid progress of RDF/OWL content creation, storage and processing tools, there are still very few attempts to merge these isolated "islands of success" into a killer application, understandable and useful also outside the expert academic community. The primary intent of this paper is to integrate such still isolated results into the unified "Semantic Latvia" conception. The other intent is to propose solutions for the identified missing components in the three fields: 1) technology for gathering of information for the Semantic Web, 2) RDF data stores and efficient access to this information, 3) Semantic Web query tools based on MDA approach.

Keywords. Semantic web, domain ontologies, in-memory RDF data sores, MDA, national information infrastructure

Introduction

Tim Berners-Lee and co-authors in their seminal paper [1] outlined the key principles for the future Semantic Web. Their vision was based on the assumption that information will be distributed globally just like web pages in the current WWW, except this information will be supplemented with the machine-readable semantic tagging. Such machine readable semantic tagging then would allow software agents to automatically perform many information processing tasks, which currently can be handled only manually (like planning a therapy course for Pete's mom in [1]).

But currently the implementation of this vision is still associated with major theoretical and technical difficulties. Despite fundamental achievements, like definition of OWL (Web Ontology Language) and rapid progress of RDF/OWL content creation, storage and processing tools, there are still not many attempts to merge these results into the unified "killer application", which would be understandable and useful also to the end-users outside the "academic/nerdy ghetto" [2] – to those without knowledge of OWL and university grade in ontology engineering.

The primary goal of this paper is to integrate the fragmented Semantic Web achievements into the unified "Semantic Latvia" conception aimed to allow a small country like Latvia already today to take advantage of the emerging Semantic Web technologies. In this paper we are intentionally ignoring the privacy issues involved, as our prime goal is to illustrate the new information system architectures enabled by the Semantic Web.

The other goal of this paper is to identify what is still missing for such unified "Semantic Latvia" conception and to propose potential solutions for filling these gaps. We have identified three gap areas: 1) technology related to information gathering for the Semantic Web, 2) RDF/OWL data stores providing fast access to this information, 3) Semantic Web query tools based on MDA approach.

In the "Semantic Latvia" conception we want to include only those technologies, which are either already implemented, or their possible implementation is fairly clear. These technologies also must fit well into our integrated system. For that reason in the "Semantic Latvia" conception we have omitted many experimental Semantic Web developments, which by our judgment have not yet reached "industrial" grade, like automatic semantic tagging of the natural language documents.

We also want to stress that our "Semantic Latvia" conception is not meant to replace the traditional information systems. Rather, its chief goal is to enable completely new kind of integrated information services – precisely as it was envisioned in [1].

According to the present state-of-the-art, Semantic Web rests on the following five pillars:

1. Ontologies;
2. RDF/OWL data extraction from distributed heterogeneous information sources;
3. Efficient storage and retrieval of RDF/OWL data;
4. Languages and tools for Semantic Web end-users;
5. Reasoning process based on the formal semantics of OWL DL.

In the following sections numbered accordingly, we will mostly elaborate the first four pillars in the context of the proposed "Semantic Latvia" conception. Moreover, we will keep in mind the Tim Berners-Lee words in [1] that all RDF data must be "massaged into shape by the office manager (who never took Comp Sci 101) using off-the-shelf software for writing Semantic Web pages along with resources listed on the … (domain ontologies) site".

1. Ontology Engineering – a Starting Point for "Semantic Latvia"

Ontology is a term borrowed from philosophy. But in the context of Semantic Web it is used in a much more precise sense: "An ontology consists of the various classes and properties that can be used to describe and represent a domain of knowledge. Classes represent concepts within a domain or across domains, and properties represent the relationships among them" [3]. In a sense such ontologies have been used for Information System design already for decades, because a well-designed classic ER (entity relationship) model is essentially the same domain ontology. But until recently these domain ontologies (ER-models) have been considered to be only an internal tool of the system designers, and there was no stimulus for their wider appreciation. But in the case of Semantic Web, the situation changes fundamentally – namely, the development of the domain ontologies becomes the first and foremost step for any Semantic Web application. Moreover, the new requirement for these domain ontologies is that they must be understandable not only by the programmers, but also by the end-users – i.e. they must match the commonly used domain terminology very closely.

According to current understanding, ontologies are the only means for the domain specialists to agree on the common comprehension about the domain. Already in the "pre-ontology era", the daily needs have required to take extra steps for establishing of such "common comprehension" about some essential domains – for example, in Latvia there are laws describing the structure of the most essential national registries, like Citizen register, Enterprise register, Transport register, Land register and others. Among other things, these laws describe the exact items (entities), which shall be stored in each register, and sometimes also relations between these registers. Only a minor step was missing, before the requirements for these registers would have been defined by the means of an ontology.

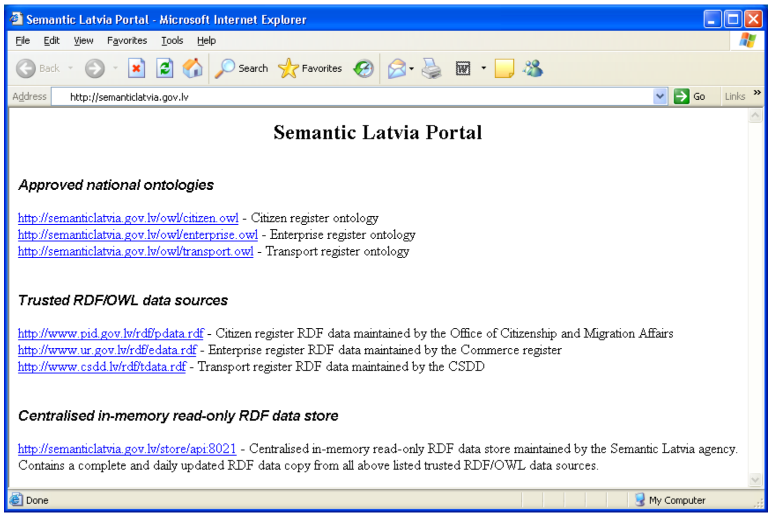
On the way to "Semantic Latvia", our first recommendation to the government of Latvia would be to develop formal ontologies for the main national registers (Citizen register, Enterprise register, Transport register, Land register), as they are forming the core of the concepts essential for the rest of public and business applications. We believe that by such initiative, government would stimulate also private sector to start developing formal ontologies for other areas, like consumer services, health services, transportation, trade, etc. which could eventually all integrate into the joint "Semantic Latvia". The development of precise domain ontologies and their "approval by law" (so that everyone would to stick to them) is the single most fundamental step towards "Semantic Latvia". In our view, the "ontology designer" profession has to become as important as the profession of programmer or lawyer today (who both presently produce complex computer-code or complex contracts/laws for people to obey). Strictly speaking here we are not original – similar national ontology development projects have been started already in USA [4] and Finland [5].

