Unsupervised Matching of Object Models and Ontologies Using Canonical Vocabulary

Matthias Quasthoff
Hasso Plattner Institute
Potsdam, Germany
matthias.quasthoff@hpi.uni-potsdam.de

Max Völkel
Institute AIFB
Karlsruhe, Germany
voelkel@aifb.uni-karlsruhe.de

Christoph Meinel
Hasso Plattner Institute
Potsdam, Germany
christoph.meinel@hpi.uni-potsdam.de

ABSTRACT

This paper presents a new method for publishing and consuming RDF data using object-oriented programming. We improve Object Triple Mapping (OTM) by separating (1) the transformation process between object-oriented data and RDF data from (2) explaining the transformation results using established Semantic Web vocabulary. To achieve this separation, we introduce a *canonical vocabulary* for object models. As a result, the Semantic Web expertise required to develop RDF-enabled applications is reduced.

Categories and Subject Descriptors

D.1 [**Programming Techniques**]: Object-oriented Programming; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*Semantic networks*

General Terms

Semantics.

Keywords

Semantic Web, Object-oriented programming, object triple mapping, schema matching.

1. INTRODUCTION

Modern Web sites are often build around huge data sets and occasionally contain references to data stored in other Web sites. For most commercial Web sites, these external data references are implemented using proprietary service interfaces using HTTP. Using a unified knowledge representation model to exchange information and organizing references are desirable instead. The Linked Data [2] community is working on using Semantic Web technologies such as the Resource Description Framework (RDF, [8]) in order to make the data stored in Web-based information systems available for other contexts. Researchers are working on best practices how to publish, announce and discover, and consume RDF data sets, how to ensure compliance regarding privacy policy and data licenses, and also how to actually develop RDF-enabled software. Developing RDF-enabled software requires a good amount of experience with different aspects of RDF data handling on top of software engineering skills. This paper proposes extensions to existing approaches on how to simplify developing Semantic Web applications using object-oriented programming. The primary focus of this paper is to lower the requirements to the software engineer's knowledge on specific RDF vocabulary.

An effective approach towards simplifying the development of Semantic Web-enabled software is Object Triple Mapping (OTM), and some amounts of practical work exists in that field [10, 11, 15]. The key principle of OTM is to let software developers design an object model reflecting the desired business logic, and let the OTM implementation handle the translation of RDF data from triple stores or from the World Wide Web to this object model. This approach has been proven to result in shorter development times, cleaner source code and reduced learning required from software engineers [13]. One important problem in Semantic Web software development that has been identified in the same experiment and is not directly addressed by OTM is researching and selecting RDF vocabulary appropriate for the intended use case. In this paper we present our research on how Object Triple Mapping can be used to further simplify developing RDF-enabled applications by incorporating OTM and existing approaches on ontology matching in order to support software developers in choosing RDF vocabulary appropriate for their software.

Our goal is to let software developers create RDF graphs from arbitrary object models without having to worry about which RDF vocabulary to use or how to correctly use it, and to let them restore their object models from these RDF graphs. Therefor we identified two independent processing steps in OTM, the separation of which adds clarity to the semantics of OTM. Creating RDF graphs useful for third parties, i. e. by using established RDF vocabulary, becomes optional. As our second contribution, we present an approach how to partially automate this optional yet desirable second step by applying known schema matching methods. The feasibility of our approach is demonstrated using an implementation prototype.

The rest of this paper is organized as follows. Related work we build upon is presented in Section 2. In Section 3 we present our adaptation to object triple mapping, which is the basis for the simplification we want to achieve, and show

Name	Language	Approach
ActiveRDF	Ruby	Naming
Elmo	Java	Annotation
OTMj	Java	Annotation
RDF2Java	Java	Code Generation
RDFReactor	Java	Code Generation
RDFAlchemy	Python	Annotation
So(m)mer	Java	Annotation
Surf RDF	Python	Naming

Table 1: Examples of existing object triple mapping implementations.

how to integrate object triple mapping and ontology matching. The first results achieved with our implementation prototype are presented in Section 4. Section 5 concludes the paper.

