# Can Software Developers Use Linked Data Vocabulary?

**Matthias Quasthoff, Harald Sack, Christoph Meinel**
(Hasso-Plattner-Institut, Potsdam, Germany
{matthias.quasthoff, harald.sack, meinel}@hpi.uni-potsdam.de)

**Abstract:** One of the next research goals in Semantic Web-enabled software engineering is to naturally use Semantic Web data within arbitrary applications. This will in most cases require some kind of mapping between object-oriented and graph-based information representations. In this paper, we investigate how well Semantic Web vocabularies specified in different formats can be mapped onto object-oriented representations. We examine what a software engineer would expect from such mappings, and how well existing, widely used vocabularies meet these expectations.

**Keywords:** Design Patterns, Linked Data, Object-Orientation, RDF
**Categories:** D.1.5, D.2.13, M.6, M.8

## 1 Introduction

Publishing, consuming, and processing RDF data is becoming more common for modern desktop and Web applications. Web-scale data integration is the dedicated goal of the Linking Open Data (LOD) initiative providing RDF-based vocabularies with billions of RDF-triples [Bizer et al. (2008)]. However, compared to developer support for traditional data structures and storage systems, such as relational databases, developer support for linked data is still in its infancy. Good support will allow software developers to process RDF [Manola and Miller (2004)] data within the programming environment and using the programming language of their choice. E.g., in an object-oriented (OO) programming language [see Gosling et al. (2005)], RDF resources should be represented as objects, and RDF schemata should be represented as ready-to-use packages for the desired programming environment [Quasthoff and Meinel (2008)]. Because Semantic Web programming interfaces have been designed rather by knowledge engineers than by traditional software engineers, using Semantic Web data structures is not straightforward for the software engineer not being an expert in Semantic Web knowledge engineering. We believe that the missing ease of use is one of the main reasons why Semantic Web technology entered the field of software engineering so hesitatingly.

As the main contribution of this paper, we define general requirements to RDF-to-OO mappings from the point of view of the software engineer, and investigate which parts of RDF schema [Brickley et al. (2004)] and OWL [Patel-Schneider et al. (2004)] contribute to such mapping. We also examine a number of RDF and OWL vocabularies widely used on the World Wide Web (WWW) how they make use of the features of their schema language helping software engineers to take full benefit, e.g., for generic mapping technologies, or for simplified data access. Finally, we lay out guidelines for the definition of Semantic Web schemata to simplify their usage in software projects while retaining their full potential.

The rest of this paper is organized as follows: In Section 2, we discuss related work on the Web of data and the mapping between data representations. In Section 3, we discuss expectations and limitations of mappings between RDF and OO and how such mappings can be obtained. In Section 4, we examine several popular RDF vocabularies on their suitability for automatic mapping. Section 5 concludes the paper.

## 2 Related work

Linked Data has become one of the most popular topics among the emerging Semantic Web [Berners-Lee (2006)]. By *Linked Data* we refer to a method of exposing, sharing, and connecting data via dereferenceable URI on the WWW. The Linking Open Data community project has picked up this approach to extend the Web of data by publishing various open data sets being represented as RDF and by defining links and mappings between vocabularies and data items from different data sources. Thus, the data available within the LOD has grown to more than 4,5 billion RDF triples and about 180 million RDF links[1]. This publicly available interconnected data enables the development of numerous data mash-up applications. To clear the way for efficient application development, a straightforward inclusion of these semantic data into the traditional software engineering process has to be implemented.

To achieve this goal, design patterns for implementation tasks of linked data applications have to be identified and described. Design patterns are not programming libraries, but solutions to frequently recurring problems, and have been introduced to software engineering by [Gamma et al. (1994)]. In previous work we showed that object-relational mapping (ORM) [Fowler and Rice (2003)] can be adopted to suite mapping between RDF and OO data [Quasthoff and Meinel (2008)], which we refer to as object-triple mapping (OTM). A number of software projects[2] address the need of OTM, but do not describe underlying design patterns for retrieving information, obtaining a mapping between RDF and OO definitions, mapping the data, and checking for policy or license compliance [see Miller et al. (2008), Weitzner et al. (2006)].