1.1. Ontology Management Infrastructure

The key advantage of conforming to W3C Semantic Web standards and particularly to Web Ontology Language (OWL) [6] (see also RDF [7]) is the eventual opportunity to integrate multiple ontologies and their namespaces into the "global Semantic Web", as well as the possibility to apply an ever growing arsenal of powerful tools being developed for handling of OWL ontologies. For OWL there have been defined three subsequent sublanguages: OWL Full, OWL DL and OWL Lite with decreasing expressivity. For OWL DL and OWL Lite the strict semantics rooted in Description Logic is defined and implemented in the form of powerful automated reasoners ("OWL DL ontology debuggers"), such as RacerPro, Fact++ and Pellet [8]. The "Semantic Latvia" ontologies preferably must be defined within OWL Lite; OWL DL should be used with care due to increased debugging and reasoning complexity. OWL Full shall not be used at all, as its semantics is not formalized.

Besides development of the ontologies themselves, on the national level must be established also the ontology management infrastructure – a national ontology portal providing a reliable access to the approved and current versions of the national ontologies (Fig.1.)

Unlike in the ad-hoc ontology portals [17], the national ontology portal must also standardize the namespaces used by the ontologies and ensure that only nationally approved namespaces are used by the nationally approved ontologies. Our proposed solution to the namespace standardization issue is following: a) establish a well-known domain name for the national ontology portal (e.g. http://semanticlatvia.gov.lv) serving also as the root for the namespaces of all approved "Semantic Latvia" ontologies; b)


Semantic Latvia Portal - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
Address
http://semanticlatvia.gov.lv
Go
Links
Done
My Computer

Semantic Latvia Portal

Approved national ontologies

http://semanticlatvia.gov.lv/owl/citizen.owl - Citizen register ontology
http://semanticlatvia.gov.lv/owl/enterprise.owl - Enterprise register ontology
http://semanticlatvia.gov.lv/owl/transport.owl - Transport register ontology

Trusted RDF/OWL data sources

http://www.pid.gov.lv/rdf/pdata.rdf - Citizen register RDF data maintained by the Office of Citizenship and Migration Affairs
http://www.ur.gov.lv/rdf/edata.rdf - Enterprise register RDF data maintained by the Commerce register
http://www.csdd.lv/rdf/tdata.rdf - Transport register RDF data maintained by the CSDD

Centralised in-memory read-only RDF data store

http://semanticlatvia.gov.lv/store/api:8021 - Centralised in-memory read-only RDF data store maintained by the Semantic Latvia agency. Contains a complete and daily updated RDF data copy from all above listed trusted RDF/OWL data sources.

Figure 1. The national approved ontology portal along with the list of trusted RDF data sources (this web page and addresses are simulated).

additionally certify essential international namespace roots, such as W3C namespace http://www.w3.org, which may also be used by the approved national ontologies; c) all national resource URIs used by the national ontologies must have the standard format "http://semanticlatvia.gov.lv/ont/ontologyname.owl#localname", where "ontologyname.owl" is one of the approved national ontologies stored on the ontology portal and containing the definition of the mentioned resource "localname" (class or property), including its natural language definition under the pre-defined "label" property. For example, if the resource under consideration is concept "boat" (localname), which is defined in the approved ontology "transport.owl" (ontologyname.owl), stored at URL "http://semanticlatvia.gov.lv/ont/transport.owl", then the "transport.owl" ontology must contain at least the following information:

```
<owl:Class rdf:ID=
     "http://semanticlatvia.gov.lv/ont/transport.owl#boat">
     <rdfs:comment> "an open vehicle for traveling on water"
</rdfs:comment> </owl:Class>
```

Finally, besides approved ontologies and namespaces, the Semantic Latvia ontology portal also must contain the list of trusted servers, where RDF/OWL data (class instances of approved ontologies) can be found. Such list will typically include the web servers of national registers, such as Population register, Enterprise register, Transport register etc. It is assumed (theoretically) that all these registers regularly post all their contents in the RDF/OWL data format according to the approved ontologies on their web server, so that interested parties can retrieve it. In practice this step would need to be optimized in a number of ways – besides more advanced security, it would be also more practical to store all this RDF/OWL data in the centralized "national" read-only in-memory data store (discussed in the section 3), and only incremental changes from various registers would need to be fed into such centralized read-only RDF/OWL store.

To set the precedent, one of the first steps could be creating of such ontology portal infrastructure for the "Semantic University".

2. Extraction of Information according to Fixed Ontologies

There is a massive amount of tools [12,13,14] and literature [10,11] about manual, semi-automatic or fully-automatic extraction of RDF data (RDF triples according to public domain ontologies) from heterogeneous, distributed data sources, such as HTML pages, legacy documents, news articles, etc. If the data source, from which RDF/OWL data needs to be extracted, has been created without knowledge of the target ontology, then such extraction is very difficult and complex task. It is particularly complex, if the data source is a natural language text. In our view these technologies currently are too immature for infrastructural use – despite enthusiasm of some early adaptors [14], this is still the key stumbling block for the "canonical" Semantic Web, envisioned as a mere extension (annotation) of the traditional web. Our proposal for "Semantic Latvia" is different and is based on the following two ideas.

