Object-Oriented Modeling of RDF Schema Ontologies

Efstratios Kontopoulos, Kalliopi Kravari and Nick Bassiliades

Department of Informatics, Aristotle University of Thessaloniki
GR-54124 Thessaloniki, Greece
{skontopo, kkravari, nbassili}@csd.auth.gr

Abstract

Ontologies are the primary knowledge representation tool in the Semantic Web and are mainly used in defining common vocabularies, used in the exchange of information among Semantic Web applications. In the process of encoding ontologies, appropriate ontology languages are applied; such a language is RDF Schema, one of the dominant standards. A variety of commercial and educational tools that address the tasks of developing and manipulating RDF Schema ontologies has been developed. None of them, however, is specifically destined for the inexperienced Semantic Web user. In this paper we present RDFSbuilder, a Java-built visual authoring tool for developing RDF Schema ontologies. The system helps users to develop their model quickly and efficiently, without being concerned about syntax or semantic errors. Furthermore, it adopts a purely object-oriented representation of the RDF Schema model, emphasizing on functional flexibility and simplicity of use. As a result, the model produced is easy to understand and equally easy to handle.

Keywords: RDF Schema, editor, ontology, semantic web, object-oriented modeling.

1. Introduction

The *Semantic Web* (SW) represents a relatively recent initiative to improve the potential of the existing Web [Berners-Lee et. al. (2001)]. The key idea is the creation of a universal medium for information exchange, by putting documents with computer – accessible meaning on the web. Thus, the organization and inter-connection of all the data found scattered on the web are essential for their efficient use by web applications.

The development of the SW is substantially based on *ontologies*. The ontology is a data model that represents a given domain and is used to reason about the objects in that domain and the relations between them. Contemporary ontologies share many structural similarities, regardless of the language in which they are expressed. An *ontology language*, on the other hand, is a formal language used to encode the ontology. There exist a number of such languages for ontologies, both proprietary and

standards-based, which provide descriptions that supplement or replace the content of Web documents. *XML* provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents. *RDF*, on the other hand, is a simple XML-based data model for referring to objects and how they are related [Beckett (2006)]. Finally, *RDF Schema* (RDFS) is a vocabulary for describing properties and classes of RDF resources [Brickley and Guha (2006)], with semantics for generalization hierarchies of such properties and classes, while *OWL* (Web Ontology Language) adds more capabilities for describing and constraining properties and classes [OWL (2006)].

Within the Semantic Web environment, developing and manipulating RDF Schema ontologies is a substantial but often tedious and error-prone task. Thus, the need for tools that assist end-users during the development of such ontologies is evident. Although several implementations exist of editors that create or manipulate RDF Schema ontologies, end-users, especially the unfamiliar ones, consider them difficult to handle. They are, thus, discouraged from developing their own ontologies, turning the vision of the SW into something most people are not able to conceive yet.

In this paper we present *RDFSbuilder*, a Java-built visual authoring tool for developing RDF Schema ontologies. The system helps users to develop their model quickly and efficiently, preventing them from syntax or semantic errors. RDFSbuilder emphasizes on offering a familiar development approach that closely assimilates visual object-oriented programming, i.e. using a variety of on-screen tools and drag & drop controls, contrary to graphic aids that focus on practical buttons and menus.

The rest of the paper is organized as follows: Section 2 introduces the syntax and semantics of RDF. Section 3 presents ontologies and RDF Schema, a primary ontology language in the SW domain. The next section introduces the object-oriented RDF Schema representation, adopted by RDFSbuilder, while section 5 presents the design and functionality of the system. Finally, section 6 discusses related work and section 7 concludes the paper, also discussing potential directions for future work.

2. RDF – A Common Information Exchange Model in the SW

Towards augmenting the available information in the Web with semantic content, *RDF* (*Resource Description Framework*) gave the solution; a common information exchange model for supporting resource description, or metadata (data about data), for the Web, usable by all SW applications.

The basic idea behind RDF is a model, consisting of a number of *statements* and connections between these statements. Thus, the statement is the basic building block of an RDF document and it consists of a *resource-property-value* triple:

- *Resources* are the objects we want to refer to or talk about. Every resource is uniquely identified by a Uniform Resource Identifier (URI).

- *Properties* are used to describe relations between resources, but, in practice, they are a special kind of resources, also identified by URIs.
- *Values* can either be resources or simply literals (strings).