## 3 Requirements to linked data software engineering

### 3.1 Schema mapping basics

Mixing different types of programming languages (e.g., imperative and declarative languages) creates programming overhead that should be avoided for various reasons [Fowler and Rice (2003)]. Rather, OO software developers must be able to process Web resources as objects native to their programming language. That is, not triples, URI, and other entities making up RDF data should be primarily visible to software developers, but classes and objects carrying the triple information as fields and values. The goal is to support the whole life-cycle of a Web resource from the OO perspective: resources need to be retrieved or created locally, to be processed with imperative statements, and to be published as linked data, or stored locally.

*Joint model.* The OO equivalent to vocabulary are packages containing classes, whose fields are equivalent to properties in the vocabulary. Intuitively, one would expect a one-to-one mapping of RDF classes to OO classes, and RDF properties to OO fields.

1 http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData#dbpedia-lod-cloud, (March 2009)

2 https://sommer.dev.java.net/, http://semanticweb.org/wiki/RDFReactor

However, due to the open world assumption, the number of properties an RDF class has depends on the context, i.e. the vocabularies we are considering. Hence, a single RDF class can have more than one OO representation. Inheritance relationships between RDF classes can be mapped on equivalent OO inheritance relationships. Also, most OO programming languages do not allow multi-valued fields, but use collection (or list or array) objects containing an arbitrary number of values to achieve this. Depending on the cardinality of the property to map, a developer would expect a field to contain a collection or just a plain object. And of course, a useful mapping translates typed literals, e.g., from XML Schema Definition [Thompson et al. (2004)], as accurate as possible onto primitive types of the programming language.

*Differing expressivity.* While some restrictions on properties such as cardinality, range, and domain can be mapped onto class definitions in most OO programming languages, other features of either sides cannot be trivially mapped. Inverse-functional properties cannot be enforced on OO classes without explicit validation code. Also, sub-property relationships do not have a direct OO equivalent. Vice-versa, not all features of OO programming languages can easily be represented in RDF schemata. E.g., OO classes frequently feature fields containing string objects, whose content varies depending on localization settings of the runtime environment. Restricting a property's cardinality per language in RDF or OWL is not trivial and involved rules.

*Implicitness vs. explicitness.* According to RDF Schema, from statements using some property one can conclude that subject and object belong to the property's domain and range. Hence, anything appearing as a property's domain or range (which are RDF properties themselves) can be concluded to be an RDF class. In many OO programming languages, classes and fields have to be explicitly defined as classes and fields. Therefore, an automatic translation from RDF schema to OO cannot happen by syntax only, but requires at least some kind of inferencing.

## 3.2 Schema mapping process

In this section we will present three options how software developers can obtain OO packages for specific vocabularies. The options are influenced by two parameters:

- automatic vs. manual mapping;
- local generation vs. retrieval.

These parameters result in three basic ways to obtain a RDF/OO mapping:

1. Mapping an existing domain model: software developers can manually map OO classes to RDF within their software projects;
2. software developers can automatically generate OO classes from vocabularies or download such generated mappings from the WWW;
3. software developers can download hand-edited mappings from the WWW.

From a general point of view, an automatic mapping, or retrieval of mapping information is preferable over local, manual mapping. Still, either of the three alternative has benefits and drawbacks being discussed in this section.

*Mapping an existing domain model.* To enrich an existing application with linked data, those fields of OO classes that do have an RDF or OWL equivalent can be mapped locally and manually onto the respective vocabulary. This way, OO data can be rendered to RDF and made available to other applications. Retrieving RDF data from the WWW and rendering it as OO objects can be difficult, as application-specific initialization data required to instantiate the class might be hard to obtain for a generic mapping mechanism. Also, if a retrieved resource belongs to RDF classes mapped to two disjoint concrete OO classes, for most OO programming environments only one of the classes can be instantiated, and hence only some of the resource's properties be reflected in OO. This can be circumvented by separating the mapping to abstract interfaces and letting the domain model implement the respective interfaces. That way, domain objects can still be rendered to RDF, but RDF data can now be instantiated to OO objects implementing the relevant interfaces. The manual effort for the software developer includes choosing the vocabulary to map to, and mapping existing OO classes to RDF, after optionally separating this mapping into abstract interfaces. The benefit for software developers is that they can use their coding conventions for the domain model and only map those parts relevant for the specific application to RDF, especially if a class is expected to contain properties from different vocabularies.