The first idea is borrowed from Google, which effectively crawls and copies the entire global web content to its own distributed and indexed data store to ensure fast access required for processing complex multi-word queries [18]. In case of Semantic Web content, fast RDF/OWL data retrieval is even more crucial due to higher complexity of the typical Semantic Web inquiries or automatic reasoning tasks. To deal with this problem, fast in-memory RDF data stores will be discussed in the following sections.

The second idea is that domain ontologies must be approved and made publicly available before the domain information systems, including domain-specific textual web content, are created (according to these approved ontologies and their proper namespaces). In this case RDF/OWL data extraction from the domain information systems and domain-specific textual web content becomes a much simpler task. In the ideal case, the information system designers themselves should be able to implement the RDF/OWL data export according to the approved domain ontologies, so we will not elaborate this further. Handling of domain-specific textual web content according to the approved domain ontologies is slightly trickier and is discussed below.

To our surprise, presently there is very limited research [9] and tool support for authoring of domain-specific text documents (web pages, other document formats) with RDF/OWL data embedded (or linked) according to pre-defined domain ontologies. Curriculum Vitae, List of Publications, Medical examination results, Office opening hours, Product catalogues, etc. are examples of text documents (web pages), which could easily be generated semi-automatically from pre-defined OWL ontologies via simple ontology-driven form-based data input interface. Adobe XMP (eXtensible Metadata Platform) [15] for embedding RDF/OWL data into PDF documents and other media files is one of the very few industrial developments in this direction.

We will illustrate our proposal by the example of creating a web page containing a List of Publications. Of course, we can create such web page directly in HTML without any tools or ontologies (as most of us still do). But in such case extracting the RDF/OWL data from such List of Publications would be a difficult task (addressed by so called "scrappers"), especially in the light of punctuation variations used by various authors. According to our "Semantic Latvia" vision, the List of Publications web page

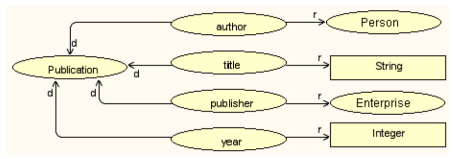
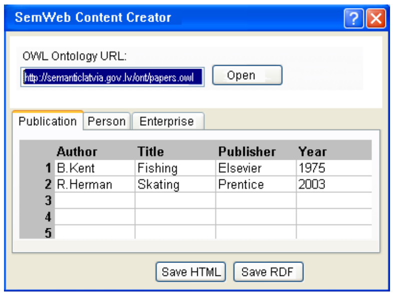


Figure 2. Graphic representation of the "papers.owl" ontology



OWL Ontology URL: http://semanticlatvia.gov.lv/ont/papers.owl

	Author	Title	Publisher	Year
1	B.Kent	Fishing	Elsevier	1975
2	R.Herman	Skating	Prentice	2003
3				
4				
5				

Figure 3. Hypothetical application for creating domain-specific web pages and corresponding RDF-data according to the given domain ontology

could have been created by a simple universal application shown in Fig.3 in following 3 steps:

1. Go to the "Semantic Latvia" web portal and find an approved ontology for lists of publications, e.g. http://SemanticLatvia.gov.lv/ont/papers.owl. Such example ontology is depicted graphically in Fig.2.
2. By loading this ontology into the application shown in Fig.3, the application automatically tunes itself and displays the data input form with the fields and options permitted by the selected ontology. User enters data into the relevant input form fields; application might prompt the already entered Person or Enterprise names (with URI) for the Author and Publisher fields
3. When all data is entered, use buttons "Save HTML" to generate the HTML version "mylist.html" of the list of publications (formatting style-sheet might be applied for nicer layout), and "Save RDF" to generate the RDF version "mylist.rdf" containing the same information in machine-readable format.

Both files shall be placed on the author's web server – the "mylist.html" file will be viewed by humans, while "mylist.rdf" file will be used by Semantic Web applications, such as Swoogle [16] (imitating Google by crawling and collecting .rdf files on the web) or those described in the following sections. Note that W3C has not defined a standard for linking the two files "mylist.html" and "mylist.rdf" together, which sometimes causes confusion and hinders reliable navigation between the human-readable and machine-readable formats. Nevertheless, following syntax variations are commonly used to provide a link from the HTML file to its corresponding RDF data file:

```
<head>
<title>My Document</title>
<meta name="OWL" content="author.rdf">
<link rel="meta" type="application/rdf+xml" href="author.rdf"/>
<link rel="alternate" type="application/owl+xml" title="OWL" href="author.rdf" />
<link rel="alternate" type="application/rdf+xml" title="RDF" href="author.rdf" />
</head>
```

The proposed 3-step process for creating machine-readable Semantic Web content, in our view, is simple enough to be handled by "a manager, who never took Comp Sci 101", as was envisioned in [1].

Strictly speaking, the proposed 3-step process is not entirely original – a similar approach is described also in [9], where additional means for input-form style-sheet control in medical domain are discussed. We will return to this subject in the section 4, where MDA and model transformations will be used to facilitate interaction with the Semantic Web RDF/OWL data.

3. RDF/OWL Data Stores

Once the RDF data is extracted, the next crucial issue is how to store it for efficient retrieval by agents, reasoners, or other applications. Awareness about significance of this issue is growing – from one related paper in the 3rd International Semantic Web Conference (ISWC 2004) to already four related papers [19,20,21,22] in the 4th International Semantic Web Conference (ISWC 2005). Various RDF data storage architectures are being proposed.

Storing of RDF data in a centralized relational database is studied in [21], where authors have tested and compared performance of 5 different relational database representations of RDF data: schema-aware (with explicit or implicit storage of subsumption relationships), schema-oblivious (with or without identifiers to represent resources) and the hybrid of both. Their conclusions were drawn from the experiments with the taxonomic queries: a) the hybrid representation is the most efficient, b) schema-aware representations exhibit better overall performance than the schema-oblivious ones, c) the schema-oblivious representation with identifiers exhibits the worst overall performance.

