published as: Matthias Quasthoff, Christoph Meinel: Semantic Web Admission Free – Obtaining RDF and OWL Data from Application Source Code;
In Proceedings of the 4th International Workshop on Semantic Web Enabled Software Engineering (SWESE 2008); Karlsruhe, Germany, October 2008

Semantic Web Admission Free – Obtaining RDF and OWL Data from Application Source Code

Matthias Quasthoff, Christoph Meinel
{matthias.quasthoff, meinel}@hpi.uni-potsdam.de

Abstract. Semantic Web standards have evolved over more than a decade now. Though semantic technologies are predicted to become ubiquitous, they are still far from that. One reason for this is that not all layers of the so-called Semantic Web layer cake are easily accessible for software developers. In this paper, we present our programming interface for the Java programming language, providing easily maintainable RDF/OWL interfaces to existing applications taken directly from application source code. Also, we propose a Semantic Web programming interface taking care of the generation, consumption, and distribution of Semantic Web data adhering to users' security and privacy demands.

1 Introduction

Representing and exchanging digital information with the help of Semantic Web technologies such as the Resource Description Framework (RDF) and the Web Ontology Language (OWL, [9]), is necessary for building decentralized, standardized applications. These technologies help developing several areas and paradigms in information technologies, such as the Service-Oriented Architectures (SOA), information security, and the World Wide Web (WWW) and bringing them closer together.

There do exist commercial and academic tools like Protégé[1] [10] supporting the design of RDF schemas and OWL ontologies, and APIs like Jena[2] and OWL-API[3] for generating and consuming Semantic Web documents in custom software projects. However, usage of these tools and APIs has hardly moved beyond the Semantic Web community. Thus, for the Semantic Web APIs there is only little documentation of best practices, common programming mistakes and solutions, and other useful

1 http://protege.stanford.edu/
2 http://jena.sourceforge.net/
3 http://owlapi.sourceforge.net/

resources just emerging in dedicated mailing lists and chat rooms. This makes it even harder for developers unexperienced with Semantic Web technologies to enrich their software products with Semantic Web features, as very few of all the obstacles a developer can stumble upon have already been discussed in these mailing lists. Up to now, it takes huge manual implementation efforts to provide some custom implementation with a Semantic Web interface for data generation, consumption and distribution. For a given object-oriented implementation, the mapping of each and every class has to be implemented by hand. This work can be compared to manually mapping class definitions in object-oriented programming languages to relational database schemas by hard-coded SQL statements, leading to programming errors and security leaks such as the famous SQL injection [2]. Frameworks like RDFReactor[4] help software developers by by generating Java source code from ontologies. However, this code generation imposes on the software development process; subsequent modifications of the code generated are at risk of being lost in case of code regeneration.

This paper aims at making the development of semantic applications much simpler. We present a solution allowing application developers to easily map their Java classes to corresponding OWL concepts. They can do so without too much background knowledge about the specifics of RDF and OWL, and are also saved from the Sisyphean labour of manually keeping mappings in sync with their class definitions. The mapping is achieved by using the Java annotations [5], a metaprogramming feature also found in other languages and runtime environments. Our implementation allows to map arbitrary Java classes and interfaces to OWL concepts and has already been employed in a social network application testing new access control mechanisms on user-generated content with the help of Semantic Web rules [12].

This paper is organized as follows. Section 2 gives a motivation for our contribution in the Web Service world and an overview on related work. In Section 3, we present the overall architecture proposed for developing Semantic Web-enabled software. In Section 4 we present the model of our implementation and the resulting API, which is also available for download. Section 5 presents our experiences while integrating our solution into an existing Social Network application. Finally, in Section 5 we conclude our work and give an outlook on our future work in this area.

2 Motivation and related work

Semantic descriptions of data in information systems apply to content, as e.g. published on the WWW, and also to Web Services. Using the Web Ontology Language for Web Services (OWL-S, [11]), a service's profile (what it does), its model (how it works), and its grounding (how to interact with it), can be described in OWL. However, it is still up to the person maintaining Web Service descriptions to provide a semantic description of the service. If the underlying implementation of the Web Service processes semantic data, the semantic description of the Web Service should be derived from the semantic information in the source code. Our implementation provides a first step in this direction, by deriving semantic descriptions of objects used in an implementation.