2. RELATED WORK

Mapping object-oriented concepts onto RDF vocabulary builds upon existing work on formalizing object-oriented programming, on knowledge representation using RDF, and on schema and ontology matching. Zhao et al. present a comprehensive survey on representing knowledge about software engineering-related concepts [17]. The EvoOnt software ontology model by Kiefer et al. [7] defines the concepts necessary to formally describe the structure of object-oriented software. This is relevant to describe how object-oriented class models can be mapped to RDF concepts. The PathLog language [4] formally describes access to object-oriented data, which is required to translate actual data held in object-oriented software to RDF and vice versa.

The idea to encapsulate operations on RDF data in an object-oriented *domain model* is inspired by Fowler and Rice's work on object-relational mapping [3]. This idea has been implemented a number of times, e. g. in So(m)mer [15], RDFReactor [16], ActiveRDF [10], and a number of other implementations as listed in Table 1. An overview on existing implementations is maintained at the Tripresso site[1]. Among the implementations, three main approaches for mapping object-oriented and RDF concepts can be identified.

Mapping by code generation. The first type of OTM implementation takes a RDF schema definition and generates object-oriented source class definitions, which contain the RDF data handling logic for the RDF properties relevant for the RDFS class the object-oriented class represents.

Mapping by annotation. A second possibility to implement OTM is to annotate existing object-oriented classes and specify the RDF class an OO class corresponds to, and for class members specify the RDF properties they correspond to.

Mapping by naming. A third option found in OTM implementations for dynamically typed languages is to construct URI identifying RDF concepts from the names of OO class members, i. e. by extracting a commonly known URI namespace prefix and the URI's local part from the name.

The third field of research involved is schema and ontology mapping. A detailed overview on schema matching is presented by Rahm and Bernstein [14]. A fundamental difference between schema matching algorithms can be whether they operate on schema level only or whether they also also use relations between instances from the schema. Due to the availability of standardized schemata and limited access to instance stored in a distributed way on the World Wide Web, our prototype performs schema matching on schema level using Similarity Flooding [9] and the Stable Marriage problem [5]. More information specific to ontology matching is presented on a dedicated Web site[2].

3. MAPPING BETWEEN OO AND RDF

The problem of simplifying the translation between object-oriented (OO) and RDF data can be split into two independent tasks. The first task is to reliably translate between object-oriented and RDF representations of information. It is not required for this step to take into account any external, publicly available RDF vocabulary. Instead, the generation of "ad-hoc" vocabulary linked to the object-oriented class model and expressed as meta-data to the primary data is acceptable, if not preferrable for traceability reasons. The second task is to link this ad-hoc vocabulary to established RDF vocabulary so that the data generated can be reused in other contexts, and other data expressed in more established vocabulary can as well be translated to a meaningful object model. By separating these two tasks, the latter becomes a traditional schema matching problem.

In the following sections we first introduce a formal representation of OTM. Then we describe how to express information from an object model in RDF using so-called *canonical vocabulary*, how to describe the links of this vocabulary to the underlying OO class model, and how to reconstruct an object model from RDF using this vocabulary description. Afterwards, we describe how to link canonical vocabulary to established, publicly available vocabulary, and how these links can be used to also consume external RDF data expressed using such established vocabulary only.

3.1 Formal basics of OTM

In this section, the formal representation of the RDF and OO data models introduced as well as a formal mapping between the two data models. The RDF data model has an established formal notation building upon the following concepts [8].

DEFINITION 1 (RDF DATA MODEL). *Let U be the set of* URI references, *B an infinite set of* blank nodes, *and L the set of* literals.

- $V := U \cup B \cup L$ *is the set of RDF* nodes,
- $R := (U \cup B) \times U \times V$ *is the set of all* triples *or* statements, *that is, arcs connecting two nodes being labelled with a URI,*
- *any $G \subseteq R$ is an* RDF graph.

PathLog [4] has been used by Oren et al. [10] to describe object-oriented access to data. In this paper we use a simplified version of PathLog, which does not differentiate between scalar and set-valued class members. Also, we use a simplified version of the *semantic structure* explained in [4].