Meanwhile for more complex Semantic Web tasks, such as semantic association discovery, according to [20], feasible performance can be achieved only by: a) storing all RDF data in the main memory; b) query programming through the low-level API „suitable to operate directly on the internal graph representation structures". Consequently, authors of [20] have developed a specialized in-memory RDF data store BRAHMS and have demonstrated its superiority compared to 3 other in-memory RDF data storage systems.

In reality, it is hard to compare different RDF data stores without bias, because they use dissimilar API, optimized for different types of tasks. Currently there are no any standards for the RDF data store low-level API (note that traditional RDF query languages like SPARQL are too high-level and thus inefficient). Our general conclusion is that the high-performance in-memory RFD data store issue is not yet adequately resolved.

In the next section we will describe our own in-memory RDF data store, code named "OUR" for the rest of the paper. This data-store is adequate for the core registers of a small country like Latvia. For example, Citizen and Enterprise registers are among the largest ones, but still contain only about 4 GBytes of raw information. At the same time the 64bit computer architecture today allows to have and efficiently use tens of GBytes of the main memory. This means that in-memory data stores are completely applicable already today, especially for optimizing read-only information retrieval tasks, where potential in-memory data loss upon sudden equipment failure is not an issue. Additionally, it shall be noted that in-memory it is necessary to store only the parts of information, which are structured and therefore meaningfully "searchable" – the rest of information, like photos, plans of buildings, copies of documents and like can be stored externally and referenced to by URLs or other means. Such distinction could be coded already in the ontology itself by adding property "unstructured" to classes representing such unstructured entities.

3.1. OUR Approach – Metamodel-Based In-memory Data Store

For RDF data storage and efficient retrieval we propose to use metamodel-based in-memory data store. Such stores allow RDF data to be stored internally according to an arbitrary user-defined metamodel (domain ontology). Such flexibility gives option to tune the data store to the specific domain ontology for optimal storage and retrieval of corresponding RDF data. Alternatively, the data store can be tuned to the more generic RDF or OWL metamodel (described in section 5), in which case it can store arbitrary RDF triples or arbitrary OWL ontologies, at the expense of slightly lower performance. These alternatives correspond to the schema-aware representations and schema-oblivious representations mentioned in [21]. The schema-aware representation has at least two advantages: a) higher performance, because the advance knowledge of the data structure considerably reduces the search-space; b) more natural queries with fewer parameters, formulated in the terms of the domain ontology.

Selection of the data store API is not easy – it must include only functions having efficient implementations, and at the same time these functions must closely cover typical Semantic Web tasks.

API of our data store is implemented as a function library. This library offers: a) a system of low-level data retrieval functions that is complete for low-level data query programming (as required for Semantic Web data stores in [20]); b) a selected set of more complicated widely usable data searching functions. By means of a sophisticated indexing mechanism, also these more complicated functions are efficiently implemented.

Our API includes three groups of functions:

1. Meta-model management - about 40 functions for creating, modifying, deleting of classes, attributes and associations, querying about their properties, class inheritance etc.:

- *CreateClass (class_name): class_id;* Creates class and returns class identifier.
- *CreateAttribute (class_id, attribute_name, base_type): attribute id;* Creates a class attribute, returns attribute identifier (base types: boolean, integer, string etc.)
- *CreateAssociation (association_name, inverse_association_name, start_class_id, end_class_id, start_multiplicity, end_multiplicity): association_id;* Creates association and the corresponding inverse association (as types) between two classes, returns association identifier.
- *ConnectSubclass (subclass_id, superclass_id);* Supports multiple inheritance.
- *GetClassIdByName (class_name): class_id;*
- *GetAttributeIdByName (class_id, attribute_name): attribute_id;*
- *GetAssociationIdByName (start_class_id, association_name): association_id;*
- …

2. Instance management - about 30 functions for creating instances, assigning attribute values, creating associations between instances, modifying and deleting, querying about instance attributes and associations, etc.:

- *CreateInstance (class_id): instance_id;* Creates a class instance, returns identifier.

- *AddAttributeValue (instance_id, attribute_id, attribute_value);* Assigns an attribute value to an instance.
- *AddAssociation (start_instance_id, association_id, end_instance_id);* Links two instances.
- *GetInstanceCount (class_id): integer;* Returns class instance count.
- GetInstance (class_id, index): instance_id; Returns identifier of i-th class instance.
- *GetAttributeValue (instance_id, attribute_id): attribute_value;* Returns attribute value.
- *GetAssociationCount (instance_id, association_id): integer;* Returns count of instances connected via association_id to instance_id.
- *GetAssociationPartner (instance_id, association_id, index): instance_id;* Returns identifier of the i-th connected instance.
- …

3. Search functions are implemented as iterators. The search process starts with specification of its scope:

- *CreateIterator (parameter_list): iterator_id;* Creates an iterator, returns iterator identifier. The search scope is specified by the parameter list (see examples below).

The following function iteratively extracts the next portion of the required instances:

- *GetNextInstances (iterator_id, instance_count): instance_id_list;* Returns identifier list of the required number of instances. This kind of flexibility may be necessary for „visiting" web-agents.

At the end, the search process must be stopped:

- *DeleteIterator (iterator_id);* Releases resources used for the iteration process.

The following search processes are efficiently implemented and included in our API:

- *CreateIterator (class_id);* Initiates scanning of all instances of a given class.
- *CreateIterator (instance_id, association_id, target_class_id);* Initiates scanning of all instances associated with a given instance via given association.
- *CreateIterator (instance_id, association_id1, ..., association_idn, target_class_id);* Initiates scanning of all instances associated with a given instance via given chain of connected associations. Length of association chain is not limited.
- *CreateIterator (instance_id1, association_id11, ..., association_id1m, instance_id2, association_id21, ..., association_id2n, ..., target_class_id);* Initiates scanning of all instances associated with several given instances via given chains of connected association (conjunction). Length of association chains and number of conjunction members is not limited.
- *CreateIterator (class_id, attribute_id, attribute_value);* Initiates scanning of all instances of a given class having a given attribute value.
- *CreateIterator (class_id, attribute_id1, attribute_value1, ..., attribute_idn, attribute_value_n);* Initiates scanning of all instances of a given type having several given attribute values (conjunction). Number of attribute values is not limited.