4 http://semanticweb.org/wiki/RDFReactor

Our implementation works in the fashion of the Java Persistence API (JPA, [4]), which provides mappings of Java class definitions to relational database schemas. These mappings, e.g. column and identifier names, auto-increment values and others, are influenced by Java annotations [5]. This configuration style reduces the effort of keeping extra configuration files besides source code files. However, no such support exist for developing semantic software, hence data types need to be converted manually from an application's internal representation to semantic formats. It is probably these redundant software development tasks preventing application developers from just giving Semantic Web technologies a try in their applications or services. Still, mapping application source code to database formats differs fundamentally from mapping data to semantic formats, as databases are used internally, where semantic formats are intended for publication, hence are subject to access control and trust considerations.

The two most popular free frameworks for incorporating OWL in Java software are the Jena and OWL API frameworks mentioned before. Although they support many aspects of Semantic Web technologies including reasoning, their support for software development processes is poor. Up to now, translating a Java object to its OWL representation requires at least one line of code for each referenced individual, each class, and each property assertion, plus some fixed initialization overhead. Also, whenever a Java class definition is modified, i.e. properties are introduced or removed, the corresponding translation mechanism has to be adapted. We see that application developers should not be expected to be Semantic Web professionals, and should be offered an intuitive way of providing their implementations with semantic interfaces on a conceptual level without having to worry about how to translate single individuals.

3 Architecture overview

A software development architecture supporting developers in enriching their existing or new software projects with semantic functionality will consist of three major function units:

1. Translation mechanism between application-internal data formats and Semantic Web formats,
2. Semantic Web connectivity; i.e. facilities for discovering, consuming, and publishing semantic data,
3. Mechanisms verifying trustworthiness of data to be consumed, and performing access control on published data being accessed by other parties.

Achieving this functionality requires building upon metaprogramming features of modern programming languages such as Java or the Microsoft .NET platform. The architecture resulting from this approach is depicted in Fig. 1. By using their reflection facilities, redundant work to software developers can be avoided. Providing software developers with such a library can happen step-by-step as indicated by the aforementioned function units. In the course of this paper, we present our implementation the translation mechanism in step 1. With regard to fields of application in the

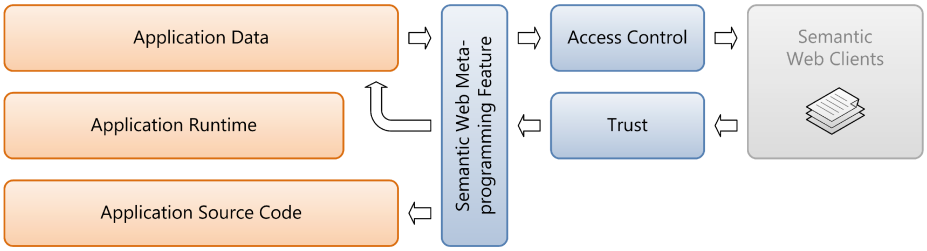

Fig 1: Existing applications with source code annotations (left) are enriched with input and output facilities for semantic formats by the programming library (center vertical). Input and output to the application is protected with the help of access control and trust policies (right).

Social Web, the security implementation in step 3 should allow for flexible specification of policies (cf. [7]), to allow for an implementation that can be understood by regular users without special background knowledge in IT security or Semantic Web.

4 Semantic Web metaprogramming

In this section, we will present our design and implementation of a Java-to-OWL translation mechanism. Examples of Semantic Web concepts will be taken from the *Friend of a Friend* (FOAF, [3]) ontology. Java classes, interfaces, methods, and fields can often be directly related to OWL classes, object and data properties. The objects constructed from Java classes then correspond to OWL individuals. To construct these OWL individuals from Java objects, the underlying class definition has to contain information about the mapping. Currently, we support the construction of OWL individuals from Java objects and vice-versa.

