Writing the Semantic Web with Java *

http://semweb4j.ontoware.org

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

ABSTRACT

To build semantic web applications, developers must master three things at the same time: Their programming language, the semantic web languages (RDF, RDFS and OWL), and the web protocols (HTTP). This paper presents a framework, semweb4j that turns Java developers into semantic web developers without *requiring* them to learn RDF, RDFS, HTTP or Servlets. We present a triple store abstraction layer (RDF_2Go), an "RDFS as Java" tool (RDFReactor), and a tool for RESTful web service creation without exposing Servlets (jREST). Finally we briefly show in two examples how these frameworks can be used together (branded as semweb4j) in real-world applications.

1. INTRODUCTION

The Semantic Web vision combines two core elements: computer-tractable *semantics* [3] and the world wide *web* [1]. These two elements, *semantics* and *web*, both contain ideas for interoperability that go beyond traditional programming. With computer-tractable semantics, sophisticated and expressive data models can be re-used across programming languages. The internet allows two machines to exchange data across locations at very low costs. Taken together, the *semantic web* enables computers to exchange data and their semantics without being bound to any programming language or geographical location.

For a developer, the realisation of this vision involves a lot of programming effort. Each programming language offers a different type system, some offer static type checking, some languages defer the check until runtime.

The semantic web has a common data model (RDF) and a way to express the semantics of it unambiguously (RDFS and OWL). Semantic web applications produce and consume semantic content—data structures annotated with their semantics.

In order to write semantic web applications, two gaps must be closed: From the specific type-system of a programming language to the generic semantic web representation languages (RDF, RDFS, OWL) and from method calls to the network protocol of the semantic web (HTTP).

For this paper, we analyse and show how Java can be used to write the semantic web. Especially, we try to bridge the gap with frameworks that do not require much knowledge about semantic web concepts and technologies. To summarise, we try to turn ordinary Java developers into semantic web developers – without telling them.

In the remainder of this paper we will first talk about the semantic representation gap in Sec. 2 and then about the web protocol gap in Sec. 3. In Sec. 4 we will present some real-world applications realised with semweb4jFinally, in Sec. 5, we will summarise the main points.

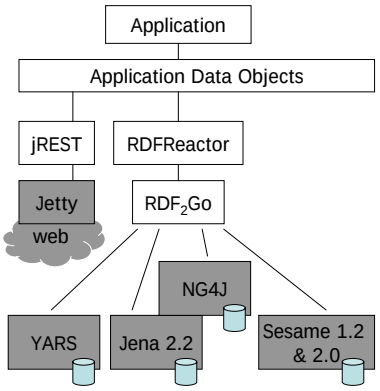

Figure 1: Building Applications with semweb4j

2. THE SEMANTIC IN SEMANTIC WEB

Today's data is mostly stored in relational data bases, documents and semi-structured data.

The *Resource Description Framework*[1] (RDF) is a graph-oriented data model, designed to store all kinds of data models. RDF Schema[2] types nodes in the graph and annotates them with the intended semantics. RDF stores data, data structures and annotations of both in the same fashion. To bridge the gap to the object-oriented Java world, we take two steps: First, we create an implementation independent triple store API (RDF_2Go). Second, on top of that, we map RDFS semantics to Java semantics (RDFReactor).

2.1 RDF_2Go

RDF_2Go[3] is a lightweight adapter framework and abstraction for existing RDF triple and quad stores and Java Applications. While there are many implementations of the

*This research was partially supported by the European Commission under contract FP6-507482 (KnowledgeWeb). The expressed content is solely the view of the authors.

CDH Seminar '05 Galway, Ireland

[1] http://www.w3.org/RDF/
[2] http://www.w3.org/TR/rdf-schema/
[3] http://rdf2go.ontoware.org

Resource Description Framework in Java, each of them has it's pros and cons and it's difficult to choose the right one for your purposes among them. Using RDF$_2$Go it's easy to change the underlying RDF store without major effects for your application: Java applications may use the RDF$_2$Go API to remove compile-time and run-time dependencies on any particular RDF implementation.