These search processes form the basis on which more complex queries can be constructed. Web-agent support for searching in distributed in-memory data stores is also under development.

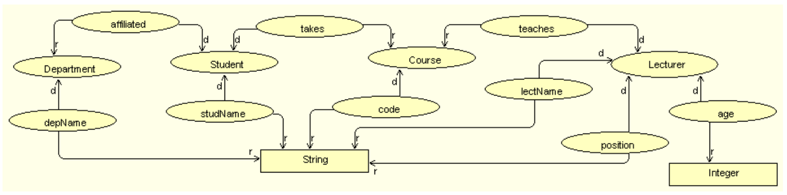


Figure 4. University Ontology as an OWL graph (simplified)

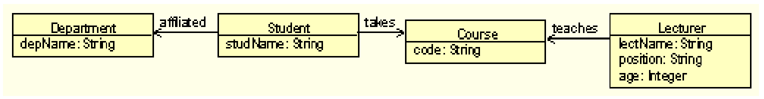


Figure 5. University Ontology as a UML class diagram

The described metamodel-based in-memory data store has been developed over many years as part of high-performance graphic modeling tools Exigen Business Modeler (EBM) [24] and GRADE [25,26]. The key requirement of graphical modeling tools was fast retrieval of data necessary for displaying various kinds of tree-like views. The above mentioned search processes were heavily optimized to support this requirement. In case of RDF, the very same search functions can be efficiently used for graph-like queries such as adjacency (retrieving 1-neighborhood or k-neighborhood), connectedness and pattern matching.

In what follows we compare the performance of or in-memory data store with that of the Sesame tool [23] for RDF data storing and querying, version 1.2.3. Note that the data store BRAHMS that has been reported to have the best query times in [20] has not yet been made available at the time of this writing. Sesame has come out the second best according to [20].

The experiment performed was a relatively simple, yet not too simple query: "for all instances of a given class X, look at all related instances in class Y and calculate the sum of attribute values A of those Y instances found, which are further related to an instance in class Z that satisfies a property P." The experiment was performed on the data stores containing 20 thousand instances of class X, each related with 100 instances of Y, 2 million instances of Y altogether, on a computer with Intel 3.2GHz dual core processor and 2GB memory. The times for calculating the requested sum was as follows:

Sesame, with single query	*6546 msec*
Sesame, access through API	*3875 msec*
OUR in-memory data store	*1109 msec*

As it is possible to observe, on this example OUR data store gives the search speed improvement about 3.5 times. The experiment also confirms that using a low level API in performing search tasks is more efficient than using high-level queries. These are only preliminary encouraging results and more detailed comparison is still necessary.

3.2. RDF/OWL Data Storing Options in OUR In-memory Data Store

As mentioned, our data store can store RDF/OWL data in two different ways:

- according to the given ontology (schema-aware way)
- according to the OWL metamodel (schema-oblivious way).

Now let us go into more details. Let us assume that we have a (very simplified) University Ontology presented in Fig. 4. This figure presents the ontology as an OWL graph (d denotes domain and r denotes range). Fig. 5 presents the same ontology as UML class diagram.

This class diagram can be treated as a domain metamodel and be used to configure OUR in-memory data store in the schema-aware mode. In this case the data store will keep the data according to this metamodel and its API can be used according to the metamodel (e.g., a following function invocation CreateInstance (student_id) , where student_id is the identifier for the class Student, will be valid).

However, on the basis of ontology for one specific domain it is difficult to define universal tools, which would be usable for any ontology (see the next section). Therefore in the general case it is better to store the data according to a universal metamodel, where any ontology can be embedded. Namely, the OWL metamodel itself serves this purpose. OMG has published the Request for proposals (RFP) for the Ontology Definition Metamodel in 2003. Currently the OMG candidate for Ontology Definition Metamodel is available [3]. An interesting independent OWL metamodel is given in [27]. For our goals it is very important to select such OWL metamodel, where an instance of this metamodel corresponding to a given ontology would be visually as close as possible to the graph of the ontology itself. Fig. 6 shows our proposed metamodel for OWL Lite (in this paper we limit ourselves to OWL Lite, and without Restrictions and Containers). We use [3] as the basis for this metamodel, only the metamodel part describing property instances is modified according to [27].

The ontology in Fig.4 can now be represented as an instance of this metamodel, Fig. 7 shows this form. Due to the adequate choice of metamodel, Fig. 4. and 7. are quite similar.

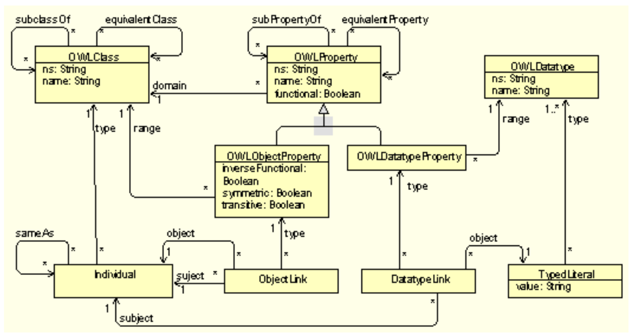


Figure 6. OWL Lite metamodel

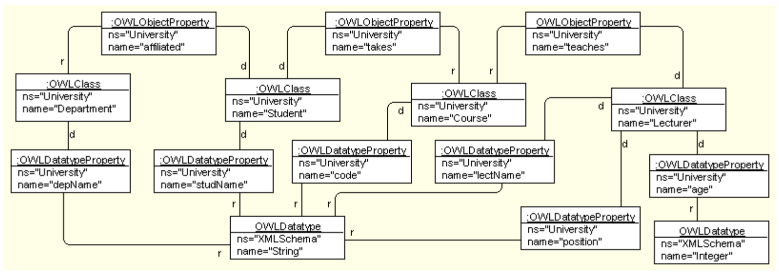


Figure 7. University Ontology as an instance of OWL metamodel

Now OUR in-memory data store can be configured according to the accepted OWL metamodel. In this case the data store will keep OWL data according this metamodel, in a uniform way for any domain ontology. This will ensure a very flexible usage of this store. However, in this case more class, attribute and association instances are required to represent the same data. Therefore we cannot achieve the same performance using the universal metamodel as that when the data store is configured to a specific domain ontology. However, due to the appropriate choice of API for OUR data store, this slowdown is not larger than 6-fold.