*Generating OO classes.* Instead of letting software developers manually generate OO classes corresponding to vocabulary, the schema definitions can be used to generate OO class definitions. The source code generator needs conventions for

- converting schema namespace to OO package names, i.e. converting URI to some programming language-specific path hierarchy;
- converting RDF class and property to OO class and field names, i.e. converting relative URI to identifiers in the OO programming language, following some widely accepted convention for, e.g., capitalization;
- storing the actual mapping information, i.e. creating separate mapping metadata, or keeping the mapping metadata as source code annotations [see Gosling et al. (2005)];
- resolving conflicts of class or property names with the programming language's reserved words or conflicting OO class and field definitions.

From the developer's perspective, there is no fundamental difference between automatically generating such a mapping locally, or retrieving a mapping package from the WWW. After obtaining the mapping, local domain classes can be modified to implement the abstract interfaces generated or retrieved. Although simplifying the bootstrapping overhead for the developer compared to manual mapping, automatic mapping heavily relies on accurate schema definitions. This includes among others reasonable domain and range specifications for properties, cardinality constraints where feasible, and good schema documentation using RDFS labels and comments. As will be shown in the next section, automatic source code generation from existing schema can sometimes give disappointing results. Hence, a hybrid approach using generated classed, which will be hand-edited and offered for download along with the schema definitions will be a good solution.

*Publishing manually edited OO definitions.* To circumvent the shortcomings of automatically generated OO classes, service providers can fine-tune the generated OO packages and let software developers download the results as commonly usable packages. The packages could be provided by the authors of the vocabularies themselves, or be published in special repositories. This approach would also mitigate some problems, such as naming incompatibilities between RDF and OO.

## 4 Evaluation of automatic RDF to OO mapping

We used an extended version of the RDF/OO mapping prototype presented in previous work [Quasthoff and Meinel (2008)] to process 33 vocabularies. The schema definitions have been downloaded from the WWW and used to generate abstract Java classes. We investigated vocabulary listed in a schema directory[3], and vocabulary we used in previous work. All in all, these vocabularies contained 1519 classes and 1727 properties. The primary findings concerned compliance to OO coding conventions on the one hand side, and the correctness of the mapping on the other.

| Abbrev. | Namespace | #c | #p |
|---|---|---|---|
| *cc* | http://creativecommons.org/ns# | 6 | 10 |
| *dbpedia* | http://dbpedia.org/ontology/ | 174 | 720 |
| *dc* | http://purl.org/dc/terms/ | 22 | 55 |
| *dcel* | http://purl.org/dc/elements/1.1/ | 0 | 15 |
| *dcmi* | http://purl.org/dc/dcmitype/ | 12 | 0 |
| *doac* | http://ramonantonio.net/doac/0.1/ | 15 | 16 |
| *doap* | http://usefulinc.com/ns/doap | 7 | 30 |
| *eor* | http://dublincore.org/2000/03/13/eor# | 4 | 3 |
| *foaf* | http://xmlns.com/foaf/0.1/ | 12 | 54 |
| *kaos* | http://ontology.ihmc.us/ | 134 | 88 |
| *mm* | http://musicbrainz.org/mm/mm-2.1# | 19 | 10 |
| *rddl* | http://rddl.org/rddl.rdfs# | 2 | 3 |
| *rev* | http://purl.org/stuff/rev# | 3 | 13 |
| *sioc* | http://rdfs.org/sioc/ns# | 11 | 66 |
| *sw* | http://sw.nokia.com/SWArch-1/ | 3 | 3 |
| *swrc* | http://swrc.ontoware.org/ontology# | 54 | 74 |
| *umbel* | http://umbel.org/umbel/ac | 748 | 9 |