DEFINITION 2 (OO DATA MODEL). *Let $\mathcal{N}$ be a set of names. The set of PathLog references $\mathcal{R}_{\mathcal{N}}$ is defined inductively as follows.*

[1]http://semanticweb.org/wiki/Tripresso

[2]http://ontologymatching.org/

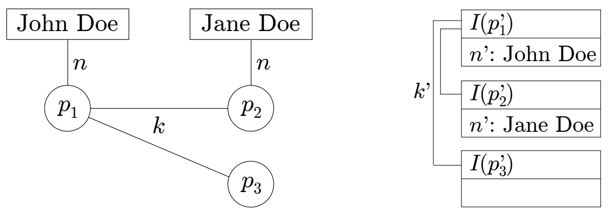

Figure 1: Representing social information in RDF (left) and OOP (right).

- *$n \in \mathcal{N}$ is a reference, also called a* simple reference.
- *for references t_0, t_1 and a simple reference s*
  - *$t_0.s$ is a reference, called a* path
  - *$t_0 : s$ and $t_0[s \to t_1]$ are references, called* molecules.

A semantic structure is a triple $(\mathcal{N}, O, I)$ such that

- *O is a set of objects and*
- *The* interpretation *$I : \mathcal{R}_\mathcal{N} \to 2^O$ relates references to objects.*

In the following, we show how RDF and PathLog can be used to express information in the respective data model.

Example 1 (comparison of RDF and OO). *Let p_1, p_2, $p_3 \in U$ denote three people, $n \in U$ be the URI* `foaf:name` *and $k \in U$ be the URI* `foaf:knows`[3]. *An RDF graph describing p_1, p_2 might look as follows (Fig. 1, left).*

$$G := \Big\{\langle p_1, n, \text{“John Doe”}\rangle,\ \langle p_1, k, p_2\rangle,\ \langle p_1, k, p_3\rangle,\ \langle p_2, n, \text{“Jane Doe”}\rangle\Big\}$$

Let furthermore p'_1, p'_2, $p'_3 \in \mathcal{N}$ be object names denoting three people and n', $k' \in \mathcal{N}$ fields labelled `name` *and* `knownPeople`. *The OO representation of G (Fig. 1, right) requires a semantic structure $(\mathcal{N}, O, I)$ such that*

$$\begin{aligned} I(p'_1.n') &= \{\text{“John Doe”}\} \\ I(p'_2.n') &= \{\text{“Jane Doe”}\} \\ I(p'_1.k') &= I(p'_2) \cup I(p'_3) \end{aligned}$$

The mapping of RDF data to OOP concepts as shown in Example 1 and vice versa has been formalized [13]. This formalization can be adopted to the concepts of PathLog as follows.

Definition 3 (Object triple mapping, OTM). *An* object triple mapping *for an RDF graph $G \subseteq (U \cup B) \times U \times V$ is a tuple $(\mathcal{N}, O, I, m_t, m_a)$, such that*

- *$(\mathcal{N}, O, I)$ is a semantic structure,*
- *the* vocabulary map *$m_t : F \to U$ maps a field names $F \subseteq \mathcal{N}$ to properties,*
- *the* instance map *$m_a : O \to U$ maps objects to resources,*

[3]The `foaf` prefix denotes the URI namespace http://xmlns.com/foaf/0.1/

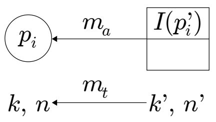

Figure 2: Mapping OO concepts (right) to RDF.

- *and the following holds for all $o \in O$, $n \in I^{-1}(\{s\})$, $f \in F$: The mapping is*
  - *complete:*
    $\forall\langle m_a(o), m_t(f), u\rangle \in G \exists o' \in I(n.f) : m_a(o') = u$,
    $\forall o' \in I(n.f) : \langle m_a(o), m_t(f), m_a(o')\rangle \in G$
  - *injective, i. e. $m_{a\,|I(n.f)}$ is injective.*

The idea of OTM is illustrated in Fig. 2 using the names from Example 1 and Definition 3. Besides such purely formal description of Object Triple Mapping, concrete implementations need to consider additional aspects such as object equivalence. Also, concrete implementations need to map the semantics of RDF, such as lists or reification, of RDF Schema, such as class hierarchies, and OWL, such as constraints on classes and properties, to the semantics of their supported programming language. The general feasibility of such features is discussed in [12]. Since these considerations mostly concern the runtime of a program, they are outside the scope of this paper.