4. Languages and Tools for Semantic Web Endusers

One of the most important problems having no satisfactory solution in the area of Semantic Web is an easy usable query language for end users. This is due to the fact that in the area of Semantic Web the types of queries cannot be standardized beforehand, as it is possible in traditional information systems. For example, in the classical Berners-Lee example [1] the way Lucy instructs her Semantic Web agent is left open. One of the more or less popular ideas is to use Structured English to formulate queries [28], but it is very far from a solution satisfactory in practice along this direction.

Apparently, the most natural way how to solve this problem is to build special (domain specific) languages, and, in our opinion, preference should be given to graphical languages which could be understood by the end user without special training.

Just to give a feeling how such end-user query language could look like, we briefly sketch an example of a graphical query language, named DEMO. Fig. 8 shows a sketch of window contents of a would-be query tool supporting this language. This diagram window shows OWL classes and properties of the University Ontology (defined in section 3) in the form of a graph (a simple class diagram). The user can select some constraint classes, e.g., Department, Lecturer, … and specify which instances of these classes are of interest. For example, for Department these instances of interest are CmpSC and Math. For properties with integer values the corresponding bounds can be specified. Then the user can select a query class, e.g., Student and specify the How many option (another alternative would be List all). In the result the tool will find how many instances of Student satisfy the query conditions. The query presented in Fig. 8

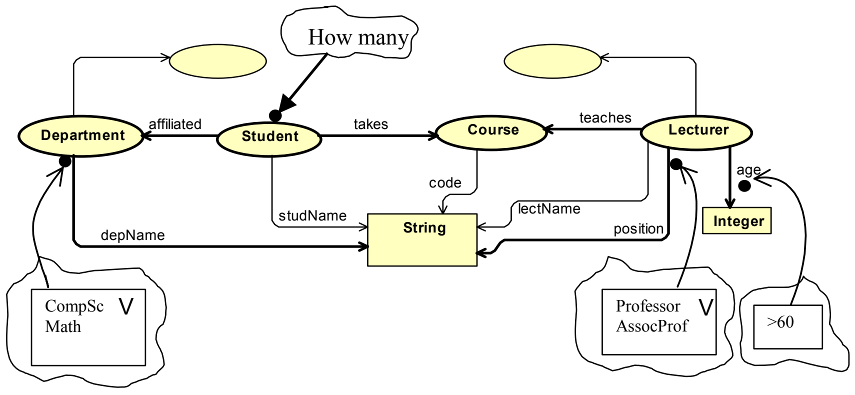


Figure 8. Window contents of the DEMO tool showing the University Ontology

informally would read this way: "How many students there are in CompSc or Math departments, for whom some courses are taught by Professors over sixty?"

The tool supporting DEMO has to build a diagram like the one in Fig. 8 from the corresponding ontology definition. The challenge is how to implement such a tool with minimum effort - due to the fact that functional requirements for such a tool would be quite unstable and additional wishes likely would spring up during the use.

Certainly, such a tool can be implemented in any standard OOPL, e.g., C++, using the Repository API, but such an implementation would be very expensive, especially the support of diagram graphics. In the area of modeling tool building a new idea has appeared, namely, generic metamodel based modeling tools [29,30]. A certain contribution to the development of this idea has been made also by the authors of this paper [31,32]. Currently the authors of this paper are developing a much more innovative approach, namely on a Tool Framework based on model transformations and their efficient implementation (a similar approach has been recently proposed also in [33]). Use of model transformations in a very flexible way is the backbone of this new framework. On the way to this framework the authors have developed a model transformation language MOLA [34-39], which is well suited for tasks arising there (as it is well known, model transformation languages form the core of the MDA approach, see, e.g., [40,41]). Below the idea of Tool Framework will be briefly sketched on the basis of a DEMO tool.

The basic idea of our framework relies on two kinds of metamodels. One of them is the domain metamodel and other the presentation metamodel. In our DEMO tool the OWL metamodel (shown already in Fig. 6) will serve as the domain metamodel. Now let us look at some details of the presentation metamodel. This metamodel defines the type of visual presentation used in a window, this time a graphical one. For the DEMO tool and many similar simple diagrams the directed graph is a very adequate presentation metamodel. Certainly, both nodes and edges can contain text Compartments. In addition, the presentation metamodel contains also Events – the possible user actions on visual elements. Fig. 9. shows both the domain metamodel (yellow classes) and the presentation metamodel (light green classes). The DEMO tool window example (Fig. 8) actually is an instance of this metamodel (with events not shown for the sake of simplicity).

The next essential component of our Tool Framework is a presentation engine library, one for each presentation metamodel. The presentation engine is a program which visualizes the instances of the given metamodel and reacts to user actions specified in the metamodel. In our example the engine for visualizing a directed graph is used and we assume it to be sophisticated enough to generate automatically a readable graph layout. The reaction on an event, such as rightclick on a node, is to set the appropriate attribute (e.g., selected) of the node to true.

Now we can return to the structure of our DEMO tool and show how it relies on the Tool Framework. The first task the tool has to do is to find in an OWL model all classes and object properties and to present in the form similar to Fig. 8. This is done in two steps. At first the relevant information is extracted from the OWL model and then stored according to the presentation metamodel. The simplest way to do this task is in a model transformation language.