The RDF$_2$Go approach can be compared with the approach of the *Jakarta Commons Logging*[4] framework, which abstracts logging framework implementations away.

RDF$_2$Go offers methods for adding and removing statements, querying by simple triple patterns (with wildcards), and a generic SPARQL query option, which forwards the query to the underlying implementation. Query results are returned as type-safe `Iterators` over `Statement` objects. RDF$_2$Go does not offer additional parsing, persistence, or inference, as the triple stores already offer this.

The overall architecture of semweb4j is depicted in Fig. 1, which shows that RDF$_2$Go gives the developer a unified view on RDF stores. As more and more triple stores offer support for *Named Graphs* or *Contexts* or *Quads*, RDF$_2$Go provides all techniques available for triple models also for context models. An adapter allows to treat any context model as a triple model (named with a URI).

The current version 0.2 offers support for the triple stores Jena [5] 2.2 and Sesame [6] 1.2.2 and 2.0, and the quad stores YARS [7] ref. 1217 and NG4J [8] V0.4 (which builds on Jena).

Internally, RDF$_2$Go has two major parts: Common, Java-centric interfaces and a set of adapters that implement them.

Creating adapters for new triple stores is not difficult: only a handful of methods have to be implemented. E. g. writing the adapter for Sesame took less than two hours. Appropriate abstract classes provide the basis for new implementations.

Summary. RDF$_2$Go enables developers to program against a triple-centric API, without having to decide for a specific implementation for most RDF model operations. It is easy to extend RDF$_2$Go for other triple stores, and all techniques apply equally to context (quad) models as well. RDF$_2$Go has no internal state, it just acts as a facade over other (stateful) triple stores. Further information about RDF$_2$Go can be found in [4].

2.2 RDFReactor

RDF$_2$Go is helpful, but still requires full knowledge about the RDF data model. We expect many real life applications to be ontology-driven, not triple-driven. Thus we don't see a need for a Java developer to learn the RDF data model.

RDFReactor makes using RDF simpler for Java developers by enabling RDF access via object-oriented proxy objects. Java classes (with Javadocs!) are generated automatically from an RDF Schema. The usage cycle is depicted in Fig. 2. Basically, RDFS classes are mapped to Java classes and RDFS properties are mapped to method calls. For each property p with domain A and range B, the following set of methods is generated:

[4] `http://jakarta.apache.org/commons/logging/`
[5] `http://jena.sourceforge.net`
[6] `http://www.openrdf.org/`
[7] `http://sw.deri.org/wiki/YARS`, without using HTTP
[8] `http://www.wiwiss.fu-berlin.de/suhl/bizer/ng4j/`

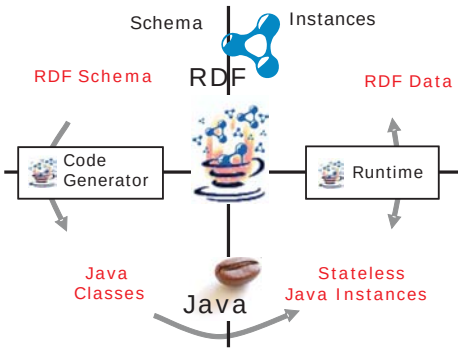

Figure 2: RDFReactorUsage Cycle.

- `public` B `get`P`()` – returns either null, a single value or throws an `RDFDataException` if the property has multiple values.
- `public void set`P`(`B `value)` – removes all existing values and sets given value
- `public void remove`p`(`B `value)` – removes the given value (if present)

By using a slight RDFS extension, schema authors can restrict the maximal cardinality. If the maximal cardinality is greater than one, two more methods are generated:

- `public void add`p`(`B `value)` – adds the value
- `public` B`[] getAll`p`()` – returns all values

All method calls result in immediate manipulation or query of the used RDF$_2$Go model. All data written to the model is ensured to be described according to the RDF Schema, e. g. on creating an instance, RDFReactor writes the appropriate `rdf:type`-statement to the model.

RDFS inheritance is mapped to Java inheritance to the possible extend. RDFReactorcalculates the longest path from an RDFS class to rdf:Resource and realises this path via inheritance. Where inheritance is not possible, property access code is duplicated.