| Abbrev. | Namespace | #c | #p |
|---|---|---|---|
| *bibtex* | http://purl.org/net/nknouf/ns/bibtex# | 15 | 40 |
| *cv* | http://kaste.lv/~captsolo/semweb/resume/cv.rdfs# | 16 | 72 |
| *cv-base* | http://kaste.lv/~captsolo/semweb/resume/base.rdfs# | 8 | 0 |
| *eswc* | http://www.eswc2006.org/technologies/ontology | 72 | 38 |
| *fresnel* | http://www.w3.org/2004/09/fresnel# | 16 | 29 |
| *geo* | http://www.w3.org/2003/01/geo/wgs84_pos# | 2 | 4 |
| *ical* | http://www.w3.org/2002/12/cal/ical# | 14 | 48 |
| *iswc* | http://annotation.semanticweb.org/2004/iswc# | 33 | 35 |
| *photo* | http://purl.org/net/vocab/2003/11/photo# | 1 | 10 |
| *pim* | http://www.w3.org/2000/10/swap/pim/contact# | 7 | 19 |
| *rei* | http://www.cs.umbc.edu/~lkagal1/rei/ontologies/ | 65 | 57 |
| *schema* | http://www.schemaweb.info/schemas/meta/rdf/ | 2 | 10 |
| *skos* | http://www.w3.org/2004/02/skos/core# | 4 | 28 |
| *vs* | http://www.w3.org/2003/06/sw-vocab-status/ns# | 0 | 3 |
| *vcard* | http://www.w3.org/2001/vcard-rdf/3.0# | 7 | 43 |
| *wot* | http://xmlns.com/wot/0.1/ | 5 | 13 |
| | | | |

*Table 1. Vocabularies investigated in this paper (#c: classes, #p: properties)*

*Complex mapping logic required.* Two vocabularies have been found leaving some parts of the specification implicit by relying on the semantics of RDF Schema (Creative Commons, *cc*) and OWL (Friend of a friend, *foaf*). The *cc* vocabulary does not explicitly declare classes, but describes some resources appearing as range or domain of RDF properties, and hence being classes. Similarly, for all *foaf* properties it is explicitly stated whether it is an OWL datatype or object property. Only for *isTopicOf*, which is inverse to an object property, this is left implicit.

A similar finding, yet not related to inferencing, concerns the definition of restrictions on properties in OWL. In RDF schema, for each class *c* to map we can issue a straight-forward query on properties having *c* as range. In OWL, we need to query for *Restriction* superclasses of *c* and map the properties following the restriction's *onProperty* predicate. This kind of schema processing makes processing OWL ontologies harder than RDF schemata, and again proves that average software developers need support when processing such vocabularies. Even worse, the OWL method of indicating property range per domain class is incompatible with OO programming languages like Java. E.g., the *KAoS* vocabulary cannot be accurately mapped to Java classes: *Groups* can have arbitrary *members*, but *ActorGroups* and *PersonGroups* can only take *Actors* and *Persons* as *members*. Java requires the *member* field defined in the OO *Group* class to have the same type for the *ActorGroup* and *PersonGroup* subclasses, hence *ActorGroup* and *PersonGroup* cannot be accurately translated to OO.

*Empty OO classes*. By average, 48% of all classes defined in the vocabulary evaluated never appeared as domain of a property nor did they extend any OWL restriction

3 http://schemaweb.org/

class. Four vocabularies contained only such “empty” classes. E.g., the Description of a project (*doap*) vocabulary defines a class *Repository*, which has several empty subclasses for different revision control systems. RDF classes without properties will be translated to OO classes without any fields. If an OO developer was about to model such structure, she would probably generate an enumeration datatype for the revision control system and add a *type* field to the *Repository* class taking one of the enumeration values. For future automatic mapping mechanisms, a heuristic or schema extension needs to be defined, whether a empty RDF class should be mapped to an OO class, or to a logical field value indicating class membership, or an enumeration field.