An example of a statement is: The person `http://www.example.org/people.rdf#john` has a property `http:// www.example.org/people.rdf#age` with value "`26`". This statement declares that a specific person named John is 26 years old. Here "`http://www.example.org/people.rdf#john`" is the resource (or *subject*), "`www.example.org/people.rdf#age`" is the property (or *predicate*) and the value (or *object*) is "`26`".

There are various ways to represent an RDF statement, with the most important of them being the XML-based representation, which allows the re-usability of the various tools available for XML processing (*syntactic interoperability*). The XML fragment that expresses the above statement can be seen in Figure 1.

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:people="http://www.example.org/people.rdf">
     <rdf:Description rdf:about="&people;john">
          <people:age> 26 </people:age>
     </rdf:Description>
</rdf:RDF>
```

Figure 1. *XML-based syntax of an RDF statement*

Every statement is represented by an `rdf:Description` element. The subject of the statement is being referred to in the `rdf:about` attribute, the predicate is used as a tag and the object is the content of the tag.

Finally, *namespaces* should also be pointed out. Namespaces define a mechanism for resolving name clashes if more than one document is imported [Antoniou and van Harmelen (2004)]. According to namespaces, the names (actually URIs) of the elements in an XML/RDF document are defined, using a prefix declared with the command: `xmlns:prefix="URI"`, and a local name that is unique within the base URI. This way, the same local name can be used freely in many RDF/RDF Schema documents, since the existence of the prefix disambiguates things. The namespaces defined in an element can be used by that element and its descendants.

In RDF, external namespaces do not only offer disambiguation, as in XML, but they are also expected to be RDF documents defining resources, which are then used in the importing RDF document. This mechanism allows the reuse of resources by other people who may decide to insert additional features into these resources. The result is the emergence of large, distributed collections of knowledge.

In the example above (Figure 1), two namespaces are declared: `rdf` and `people`. The first contains the necessary vocabulary for RDF statements, while the other includes domain-specific vocabulary for representing people.

3. Ontologies and RDF Schema

The main knowledge representation tool in the SW is the *ontology*, which is simply a structured representational formalism that shares a lot of common elements with frames and semantic nets in AI. The ontologies are mainly used in defining common vocabularies, used in the exchange of information among SW applications.

However, ontologies also offer interoperability among the available information and the various WWW applications like search engines, portals, intelligent agents and Web services. For example, today's search engines have a low precision and high recall. If there existed the possibility to perform an ontology-based search, then the input keywords would be associated with the intended meaning. The search results would not only be more accurate, but a number of results that would contain conceptual synonyms of the inserted keyword would also be returned.

The main ontology languages in the SW today are *RDF Schema* and *OWL*. The former is a vocabulary description language for describing properties and classes of RDF resources, while the latter is a richer vocabulary description language that can also describe relations between classes, cardinality, equality etc. As its name implies, RDFSbuilder handles RDF Schema ontologies only.

Thus, although RDF lets the user describe resources using a custom vocabulary, as seen in the previous section, apparently the level of expressivity it offers is not sufficient. RDF Schema, RDF's "*semantic extension*", equips users with mechanisms for describing domains and correlations among resources, namely the allowed vocabulary for resources and properties, i.e. resources types and property types.

Though there are significant differences between RDF Schema and conventional object-oriented programming languages, the former is also based on classes and properties, similarly to the latter. Therefore, *classes* are sets of resources, while members of classes are called *instances*. On the other hand, users can also describe specific *properties* of class instances; properties impose a relation between subject and object resources and are characterized by their *domain* (the class of resources that may appear as *subjects* in a triple with the property as predicate) and *range* (the class of resources that may appear as *values* in a triple with the property as predicate).

Instances are associated with the corresponding classes by declaring their type, using a statement like: `people:john rdf:type people:person`. This statement declares that instance `john` belongs to the class `person`. However, this is also indirectly extracted by the fact that `rdf:type` has `rdfs:Class` as range, which immediately results in `person` being a class, according to the descriptional nature of the RDF semantics.

Naturally, the notions of hierarchy and inheritance are also applied in RDF Schema. However, they are not only applied in the case of classes but in the case of properties

as well. Thus, there can be defined *super-* and *subclasses* of a specific class, but there can also exist *super-* and *subproperties* of a specific property.

An example of all the above can be seen in Figure 2. Two classes are described: class "`person`" and its subclass "`parent`". Similarly, two properties are also described: "`hasParent`", which has class "`person`" as its domain and class "`parent`" as its range and "`hasFather`", which is a subproperty of "`hasParent`" and, thus, inherits from the latter the domain and range. Note here that, contrary to traditional object-oriented programming, properties are not part of a class description, but are "*globally*" visible in RDF Schema ontology. This not only imposes a different programming approach, but also offers flexibility in extending ontologies.