3.2 Canoncial vocabulary

Existing OTM implementations either require a manual specification of the vocabulary map m_t by the developer (e. g., So(m)mer, Elmo and OTMj), or generate separate OO classes from RDF schema definitions that have m_t hardcoded (e. g., RDF2Java and RDFReactor), or implement m_t via naming conventions (e. g., ActiveRDF and Surf RDF). In contrast, we propose to separate the translation between OO and RDF representations and the addition of meaning to m_t by the introduction of a *canonical vocabulary* for object models as follows.

Definition 4 (Canonical vocabulary). *When a set of objects $\{o_1, \ldots, o_n\} \subseteq O$ is translated to RDF, the* canonical vocabulary *$V \subseteq U \cup B$ and the injective* canonical map *$m_t : \mathcal{N} \to V$ are constructed as follows.*

- *For each class $c \in \mathcal{N}$ of $o_i \in \{o_1, \ldots, o_n\}$*
  - *Let $c' \in V$ and $m_t : c \mapsto c'$, i. e. c' is the* canonical RDFS class *of c.*
  - *Add RDFS super-class relations to c'.*
  - *For each member variable $f \in \mathcal{N}$ of c*
    - *Let $f' \in V$ and $m_t : f \mapsto f'$, i. e. f' is the* canonical RDFS property *of f.*
    - *Add RDFS domain and range information to f'. Include the range class in the canonical vocabulary.*

During the construction of the canonical vocabulary, fixed mappings for known OO types (e. g. primitive types) on

```
# OO model
_:class1 a som:Class;
    som:name "org.example.Person";
    som:hasAttribute _:att1, _:att2.
_:att1 a som:Attribute; som:name "name".
_:att2 a som:Attribute; som:name "knows".

# RDF schema
_:class2 a rdfs:Class; rdfs:label "Person".
_:property1 a rdfs:Property; rdfs:label "name";
    rdfs:domain _:class2.
_:property2 a rdfs:Property; rdfs:label "knows";
    rdfs:domain _:class2; rdfs:range _:class2.

# Vocabulary map
_:class1 otm:mapsTo _:class2.
_:att1 otm:mapsTo _:property1.
_:att2 otm:mapsTo _:property2.
```

Figure 3: N3 representation of the canonical vocabulary derived in Example 2.

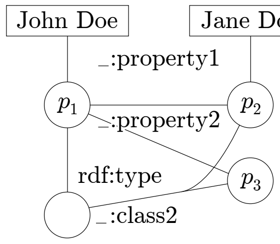

Figure 4: RDF created from the object model and canonical vocabulary in Example 2.

primitive types from XML schema can be defined. Also, existing OTM implementations often do not map collection and array types, but rather translate object-oriented collections of objects to a set of statements about the elements of the collection.

Using canonical vocabulary, object-oriented data can be translated to RDF and vice versa at the cost of complete lack of meaning of the resulting RDF data to other applications. To not rely on URI naming conventions and to allow the use of blank nodes for the canonical vocabulary, the vocabulary V and m_t need to be described as meta-data to the primary data. This in turn requires to also describe the relevant concepts from V, e. g. using the EvoOnt software ontology model [7]. The description of m_t requires an additional RDF predicate to relate EvoOnt entities, i. e., classes and fields, to RDFS concepts.

Example 2 (Canonical vocabulary). *Let $p_1, p_2 \in O$ be two objects representing acquainted people as illustrated in Fig. 1 (left). The corresponding canonical vocabulary is described in RDF (N3) in Fig. 3*[4]*. Using the canonical vocabulary, the information represented by p_1, p_2 can now be expressed in RDF as displayed in Fig. 4.*

Note that the description of the object-oriented class model using the EvoOnt vocabulary is refers to a specific class model created in a specific programming language. Canonical vocabulary created from different programming languages will result in separate descriptions of the class model. Although the original object model in Example 2 can easily be re-created from the RDF data and the canonical vocabulary, the RDF data produced is utterly useless for consumers unaware of the OO class model. Hence in the next section, we discuss how link the canonical vocabulary to established RDF schemata.