In a given implementation, not every class, interface, field, or method necessarily needs to have an OWL equivalent. A Java class can e.g. contain internal information such as system-wide identifiers without a semantic representation. But still, the class can contain other information that do have a semantic representation. The semantic representation of an object belonging to this class should then contain all the relevant information that can be extracted. When translating a Java object to an OWL individual, the object's class and its super classes and interfaces need to be checked whether they can contain semantic information.

Annotated elements

Each Java class, interface, field, and method being mapped to OWL equivalents require a URI referencing the respective class or property. In this section, we describe how the mapping of the different entities is controlled using Java annotations.

Classes and interfaces. As mentioned before, classes can be mapped to an OWL class (Fig. 2). For the developer's convenience, and to eliminate one likely source of typing errors, the *Semantics* annotation allows for separately specifying a URI base

for Java classes and interfaces, which will then be used for all relative URIs used in the same class or interface, or for the class or interface itself (Fig. 3, line 1). When the relative URI for a OWL class or property is equal to the corresponding Java name, it can be omitted (Fig. 3, line 3, Fig. 4, line 1). Keep in mind as by far not all fields and methods of Java classes do have OWL equivalents, properties still have to be annotated in order to be mapped, even though the URI can be omitted (Fig. 3, line 3). The situation is slightly different with Java interfaces. If an interface is annotated to have an OWL equivalent, we assume that all getter-like methods (i.e. non-void return type, no arguments) have an OWL equivalent of the same name, regardless whether there is a *Semantics* annotation (Fig. 4, line 3). Getter-like method in interfaces can be prevented from being mapped to an OWL property by accompanying the annotation with a parameter *include=false*.

```
1  @Semantics("http://xmlns.com/foaf/0.1/Person")
2  class User {
3     @Semantics("http://xmlns.com/foaf/0.1/name")
4     String name;
5  }
```

Fig 2: Simple mapping of Java class and field names to semantic concepts.

```
1  @Semantics(base="http://xmlns.com/foaf/0.1/", value="Person")
2  class User {
3     @Semantics        // guess "name" as relative
4     String name;      // property URI automatically
5  }
```

Fig 3: Convenience mechanisms for software developers are provided by inheriting URI bases from class definitions to methods and fields and by using Java method and field names to construct the URI of corresponding Semantic Web properties.

```
1  @Semantics(base="http://xmlns.com/foaf/0.1/")
2  interface Person {      // "Person" is the relative class URI
3     String getName();    // "name" is the relative property URI
4  }
5
6  class User implements Person {
3     ...                  // inherits complete OWL mapping
4  }
```

Fig 4: When mapping Java interfaces to Semantic Web concepts, methods and fields are automatically mapped without need for further specification.

```
1  @Semantics(base="http://xmlns.com/foaf/0.1/")
2  interface Person {
3     String getName();
4     @Semantics("knows")
5     Collection<Person> getFriends();
6     // equivalent to Collection<Person> knows(),
7     // which wouldn't require @Semantics annotation
8  }
```

Fig 5: Java methods and fields containing collection types can be chosen to be translated to a single property assertion about a RDF list or to multiple property assertions about individuals.

```
1  // given OWLOntology ont, OWLOntologyManager man
2  Person person = new User(...);
3  OWLPersistence p = new OWLPersistence(ont, man);
4  OWLIndividual foaf = p.persist(person);
```

Fig 6: Creating Semantic Web data from application data in two lines of code.

Methods and fields. As can be seen in Fig. 4, whenever the name of a method begins with "get" or "is", the de-capitalized remainder of the method name is taken as the property name, if it is not explicitly specified using an annotation. One more thing that can be specified for methods and fields is how property values that implement the *java.lang.Iterable* interface (typically collections, lists and the like) will be represented in OWL. By default, for each member of the *Iterable* a separate property will be added to the containing individual (Fig. 5, line 5). In order to explicitly render a RDF list, the *Semantics* annotation had to carry a parameter *lists=AS_LIST*, where *lists=INDIVIDUALLY* is the default.