```
<rdfs:Class rdf:ID="person"/>
<rdfs:Class rdf:ID="parent">
     <rdfs:subClassOf rdf:resource="#person"/>
</rdfs:Class>
<rdf:Property rdf:ID="hasParent">
     <rdfs:domain rdf:resource="#person"/>
     <rdfs:range rdf:resource="#parent"/>
</rdf:Property>
<rdf:Property rdf:ID="hasFather">
     <rdfs:subPropertyOf rdf:resource="#hasParent"/>
</rdf:Property>
```

Figure 2. *Classes and Properties in RDF Schema*

4. The Object-Oriented RDF Schema Representation

RDFSbuilder adopts a *purely* object-oriented model for RDF Schema, adopted from the R-DEVICE system [Bassiliades and Vlahavas (2006)]. According to this model, properties are encapsulated as attributes in classes, resulting in a representation that appears dissimilar from the standards of the RDFS model, but is not very differentiated from UML class diagrams [UML (2006)]. Classes are represented as rectangles, properties are encompassed by the classes and the subclass relationship is represented by an arrow that commences from the subclass and ends on the superclass rectangle. Users can also define globally visible properties, by declaring `rdfs:Resource` as the property domain. Figure 3 displays an example of two classes in RDFSbuilder that share a super-/subclass relationship, accompanied by the corresponding RDF Schema fragment.

Developing an ontology following this practice accomplishes the primary design goal of the system, i.e. to make it directly usable by users that are not accustomed to the particular modelling style of RDF Schema. A development approach is applied, which is very similar to common visual object-oriented programming. Nevertheless, this approach is superficial and the final result, namely the ontology eventually developed, remains compliant with the specifications of the RDF Schema model, according to which, properties have a global scope and are not strictly encapsulated by classes.

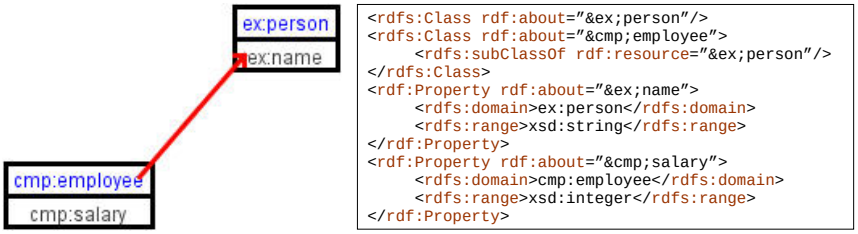

Figure 3. *Example of two classes in RDFSbuilder, accompanied by the corresponding RDF Schema fragment*

5. RDFSbuilder Design and Functionality

RDFSbuilder is a Java-built visual development tool that allows users to develop and deploy RDF Schema ontologies in a visual fashion. By using the system, users can develop their model quickly and efficiently, without being concerned about syntax or semantic errors. An overview of the system is depicted in Figure 4.

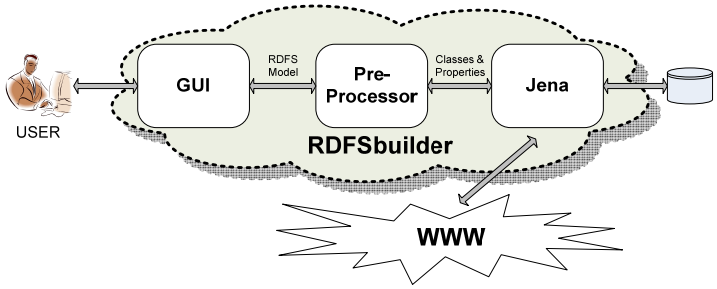

Figure 4. *RDFSbuilder system overview*

As can be observed, RDFSbuilder consists of the following subsystems: (a) the *GUI* (Graphical User Interface), concerned with the interaction between the system and the end-user. It constitutes the tool for modelling and viewing the ontologies developed as well as those imported from the Web, (b) the *Pre-Processor*, the module responsible for translating the graph drawn (or modified) by the user into vectors of classes and properties and vice-versa, i.e. convert the ontologies loaded into a sound RDFSbuilder graph and (c) *Jena Semantic Web Framework* [McBride (2001)], a flexible Java API for processing RDF documents, primarily concerned with transforming the data regarding the graph (number and type of classes and properties etc.) into a valid RDF Schema model, as well as for parsing the ontologies loaded or those imported from the Web.

5.1 The User Interface

Figure 5 displays the main window of the system, which is composed of two major parts: the upper part includes the toolbar, which contains icons, representing the most

common utilities of the editor, while the central part comprises the drawing panel of the main window, where the user can design the ontology model.

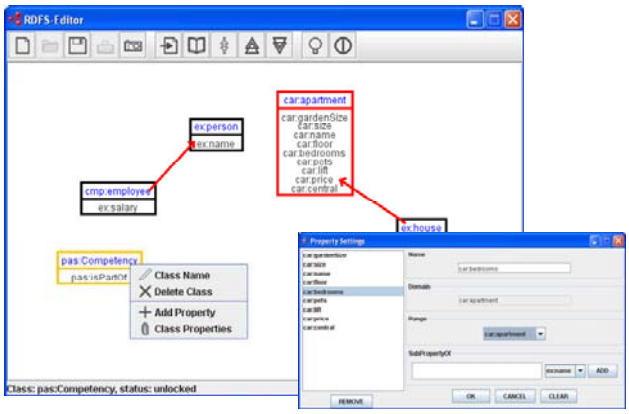

Figure 5. *The main window of the system and the properties dialog box*

By right-clicking on a spare space within this drawing panel, a popup menu with the choice "*Insert Class*" appears. The user has the option of (a) creating a completely new "empty" class, which can then be manipulated and "filled" with properties or, alternatively, (b) importing a class that belongs to an existing ontology. The process behind the second option is more extensively described in the following section. The new class is drawn at the point, over which the mouse button was initially clicked.