[4]The som prefix denotes the URI namespace http://www.ifi.uzh.ch/ddis/evoont/2008/11/som#

3.3 Linking to established vocabulary

The canonical vocabulary needs to be linked to existing vocabulary with the means of RDFS or OWL. Using `owl:sameAs` links has several disadvantages, such as, e. g. meta-data attached to the canonical vocabulary would clutter the description of the established vocabulary being linked to. Using `owl:equivalentClass` would improve the situation, but still potential semantic differences between the canonical vocabulary and the established vocabulary do not let this approach seem appropriate.

The intended meaning of the object model in Example 2—that resources representing a former person object p_i are of type `foaf:Person`—can be conveyed using `rdfs:subClassOf` statements, and the properties of the canonical vocabulary can be linked using `rdfs:subPropertyOf` statements. Using RDFS reflects a specific issue found in OTM, but also in similar approaches like mappings from relational databases. RDF data produced from structured data like models using OTM will be more structured than RDF graphs from arbitrary sources: Due to constraints in the object model, statements with a certain predicate might always exist in the generated RDF graph, or never, depending on the actual OO class model. In contrast, if an RDF graph from another data source contains information about a resource r of type `foaf:Person`, we of course cannot assume that r is of the type `_:class2` in the canonical vocabulary from Example 2 or that statements with certain predicates about r do or do not exist.

It becomes clear that we cannot safely apply OTM directly to RDF data from other data sources. Instead, consuming linked data from other sources relies on *assuming* that a resource of a certain type t might also be of canonical class t', if this is an RDFS sub-class of t. Although this approach might not seem convincing at first sight, it is correct as the structure of external data to be consumed cannot be assumed to adhere to explicit or implicit constraints of the object-oriented piece of software operating on the object-model. The choice of t' can be easy, and will be in many cases, when, e. g. t' is the only known sub-class of t and no further constraints are known. The choice of t' will become arbitrarily hard if more sub-classes of t are known, and constraints about these sub-classes are formulated, e. g. using OWL or exist with regard to different object-oriented classes these sub-classes are mapped to.

4. IMPLEMENTING SCHEMA MATCHING

By strictly separating the translation from object-oriented data to RDF and vice versa from the mapping of the canonical vocabulary introduced in Section 3, this section can clearly focus on how to implement schema matching without having to deal with the semantics of object-oriented programming or OTM. First, we discuss the selection of appropriate vocabulary to map against. Then we present our implementation and lessons learned from it.

4.1 Choice of established vocabulary

Before schema matching can be performed on the canonical vocabulary and existing RDF vocabulary, some existing vocabulary to map to needs to be identified. The vocabulary to consider can be obtained from the World Wide Web, e. g., by querying a Semantic Web search engine for keywords extracted from the canonical vocabulary. By using a search engine to find potentially relevant schema definitions, the OTM implementation will benefit from the popularity ranking, such that popular vocabulary is more likely to be considered for schema matching than rather unknown special-purpose vocabulary.

It has been observed that not all popular schema definitions were available via HTTP at all time, and that this limited availability can reduce the productivity of software engineers [12]. For this reason, schema definitions should be searched and retrieved from the Web only when necessary, and should be cached locally. To focus on the schema matching part only, we did not implement this schema lookup so far, but operate on a fixed sub-set of the schemata investigated in [12].

4.2 Schema matching algorithm

The algorithm we implemented to prove our concepts consists of three steps.

1. Initialization of similarities between all OO and RDF concepts, based on the edit distance of string representations of the concepts,
2. Applying similarity flooding [9] to the RDF graph representing the canonical vocabulary, and to the RDF graph containing all vocabulary to map to,
3. Solving the Stable Marriage problem [5] for the OO and RDF concepts.

In the first step of the computation we use the RDFS labels of classes and properties and obtain the Levenshtein distance of these labels. There has been some work on finding more elaborate string representations, e. g. in the implementation of RDFReactor [16], which considers RDFS labels from different languages and also considers local parts of URI identifying the concepts. Although analyzing URI is not encouraged theoretically, it has shown good results in practice.