*Naming conflicts.* As mentioned before, any mechanism generating an RDF/OO mapping will have to translate names of RDF classes and properties to OO class and field names. As the OO names should somehow reflect the relative URI of the RDF or OWL names, it can happen that two distinct properties will be mapped to distinct fields carrying the same name. In our evaluation, conflicts arose with the *fresnel:label* and *rev:comment* vs. the respective RDFS properties and with *foaf:publication* vs. *doac:publication*, both having domain *foaf:Person*. If a generator has access to all vocabularies subject to OO class generation, as would be the case with local mapping or generation, the technical part of the problem can be solved, e.g. using such rules:

1. prefix the OO name of that conflicting property that has a domain outside its own vocabulary (e.g., *doacPublication* on *foaf:Person*);
2. prefix the OO name of properties conflicting with a superclass’ properties (e.g., *fresnelLabel*, *revComment* to avoid conflicts with the RDFS properties)

But if, as would be the case with pre-produced mappings scattered over the WWW, different vocabularies would be translated independently of each other, any resource belonging to two classes from different vocabularies sharing a conflicting property name could not be instantiated as OO object. These problems could be solved using dynamically-typed programming languages, which are becoming more popular.

*Documentation, correctness, usability.* Three of the vocabularies investigated did not include any comments or labels on the classes and properties they defined. One vocabulary had no comments and labels on classes, but only property labels. Two more did not include any comments, but did include labels, which have mostly been human-readable versions of the relative class or property URI, and hence did not provide additional information. By average, 60% of the classes and 43% of the properties of the vocabularies investigated had comments. Comments are very important for modern software engineering, as they allow external developers to efficiently use code from other sources. On the contrary, the *doap* vocabulary features comments in English, French, and Spanish. Further investigations how to optimally use this multilingual information within the software development lifecycle will be interesting.

Besides limited developer support through comments or labels, typing errors and unavailable schemata have been a problem during our evaluation. E.g., the *eswc* vocabulary specifies additional properties for a *foaf:Organisation* class, whereas *foaf* only specifies a *foaf:Organization* class. Such errors just reduce the usability of the affected vocabularies. Also, the *umbel* vocabulary includes an OWL *Restriction* having *minCardinality*’s the parse type not explicitly set to non-negative integer, resulting in the Jena framework parsing the numeric value as a string and hence, in our class generator ignoring the cardinality information. Over the course of the evaluation, several vocabularies could not be retrieved temporarily (*dbpedia* and *umbel* on April 2, *KAoS* on April 3, 2009). Since vocabulary is central to developing simpler linked data applications, the schemas should be highly available.

The usability of a vocabulary is reduced if for the sake of generality an RDF property's range or cardinality is less restricted than software engineers would wish in their specific applications. Reading the *foaf:name* (or any other of the 92% unrestricted properties found) of an OO object requires to get the collection of *foaf:names* of the object, checking if this collection is empty, and if not so, choosing the right name from this collection. If a specific application is guaranteed to only store one name per *foaf:Person*, a direct access to this name is desirable, resulting in reduced recurring programming. To mock this, our OO class generator currently creates for each property both a scalar field and a collection field in the OO class unless the property is restricted otherwise, and the developer can choose which field she preferably reads or writes. Also, 15 vocabularies did not follow the linked data principles, as they could not be retrieved using the namespace URI only.

## 5 Conclusion and outlook

In this paper, we argued for simplified access to linked data for the average software engineer. Such simplifications include transparent access to RDF and OWL schemata with the means of widely accepted OO programming languages. We analysed the expectations to such mapping between RDF/OWL and OO, and discussed general limitations. We then used an extended version of our RDF/OO mapper presented in previous work to actually process a number of relevant RDF/OWL schemata from the Web of data, and investigated certain mapping-related features of these schemata. As a result of this investigation, we can postulate recommendations from the software engineering perspective for the development of future schemata:

1. *Be explicit.* If a resource is an RDF class, say so. Leaving this information implicit makes processing schemata unnecessarily complex and prevents software engineers from becoming familiar with linked data.
2. *Be accurate.* Restrict domain, range, and cardinality of properties, if possible. Leaving this overly flexible will lead to interpretation among data sources and hence limit automatic processing.
3. *Be concise.* If you are about linking data (and not creating a taxonomy), you don't need lots of classes that never appear as range or domain of a property.
4. *Be cooperative.* Many vocabularies exist for various purposes. If you want to create a property with the same (relative) URI of an existing property, see if you can use or least link to it. If this is not feasible, give your property a distinct name.
5. *Be perfect.* Add comments and (multi-lingual) labels to classes and properties that help software developers or can be used in some application's presentation logic. Avoid typos in URI at any time as they break links from your vocabulary to others. And make sure you follow linked data principles.