OWL individual generation

Our implementation builds upon the OWL API mentioned before. With minor modifications only, it will support Jena as well. The generation of OWL individuals from Java objects requires an *OWLOntology* and an *OWLOntologyManager* from OWL API. Those have to be used to generate our *OWLPersistence* manager. When generating the OWL individuals and their properties, *OWLPersistence* adds these to the given ontology. As a consequence, whenever one Java object is to be repeatedly turned into OWL individuals in different contexts, i.e. in different ontologies, for each ontology separate *OWLPersistence* instances have to be used.

When generating a new OWL individual (Fig. 6, line 4), the class of the object, all super-classes and interfaces, and all their methods and fields are checked for *Semantics* annotations. All properties discovered are then recursively translated using depth-first search, until no more object properties are to be translated. Once an object has been translated, the resulting OWL individual is stored. Whenever the translation mechanism encounters the same object which has been translated before the previous translation is returned. The translation mechanism is illustrated in Fig. 7.

```
1  persist(Object object):
2     if object has been translated before, return translation
3     determine object's uri
4     create new individual(uri)
5     for each class to which object belongs:
6        obtain URI base from class' @Semantics(base="...") annotation
7        for each field and "get*" method with @Semantics annotation:
8           set property = URI base + method/field name
9           set value = method.invoke(object) or field.get(object)
10          add triple (individual, property, persist(value))
11    return individual
```

Fig 7: Pseudo code illustrating translation from Java to OWL. Java reflection is used to invoke an object's getter methods and to read the object's fields (line 9).

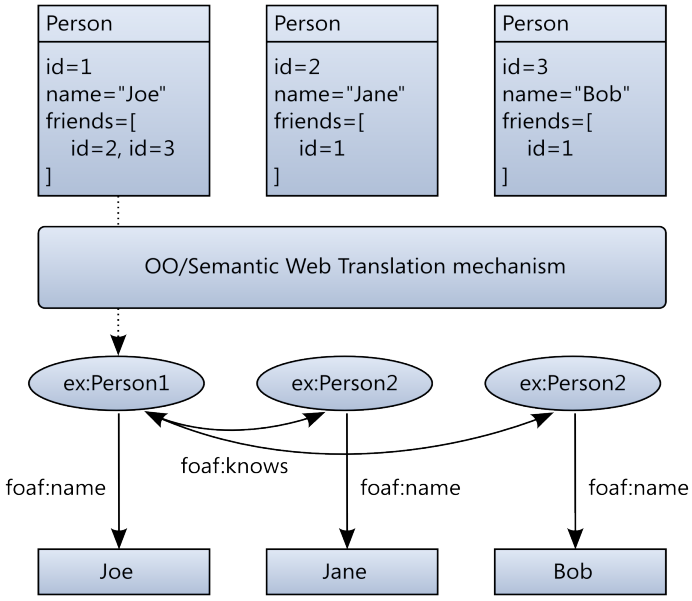

Fig 8: Translation of three objects with mutual references in the "friends" field into three individuals with corresponding properties from the FOAF ontology.

Constraining individual generation

Using plain depth-first traversal only, connected components are always translated into OWL as a whole. In practice, for reasoning performance, data protection and other reasons, often the translation of smaller, well-defined parts only is desired. In our implementation, this is achieved using constraints. Constraints can apply to individuals, i.e. determine on a per-individual basis whether a Java object should be added to the given ontology. These *IndividualConstraints* can e.g. be used to restrict the *foaf:Person* individuals visible in one's social network to direct contacts. Another

type of constraint are *PropertyConstraints,* used to exclude properties, such as other *foaf:Persons*' personal information, from being translated to OWL without completely excluding the affected OWL individuals.

5 Observations and conclusion

The translation mechanism presented in this paper has been used to implement the tag-based access control model [12]. This access control model focuses on user annotations in Social Tagging Systems [8] and involves rules in the Semantic Web Rule Language (SWRL). The Java-to-OWL translation mechanism is used to translate user information from a social networking prototype [12] to concepts from the FOAF ontology. The code examples in Figures 2 to 8 originate from this social network implementation. The application has been implemented in the Groovy dynamic language[5] for the Java Virtual Machine. Achieving the translation of information generated and stored in the social network did not take more than mapping of programming data types to concepts from the Semantic Web; the deeper structure of application and Semantic Web data does not require any further manual, error-prone mapping and maintenance.