The choice of similarity flooding is the most influential part of our implementation. The algorithm iteratively propagates the estimated similarity between nodes from the to graphs to their neighbour nodes until some level of convergence is achieved. Due to its design, the algorithm is unaware of what is a class and what is a property, but these are kept separate due to the bi-partite nature of the graphs (classes are only connected to properties and vice-versa.) Although the algorith gives good results most of the time, bad similarity initializations in the previous step can let the algorithm converge in wrong mappings. Assigning fixed similarity for known mappings, i. e. `rdf:Literal` to strings or mappings interactively confirmed by the user, dramatically improved the quality of the mappings produced by the algorithm. An advantage of similarity flooding is its support for interactivity, i. e. after assigning fixed similarities by user input, new iterations of the algorithm can be computed.

Solving Stable Marriage does not have such large impact on the quality of the mapping. It is however a characteristic of the solution that only one-to-one mappings are returned. Consequently, bean-style accessors to one and the same member variable (such as `setName(String)` and `getName()`) need to be treated as a single property when creating the canonical vocabulary.

4.3 Evaluation

To evaluate the feasibility of our approach, we asked eight CS graduate students to come up with an object model meeting the following requirements.

- One class should describe human beings, and contain a name (either first and last or full name), a list of acquaintances, and two more attributes to be made up by the participant.
- The class describing human beings should be associated to a second class like a product, a creative work or an event, which should also be described using some made-up attributes.

These requirements were expressed in German since participants were likely to deliver an English object model and we did not want to suggest entity names. Seven object models were indeed expressed in English language, the eighth model was described in German. We applied the schema matching algorithm described in the previous section to the individual object models and the union of the Friend of a Friend (FOAF) ontology and DCMI Metadata Terms[5].

Our hypothesis is that the class describing humans will be matched to `foaf:Person` with high confidence due to the structural information conveyed by the mandatory “has friend” and “has name” attributes, and that the similarity values computed by Similarity Flooding will help separating correct mappings from incorrectly guessed mappings.

Four of the data participants called the human class “Person”, three participants called it “Human”, and one had the German name “Mensch”. Similarity Flooding does not always converge to the same results, but still all “Person” classes were always mapped to `foaf:Person`. The “Mensch” class and one of the “Human” classes were also always mapped correctly. A second “Human” class was alternately mapped to `foaf:Person` or `wgs84:SpatialThing`[6], the third “Human” class was alternately mapped to `foaf:Person` and `foaf:Document`, and the remaining “Human” class was always mapped to `wgs84:SpatialThing`. As Similarity Flooding highly relies on the structure of the schemata, attributes like “first name”, “last name” or “name” were correctly mapped on the corresponding FOAF predicates whenever the person class was mapped correctly. In a second round we renamed the human class in all object models to “Person”, which resulted in correct mappings at all time. The mapping of the second class provided by the participants (e. g., “ConcertTicket”, “House”, “Book”, “Pet” etc.) was always stable, but never correct or usable. The stability of this part of the mapping can be partially explained as, e. g., the “ConcertTicket” was modelled to have an “eventName”, which was initialized similar to `foaf:accountName`. The similarity of these names has then been propagated to the classes “ConcertTicket” and `foaf:OnlineAccount`.

[5]http://dublincore.org/documents/dcmi-terms/

[6]The `wgs84` prefix denotes the URI namespace http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing.

Regarding the first part of our hypothesis, we can say that a small overlap in schema structure and in the naming of concepts already leads to a a promising mapping of central concepts. Even if names differ a lot, chances are that matching concepts will be identified correctly. One approach to further improve our implementation could include some simple natural language processing to identify, e. g., a possible relation between a class called "Human" and a "Person" class. Also, most class models provided by the participants featured a very simple structure, consisting only of two classes. More realistic class models with a higher number of classes might also lead to further improved results. Regarding the second part of our hypothesis, the similarity values computed by Similarity Flooding have been found unsuitable for applying thresholds, but are rather to be interpreted as relative similarities, just that a best match, e. g. by applying Stable Marriage, can be constructed [9].

5. CONCLUSION AND OUTLOOK

In this paper we presented an extension to Object Triple Mapping that further simplifies the development of Semantic Web applications using object-oriented programming. We proposed to separate the transformation between the two knowledge representations and the mapping of the RDF data generated to established RDF vocabulary. We introduced canonical vocabulary to represent object models in RDF and showed how to link this canonical vocabulary to established RDF schema definitions from the World Wide Web using existing schema matching algorithms. Finally, we presented our implementation of the proposed method using an existing schema matching algorithm and discussed the first results obtained using the implementation.