Unfortunately, these recommendations have been violated for a large number of vocabularies we found on the WWW. For our current research, we are following three directions: We will extend our OO class generator and make it and the class mappings available as a Web service ready to be included in standard build processes, so software developers can actually start using OO representations of RDF and OWL vocabularies. As the second direction of research, we will further improve our mapping mechanism, which facilitates runtime access to the actual linked data inside OO software. For these two directions of research, we also want to investigate how our approach can further simplify software development with dynamically-typed languages and languages that, e.g., use prototypes instead of class definitions. Helping software developers in complying to service policies and data licenses is our third direction of research. This will involve identifying those processes in our mapping framework relevant for policy evaluation and providing software developers with both simple tools for complying to standardized policies and licenses, and with developer hooks where they can add customized policy evaluation strategies.

To gain full benefit of RDF, OWL, and linked data, the WWW needs more data, hence more data sources, and as demand determines supply, more data consumers. By drastically simplifying the development of applications consuming and publishing linked data, we hope to contribute to shaping the future Web of data.

## References

[Berners-Lee (2006)] T. Berners-Lee: *Linked Data*. Online available at http://www.w3.org/DesignIssues/LinkedData.html, 2006.

[Bizer et al. (2008)] C. Bizer, T. Heath, K. Idehen, T. Berners-Lee: *Linked data on the web*. In Proc. of the 17th Int. Conf. on World Wide Web, pp. 1265-1266, ACM, Beijing, 2008.

[Brickley et al. (2004)] D. Brickley, R.V. Guha, B. McBride: *RDF Vocabulary Description Language 1.0: RDF*. Online available at http://www.w3.org/TR/rdf-schema/, Feb. 2004.

[Fowler and Rice (2003)] M. Fowler, D. Rice: *Patterns of Enterprise Application Architecture*, Addison-Wesley, Boston, 2003.

[Gamma et al. (1994)] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns. Elements of Reusable Object-Oriented Software*, Addison Wesley, Boston, 1994.

[Gosling et al. (2005)] J. Gosling, B. Joy, G. Steele, G. Bracha: *The Java Language Specification, Third Edition*, Prentice Hall, Englewood Cliffs, 2005.

[Manola and Miller (2004)] F. Manola, E. Miller: *RDF Primer. W3C Recommendation*. Online available at http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, Feb. 2004.

[Miller et al. (2008)] P. Miller, R. Styles, T. Heath: *Open Data Commons, a Licence for Open Data*. In Proceedings of the WWW2008 Workshop on Linked Data on the Web, pp. –, CEUR Workshop Proceedings, Beijing, 2008.

[Patel-Schneider et al. (2004)] P. F. Patel-Schneider, P. Hayes, I. Horrocks: *OWL Web Ontology LanguageSemantics and Abstract Syntax*. Online available at http://www.w3.org/TR/owl-semantics/, Feb. 2004.

[Quasthoff and Meinel (2008)] M. Quasthoff, C. Meinel: *Semantic Web Admission Free – Obtaining RDF and OWL Data from Application Source Code*. In Proc. of the 4th Int. Workshop on Semantic Web Enabled Software Engineering, pp. 17-25, Springer, Karlsruhe, 2008.

[Thompson et al. (2004)] H. Thompson, D. Beech, M. Maloney, N. Mendelsohn: *XML Schema Part 1: Structures Second Edition*. Online available at http://www.w3.org/TR/xmlschema-1/, Oct. 2004.

[Weitzner et al. (2006)] D. Weitzner, J. Hendler, T. Berners-Lee, D. Connolly: *Creating a policy-aware web: Discretionary,rule-based access for the world wide web.*. In Web and Information Security, pp. 1-31, IRM Press, Hershey, 2006.