Then the presentation engine for directed graphs is invoked, which actually displays the nodes and edges with text compartments in a graph window and starts to listen to user actions. When user selects a class node for the query condition, the engine stores the selection in the node and invokes another model transformation program,

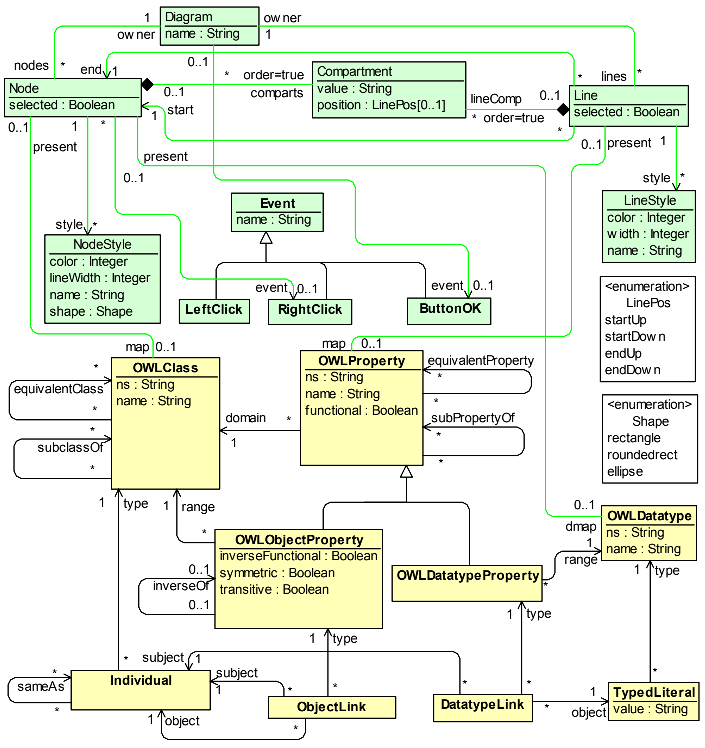


Figure 9. The extended OWL metamodel

which transfers the selection to the domain (OWL) model. The query result node is processed similarly. Finally, when user presses the OK button, the transformation is invoked, which evaluates the query and presents the result (via the presentation engine for simple dialogs).

Our current experience shows that a tool like DEMO in this way can be built with 10 times less effort than required for implementation directly in C++. Certainly, this speedup is under the condition that the presentation engine library for most used presentation metamodels is pre-built. This library is universal – it can be used for any tool within the Tool Framework, and it has to be built only once. Currently such a library is under development.

5. Conclusion

Following is the summary of the proposed Semantic Latvia vision:

1. It is necessary to develop and approve formal ontologies for the domains, which will join the Semantic Latvia. (Most of the national registers are very close to that, as their structure is already described and approved by the law.)
2. It is necessary to create the national approved ontologies portal, which should also list the web servers containing trusted RDF/OWL data corresponding to these ontologies.

3. The existing information systems and registers, which would like to join Semantic Latvia, must define their ontologies and have them approved and included into the national ontology portal. They also must ensure regular export of their data into the RDF/OWL format according to the approved ontology, and place this data on the trusted web server. (Internally such registers may continue to use a different architecture based on the relational database, but we believe that getting their ontology approved will be a good stimulus to eventually migrate to the RDF data-store architecture also internally.)
4. It is possible to publish RDF/OWL data according to approved ontologies also in the format of the regular textual web pages, complemented with their OWL/RDF data pages (as described in the section 2). For such textually originated RDF/OWL data to be part of Semantic Latvia, it must be published on a trusted web server.
5. Similar to Google, Semantic Latvia agency must regularly collect all RDF/OWL data from the trusted web servers and store in its own ultra-fast in-memory RDF/OWL data store (or stores).
6. Semantic Latvia agency can grant controlled access to the parts of its in-memory RDF/OWL data to the wide range of end-users, based on their access rights. Such access-rights could be encoded already in the domain ontologies themselves via a special “access-rights” property
7. End-users must be equipped with the new generation of Semantic Web browsers, similar to the tool described in the section 4. The purpose of such tool is to enable end-users to enter complex Semantic Web queries in the most intuitive format possible, which we believe, is the illustrated graphic format.
8. In this paper we have discussed only the information retrieval aspect of the Semantic Web. This gives possibility to retrieve information about availability of the complex resources, like a free timeslot in the therapist schedule in the Tim Berners-Lee example. Meanwhile there is a related issue, outside of the described Semantic Latvia vision, about how to automatically reserve the appointment with the found therapist. This would be an interesting issue to explore next.

References

[1] Tim Berners-Lee, James Hendler and Ora Lassila. The Semantic Web. Scientific American, May 2001.

[2] H.Alani et al. Towards a Killer App for the Semantic Web. ISWC 2005, LNCS 3729, pp.829-843, 2005.

[3] IBM, Sandpiper Software. Ontology Definition Metamodel. Third Revised Submission to OMG/RFP, ad/2003-03-40, August 2005. URL: http://www/omg.org/docs/ad/05-08-01.pdf

[4] NCOR (National Center for Ontological Research), URL: http://ncor.us

[5] E. Hyvonen, A. Valo et al. Creating a National Content and Service Infrastructure for the Finnish Semantic Web. Poster & Demonstration Proceedings, ISWC2005, Galway, Ireland, 2005.

[6] Web Ontology Language (OWL). W3C, 2004. URL: http//www.w3.org/2004/OWL/

[7] Resource Description Language (RDF). W3C, 2004. URL: http://www.w3.org/RDF/

[8] Hai Wang, Matthew Horridge, Alan Rector, Nick Drummond, and Julian Seidenberg. Debugging OWL-DL Ontologies: A Heuristic Approach. ISWC 2005, LNCS 3729, pp. 745–757, 2005.

[9] V.Kashyap et.al. Definitions Management: A Semantics-Based Approach for Clinical Documentation in Healthcare Delivery. ISWC 2005, LNCS 3729, pp. 887-901, 2005.