We plan further research in two directions. First, the canonical vocabulary and its links to established vocabulary are meta-data to the RDF data produced by OTM implementations. The quality of information inferred from the data produced and these links directly depends on the quality of the schema matching results. Luckily, some schema matching algorithms, such as Similarity Flooding, which we used, also return metrics on the quality of the mapping achieved. Publishing these quality attributes along with the mapping will lead to increased transparency for data on the Web and can be used in the context of data provenance tracking [6] and trust management [1]. Since we intend our work being valuable for a variety of use cases, and also not be tied to specific programming languages, the second direction of research will be to evaluate practictal implementations of our approach with different programming languages and also with different schema matching algorithms, in order to provide maximum support for a large number of software developers.

Simplifying the development of Semantic Web applications is a necessity in order to get non-academic software engineers start using Semantic Web technologies in their projects. Experiments have shown that Object Triple Mapping does simplify such software development, but also that OTM still has issues to be solved [13]. One difficulty for software developers lies in the task of manually researching RDF vocabulary to use and making sure to use it correctly. In this paper we showed how to reduce, and potentially eliminate, these manual efforts, so that software developers can start using Semantic Web technologies immediately. When implementing object-oriented software, they can have a usable mapping of their object model to commonly used RDF vocabulary without any extra effort. In combination with emerging tools supporting the development of complete Linked Data applications, such as OTM-based RDFa plugins for Web application frameworks[7], our results make Semantic Web software technologies an easy-to-use option for a broad range of software engineering tasks.

6. REFERENCES

[1] R. Alnemr, J. Bross, and C. Meinel. Constructing a context-aware service-oriented reputation model using attention allocation points. In *IEEE SCC*, pages 451–457. IEEE Computer Society, 2009.

[2] T. Berners-Lee. Linked data. http://www.w3.org/DesignIssues/LinkedData.html, 2006.

[3] M. Fowler and D. Rice. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003.

[4] J. Frohn, G. Lausen, and H. Uphoff. Access to objects by path expressions and rules. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *VLDB*, pages 273–284. Morgan Kaufmann, 1994.

[5] D. Gusfield and R. Irving. *The stable marriage problem: structure and algorithms*. MIT Press Cambridge, MA, 1989.

[6] O. Hartig and J. Zhao. Using web data provenance for quality assessment. In J. Freire, P. Missier, and S. S. Sahoo, editors, *SWPM*, volume 526 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[7] C. Kiefer, A. Bernstein, and J. Tappolet. Analyzing software with isparql. *Proc. of the 3rd Int. Ws. on Semantic Web Enabled Software Engineering*, 2007.

[8] F. Manola and E. Miller. Rdf primer. w3c recommendation 10 february 2004. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, 2004.

[9] S. Melnik, H. Garcia-Molina, E. Rahm, et al. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the International Conference on Data Engineering*, pages 117–128. IEEE Computer Society Press; 1998, 2002.

[10] E. Oren, B. Heitmann, and S. Decker. Activerdf: Embedding semantic web data into object-oriented languages. *J. Web Sem.*, 6(3):191–202, 2008.

[11] M. Quasthoff and C. Meinel. Design patterns for object triple mapping. In *Proc. of IEEE SCC 2009*, 2009.

[12] M. Quasthoff, H. Sack, and C. Meinel. Can software developers use linked data vocabulary? In *Proc. of I-Semantics '09*, 2009.

[13] M. Quasthoff, H. Sack, and C. Meinel. How to simplify building semantic web applications. In *Proc. of the 5th International Workshop on Semantic Web Enabled Software Engineering*. CEUR-WS.org, 2009.

[14] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[7]e. g., Grails RDFa, http://grails.org/plugin/rdfa

[15] H. Story. Java annotations and the semantic web. http://blogs.sun.com/bblfish/entry/java_annotations_the_semantic_web, 2005.

[16] M. Völkel. Rdfreactor – from ontologies to programatic data access. In *Proc. of the Jena User Conference 2006*. HP Bristol, Mai 2006.

[17] Y. Zhao, J. Dong, and T. Peng. Ontology classification for semantic-web-based software engineering. *IEEE Transactions on Services Computing*, 2:303–317, 2009.