The characteristics of each class can be easily modified, by right-clicking on the appropriate class. A second popup menu shows up, which includes a variety of choices that can be notionally divided into two groups: the first group is concerned with the classes and includes the "*Class Name*" and "*Delete Class*" options, while the second group is concerned with the properties and includes the "*Add Property*" and "*Class Properties*" options.

- The functionality behind the "*Class Name*" and "*Delete Class*" options is quite straightforward, allowing the user to modify the class name and delete the class respectively.
- "*Add Property*" allows users to add a new property to the class, by declaring its name, namely prefix and id, and its range. The class domain is automatically set to the corresponding class.
- "*Class Properties*" let the user modify the name and/or range of a property and remove properties from the class. Moreover, the user is allowed to set the relation of "`subPropertyOf`" between this property and one or more other properties.

Defining subclass relationships is equally easy. By clicking on the "*Define Subclass*" toolbar button, the mouse cursor automatically changes to a "*hand*" cursor, waiting

for the user to indicate the subclass and the superclass. All the user has to do is drag a line from the subclass towards the superclass and, on the release of the mouse button, a "subclass" arrow, similar to the one in Figure 3, is drawn that connects the two classes. Classes that have no superclass in the graph are considered to be direct subclasses of rdfs:Resource, which represents the class of all resources.

Furthermore, although the default size of the drawing panel is relatively limited (i.e. 640x480 pixels), the user is offered the capability of dynamically adjust its size, when the ontology model developed becomes too big to fit. At this case, the changes in the panel size are not directly visible, but the user can traverse the newly-created parts of the graph, by using the scroll-bar controls located at the edges of the main window of the program.

5.2 Importing Classes and Properties from an Existing Ontology

One of the important aspects of the RDF model is the *namespaces*. As explained in a previous section, namespaces introduce a means to resolve name clashes (that would otherwise be inevitable) among ontologies from different sources. However, RDFSbuilder gives namespaces an extra meaning, by treating them as *addresses* of input RDF Schema ontologies that contain essential vocabulary for the modeling of the ontology under development. The namespaces applied are used in pull-down menus and lists, in order to prevent potential errors on behalf of the user.

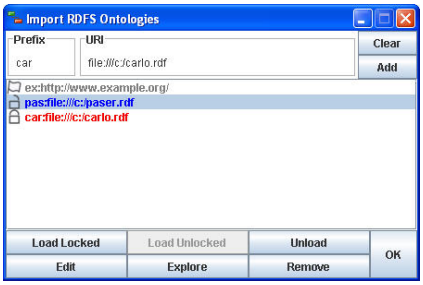

Figure 6. *Importing an RDFS ontology*

Thus, importing classes and properties from an existing RDF Schema ontology is quite simple: by clicking on the appropriate toolbar button a frame appears (see Figure 6), where the user can insert the URL of the ontology to be imported as well as a corresponding prefix. Note that a default system-defined namespace (`ex:http://www. example.org/`) is already declared.

By clicking on the "Add" button, the ontology is added to the list of imported ontologies, but is not yet loaded. The user now has the choice of loading the ontology either *locked* or *unlocked*; the former allows no modifications to the classes and properties imported from the specific ontology, while the latter performs exactly the opposite functionality, allowing the user to modify the imported elements.

Alternatively, users can keep the ontology in an unloaded status (i.e. ontologies simply appear in the list, but do not affect the ontology under development).