At present we implemented the first of the three steps mentioned in Section 3. Since the other two still remain to be implemented, only information explicitly stored or imported in this single social network application is available for semantic representation. Our vision is now to be able to automatically use all information accessible on the Semantic Web to enhance the user experience in modern Web applications, ensuring the use of trusted high-quality information only and obeying users' privacy and data protection concerns using flexible authorization mechanisms.

Besides the plain implementation of mappings between the object-oriented and the semantic world, research on fundamental differences of closed-world and open-world systems is needed. Information can be processed by traditional implementations, by processing formats like XML with XML stylesheet transformations (XSLT), or by reasoning and querying semantic data. Which programming paradigm is suitable for which task or problem domain? Research is also necessary on how to deal with incomplete or inconsistent data from multiple sources within one instance of a software using our translation mechanism. Consider e.g. processing the Java representation of a FOAF person that has been generated from an RDF graph by our translation mechanism. If we change the name of the person by setting the corresponding Java field, do we want this update to be propagated to the originating RDF graph? Or do we want the Java application to form its own graph or context that we can use to override data from external sources? To do this, we need a trust hierarchy of contexts that allows us to always look for data in the "best" context first, and fall back to other contexts if we don't find the information we are looking for.

By making the Semantic Web accessible to software developers, we prepare the integration of semantic data formats and interfaces into computer software. The current status of the implementation is available for download. Taking semantic

5 http://groovy.codehaus.org/

implementations further now requires network effects. Software developers need incentives and support through technology making them prefer semantic formats and technologies for data exchange over proprietary interfaces. Increased use of semantic technologies will in turn give a better understanding of the support needed to develop semantic software, triggering another iteration of developing supporting tools and obtaining more semantic software.

References

1 R. Alnemr, C. Meinel: Getting more from Reputation Systems: A Context-aware Reputation Framework based on Trust Centers and Agent Lists. In Proc. of the 3rd Int. Multi-Conference on Computing in the Global Information Technology, Athens, 2008 (to appear).

2 Stephen W. Boyd, Angelos D. Keromytis: SQLrand: Preventing SQL Injection Attacks. In Applied Cryptography and Network Security, Springer 2004.

3 Dan Brickley, Libby Miller: FOAF Vocabulary Specification 0.91, FOAF-Project, May 2007, available at http://xmlns.com/foaf/spec/20070524.html

4 Linda DeMichiel, Michael Keith: Java Persistence API. 2006 JavaOne Conference. To be downloaded at http://www.agilejava.com/downloads/TS-3395.pdf

5 James Gosling, Bill Joy, Guy Steele and Gilad Bracha: The Java Language Specification, Third Edition. Prentice Hall, June 2005.

6 Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean: SWRL: A semantic web rule language combining owl and ruleml, W3C Member Submission, 21 May 2004. Available at http://www.w3.org/Submission/SWRL/

7 L. Kagal, T. W. Finin, A. Joshi: A Policy Based Approach to Security for the Semantic Web. In Proceedings of the Second International Semantic Web Conference, Springer, 2003

8 A. Mathes, "Folksonomies – Cooperative Classification and Communication Through Shared Metadata", Graduate School of Library and Information Science, University of Illinois Urbana-Champaign, 2004.

9 D. L. McGuinness and Frank van Harmelen (Eds.): OWL Web Ontology Language Overview, W3C-Recommendation, Feb. 2004, http://www.w3.org/TR/owl-features/

10 Noy, N. F. & McGuinness, D. L. Ontology Development 101: A Guide to Creating Your First Ontology. Knowledge Systems Laboratory, March, 2001.

11 OWL-S Coalition. OWL-S 1.0 Release. At http://www.daml.org/services/owl-s/1.0/

12 M. Quasthoff, H. Sack, C. Meinel: Who Reads and Writes the Social Web? A Security Architecture for Web 2.0 Applications. To appear in Proceedings of the Third International Conference on Internet and Web Applications and Services, Athens, 2008.