[10] Fabian Abel, Robert Baumgartner, Adrian Brooks, Christian Enzi, Georg Gottlob, Nicola Henze, Marcus Herzog, Matthias Kriesell, Wolfgang Nejdl, Kai Tomaschewski. The Personal Publication Reader. ISWC 2005, LNCS 3729, pp. 1050–1053, 2005.

[11] V.Uren et.al. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. Journal of Web Semantics, Elsevier, Vol 4 (2005), p.14-28.

[12] http://cerebra.com/

[13] http://www.landcglobal.com

[14] David Huynh, Stefano Mazzocchi, and David Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. ISWC 2005, LNCS 3729, pp. 413-430, 2005.

[15] www.adobe.com/products/xmp/pdfs/whitepaper.pdf

[16] http://swoogle.umbc.edu/

[17] http://www.schemaweb.info

[18] http://www.googleguide.com/google_works.html

[19] Raul Garcia-Castro, Asuncion Gomez-Perez. Guidelines for Evaluating the Performance of Ontology Management APIs. ISWC 2005, LNCS 3729, pp.277-292.

[20] Maciej Janik, Krzysztof Kochut. BRAHMS: A workBench RDF store And High performance Memory System for Semantic Association Discovery. ISWC 2005, LNCS 3729, pp.431-445, 2005.

[21] Yannis Theoharis, Vassilis Christophides, Grigoris Karvounarakis. Benchmarking Database Representations of RDF/S Stores. ISWC 2005, LNCS 3729, pp.685-701.

[22] Sui-Yu Wang, Yuanbo Guo, Abir Qasem, Jeff Heflin. Rapid Benchmarking for Semantic Web Knowledge Base Systems. ISWC 2005, LNCS 3729, pp.745-757.

[23] J.Broekstra, A.Kampman, F.v.Harmelan. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. Proc. International Semantic Web Conference, Sardinia, Italy, 2002.

[24] A.Kalnins, K.Podnieks, A.Zarins, E.Celms, J.Barzdins. Editor Definition Language and its Implementation. LNCS 2247, pp.530-537, 2001.

[25] J.Barzdins, I.Etmane, A.Kalnins, K.Podnieks. Towards Integrated Computer Aided Systems and Software Engineering Tool for Information Systems Design. Proc. 2nd International Workshop on Advances in Databases and Information Systems (ADBIS'95), Springer, pp. 3-11, 1996.

[26] J.Barzdins, A.Kalnins, K.Podnieks. MiniGRADE – A Tool for Conceptual Modeling by Class Diagrams. Proc. 18th International Conference on Conceptual Modeling, LNCS 1728, pp. 11-12, 1999.

[27] S.Brockmans, R.Volz, A.Eberhart, P.Loffer. Visual Modeling of OWL DL Ontologies Using UML. LNCS 3298, pp.198-213, 2004.

[28] A.Bernstein, E.Kaufmann, A.Gohring, C.Kiefer. Querying Ontologies: A Controlled English Interface for End-Users. ISWC 2005, LNCS 3729, pp.112-126, 2005. .

[29] A.Ledeczi, M.Maroti, A.Bakay, G.Karsai, J.Garrett, C.Thomason, G.Nordstrom, J.Sprinkle, P.Volgyesi. The Generic Modeling Environment, Workshop on Intelligent Signal Processing, Budapest, Hungary, May 2001.

[30] MetaEdit resources. URL: http://www.metacase.com/papers/index.html

[31] A.Kalnins, J.Barzdins, E.Celms, L.Lace, M.Opmanis, K.Podnieks, A.Zarins. The First Step Towards Generic Modelling Tool. Proceedings of Baltic DB&IS 2002, Tallinn, 2002, v. 2, pp. 167-180.

[32] E. Celms, A. Kalnins, L. Lace. Diagram definition facilities based on metamodel mappings. Proc. 18th International Conference, OOPSLA'2003 (Workshop on Domain-Specific Modeling), Anaheim, California, USA, October 2003, pp. 23-32.

[33] K. Ehrig, C. Ermel, S. Hänsgen, and G. Taentzer. Generation of visual editors as eclipse plug-ins. Proc. 20th IEEE/ACM International Conference on Automated Software Engineering, IEEE Computer Society, Long Beach, California, USA, 2005.

[34] A.Kalnins, J.Barzdins, E.Celms. Model Transformation Language MOLA. Proc. MDAFA 2004 (Model-Driven Architecture: Foundations and Applications 2004), Linkoeping, Sweden, June 2004. pp.14-28.

[35] A.Kalnins, J.Barzdins, E.Celms. Model Transformation Language MOLA: Extended Patterns. Selected papers from the 6th International Baltic Conference DB&IS'2004, IOS Press, FAIA 118, pp. 169-184, 2005.

[36] A.Kalnins, J. Barzdins, E. Celms. Efficiency Problems in MOLA Implementation. 19th International Conference, OOPSLA'2004 (Workshop "Best Practices for Model-Driven Software Development"), Vancouver, Canada, October 2004. URL: http://www.softmetaware.com/oopsla2004/mdsd-workshop.html

[37] A. Kalnins, J. Barzdins, E. Celms. MOLA Language: Methodology Sketch. Proc. EWMDA-2, Canterbury, England, pp.194-203, 2004.

[38] A.Kalnins, E. Celms, A. Sostaks. Tool support for MOLA. GPCE'05. Workshop on Graph and Model Transformation (GraMoT), Tallinn, Estonia, September 2005, pp. 162-173.

[39] A.Kalnins, E.Celms, A.Sostaks. Model Transformation Approach Based on MOLA. ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems, Workshop: Model Transformations in Practice (MTIP), Montego Bay, Jamaica, October 2005, 25p. URL: http://sosym.dcs.kcl.ac.uk/events/mtip/programme.html

[40] Object Management Group Request for Proposal: MOF 2.0 Query / Views / Transformations RFP. URL: http://www.omg.org/cgi-bin/apps/doc?ad/02-04-10.pdf

[41] Object Management Group MOF QVT Final Adopted Specification. URL: http://www.omg.org/cgi-bin/apps/doc?ptc/05-11-01.pdf