When the user selects ontology to be loaded (either locked or unlocked), the selected ontology is downloaded and parsed by Jena, which collects all the classes and properties contained in the loaded RDFS document. The elements collected will then be available during the modeling of the ontology developed and the user is offered the capability either to insert an imported class (see previous section), including the corresponding properties (properties that have the specific class as their domain), or to apply an imported property, through the "Class Properties" menu (also described in the previous section).

Imported classes that belong to a loaded ontology can then be added to the current model, with their locked/unlocked status indicating whether they can be modified, enhanced or extended. Visually, imported classes are distinguished from the rest by their outline color, which is *red* for the "locked" classes and *blue* for the "unlocked".

The user can also manually "discover" more namespaces, by pressing the "*Explore*" button in the bottom of the window. The system then downloads the namespace documents contained within the specific document and displays them in the namespaces list, accompanied by an "Unloaded" flag.

5.3 Exporting the RDF Schema Ontology

Besides RDF/XML syntax (normal and abbreviated), the RDFS ontology developed can also be exported to Notation 3 and N-Triple formats:

- *Notation 3* (N3) comprises a compact and easily readable alternative to RDF's XML syntax, extended to allow greater expressiveness. N3 files typically have the extension '`.n3`'.
- *N-Triples* is a line-based, plain text format for encoding an RDF graph. It was designed to be a fixed subset of N3. N-Triples content is typically stored in files with an '`.nt`' suffix to distinguish them from N3.

6. Related Work

There exist several implementations of editors that create or manipulate RDF Schema documents. *MR3* [Takeshi et. al. (2006)] is such a tool for editing RDF-based content. It is efficient and supports all the basic functions of the RDFS model, but follows the traditional RDF visual representation, namely, the graph is created by drawing a distinct geometrical figure (i.e. ellipses and boxes) for each entity, which finally leads to a quite confusing graph. Furthermore, its utilities are complicated to use by an unfamiliar user and, thus, comprises a limited solution, suitable only for experienced users.

Altova SemanticWorks [Altova (2006)] is a commercial product, which also offers all the basic functions of the RDFS model. Users can visually design Semantic Web instance documents, vocabularies, and ontologies and then output them in either RDF/XML or N-triples formats. Its RDF representation and utilities, however, are extremely sophisticated even for familiar with RDF and RDF Schema users. Furthermore, it is an obscure system, difficult to use, as the representation of the graph is too complicated to understand and to handle.

Protégé [Protege (2006)], also a commercial product, is an open-source ontology editor and knowledge base framework. It supports the creation, visualization, and manipulation of ontologies not only in RDFS but also in OWL, RDF and XML Schema. Protégé also features flexible plug-in mechanisms that add extensibility to the system as well as a wide user community, involved in a variety of research and industrial projects. However, Protégé features a modeling environment, based on graphical aids and mainly a tree representation of the ontologies developed, while RDFSbuilder is purely visual. Nevertheless, there also exists OWLViz, one of Protégé's plug-ins, but it is merely a visualization tool for OWL ontologies, not allowing development or modeling of ontologies.

IsaViz [Pietriga (2006)] and *RDFSViz* [Sintek (2006)] are two more similar systems. The former represents models as directed graphs, using the traditional representation of a model. It is simple to use but with limited functionality. The latter is a web implementation of an RDFS visualisation service. Its online demo allows users to enter the URL of their own RDFS files and the corresponding graph is generated. The main drawback of both systems is that they are not editors but visualization tools.

7. Conclusions and Future Work

In this paper we argued that ontologies are the fundamental representation tool in the Semantic Web and RDF Schema is one of the dominant ontology languages. RDF Schema ontologies are extremely useful in the exchange of information among SW applications. However, there is a lack of ontology developing and manipulating tools, particularly destined for non-experienced users. We have developed RDFSbuilder, a visual authoring tool to facilitate users in developing RDF Schema ontologies. RDFSbuilder allows users to develop their model quickly and efficiently, without being concerned about syntax or semantic errors. Furthermore, contrary to the majority of similar systems, it adopts a completely object-oriented representation of the model, which enhances functional flexibility and simplicity of use. As a result, the graph produced is not only easy to understand but also extremely easy to handle.

Our future plans for RDFSbuilder include enhancing it further with RDF editing, so that the next version of the tool will constitute a full visual RDF and RDF Schema development environment. We also plan to proceed to an extensive user evaluation, which will identify what the users really need and prefer and the results will be used

to provide an even more user-friendly environment. However, our ultimate aim concerning RDFSbuilder involves its integration with an existing system we have developed, called VDR-DEVICE [Bassiliades et. al. (2005)]. VDR-DEVICE is a visual, integrated development environment for modeling and using defeasible logic rule bases on top of RDF ontologies in the SW environment.

7. Acknowledgements

This work was partially supported by the PYTHAGORAS II program which is jointly funded by the Greek Ministry of Education (EPEAEK) and the European Union.

References

Altova, SemanticWorks (2006), *Visual Semantic Web design tool for RDF and OWL*, http://www.altova.com/products/semanticworks/semantic_web_rdf_owl_editor.html, last accessed: November 20, 2006.

Antoniou G., Harmelen F. van (2004), *A Semantic Web Primer*, MIT Press.

Bassiliades N., Kontopoulos E., Antoniou G. (2005), *A Visual Environment for Developing Defeasible Rule Bases for the Semantic Web*, in Proc. RuleML-2005: International Conference on Rules and Rule Markup Languages for the Semantic Web, (pp. 172-186), Springer-Verlag, LNCS 3791, Galway, Ireland.

Bassiliades N., Vlahavas I. (2006), *R-DEVICE: An Object-Oriented Knowledge Base System for RDF Metadata*, International Journal on Semantic Web and Information Systems, Idea Group, Vol. 2, No. 2, pp. 24-90.

Beckett D. (ed.) (2006), *RDF/XML Syntax Specification*, http://www.w3.org/TR/rdf-syntax-grammar/, last accessed: November 29, 2006.

Berners-Lee T., Hendler J., Lassila O. (2001), *The Semantic Web*, Scientific American, 284(5), pp. 34-43.

Brickley D., Guha R.V. (eds.) (2006), *RDF Vocabulary Description Language 1.0: RDF Schema*, http://www.w3.org/TR/rdf-schema/, last accessed: November 29, 2006.

McBride B. (2001), *Jena: Implementing the RDF Model and Syntax Specification*, in Proc. 2nd Int. Workshop on the Semantic Web.

OWL (2006), *Web Ontology Language*, http://www.w3.org/2004/OWL/, last accessed: November 25, 2006.

Pietriga E. (2006), *IsaViz: A Visual Authoring Tool for RDF*, http://www.w3.org/2001/11/IsaViz/, last accessed: November 20, 2006.

Protégé (2006), *Ontology Editor and Knowledge Acquisition System*, http://protege.stanford.edu/, last accessed: December 4, 2006.

Sintek M., Lauer A. (2006), *The FRODO RDFSViz Tool*, http://www.dfki.uni-kl.de/frodo/RDFSViz/, last accessed: November 20, 2006.

Takeshi M., Noriaki I., Naoki F., Takahira Y. (2006), *A Graphical RDF-based Meta-Model Management Tool*, IEICE Transactions on Information and Systems, Special Issue on Knowledge-Based Software Engineering, Vol.E89-D, No.4, pp 1368-1377.

UML (2006), *Unified Modeling Language*, `http://www.uml.org/`, last accessed: November 6, 2006.