Directly ore indirectly, all classes inherit from common base class. All instances of this class have a URI S and offer methods manipulating triples of the form `(`S`,*,*)`. The generated method implementations consists of a single line of code which calls the generic functionality from the base class. This design makes the code very "hackable".

Input files can be in RDF/XML, N3 or NT syntax. The mapping from RDF Schema to Java is fully customizable with a Velocity[9] template.

Summary. RDFReactor allows the developer to think in objects, not in statements. He can access (read and write) RDF by using familiar Java Bean idioms, e. g. use `person.setName("Max Mustermann")` instead of `addTriple( personURI, nameURI, "Max Mustermann" )`. RDFReactor is completely stateless, all method calls are ultimately forwarded to an RDF$_2$Go model. This allows e. g. simultaneous access through multiple APIs. Further information about RDFReactor can be found in [5].

[9] `http://jakarta.apache.org/velocity/`

3. THE WEB IN SEMANTIC WEB

In order to communicate with remote machines on the web, some protocol is needed. The WWW uses the application protocol HTTP to transfer resource representations. In Java, the most common API for HTTP is the *Servlet API*. There is still a gap from generic HTTP methods, which address resources via URL, and standard Java classes and instances with specific methods. The framework jREST [10] bridges this gap and exposes (annotated) standard bean-style objects as RESTful [2] web services.

jREST comes with a generic server, which allows to register arbitrary objects under a given URL-path. Incoming HTTP method calls are forwarded to bean-style methods of the appropriate registered object instance. The methods GET, PUT, POST and DELETE and forwarded to Java methods with the same name. Multiple methods with the same name can be used to model optional parameters. We use the Java 1.5 annotations[11] to annotate Java parameters with a key (`@RestAddress`) by which they should be exposed to the web. Note that Java's introspection cannot determine the parameter names in a method. Default values of parameters can also be stated using the annotation `@DefaultValue`.

Handling of the HTTP request body is achieved with a third annotation (`@RestContent`). We assume the content is a valid XHTML file containing a single description list (`dl`) in the body. jREST support serialising to and de-serialising of common data structures to an XHTML subset. We map instances from the *Java Collection Framework* in the following way: A `Set` is mapped to XHTML's `ul` (unordered list), `List` becomes `ol` (ordered list), and `Map` is mapped to a `dl` (description list). Nested data structures are mapped to nested document structures. Re-using XHTML enables a very simple an convenient debugging—any standard browser can render it.

Incoming request are matched against the available methods of the registered object, using jRESTs auto-conversion from String values to the desired data types. If all parameter values can be filled from the web request or given default values, the method is executed and the result is auto-converted into XML, using the same XHTML-subset to encode data structures as above.

Summary. jREST relieves developers from learning HTTP or Servlets. They can simply register any object with annotated methods with the jREST server and it will be exposed at runtime as a RESTful web service.

4. EXAMPLES

4.1 SemVersion

SemVersion[12] is a semantic data versioning system [4]. It uses an RDF$_2$Go quad model to store named triple sets. One of these RDF models contains all metadata about the relationship between the other versioned RDF models (author, time stamp, predecessor, ...). This information is accessed via generated RDFReactor proxy classes. In the near future, SemVersion will get a web API, which will use jREST.

[10] `http://jrest.ontoware.org`

[11] `http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html`

[12] `http://semversion.ontoware.org`

4.2 SemWiki

A semantic wiki is a wiki with additional support for authoring, browsing and querying semantic data. The implementation SemWiki[13] uses RDFReactor for its internal data management. A wiki page in this wiki consists of an `rdfs:List` which contains wiki text, resources or queries. As wiki pages are resources themselves, nested pages are possible—a page may even contain itself. The parser, a static file server and the wiki page controller are realised as RESTful services via jREST.

4.3 Cleaner

To evaluate jREST, a sample application to clean web pages was built. It fetches a given URL, applies HTML to XML cleaning to it and transforms the result with a style sheet specified by a second URL in the web request. This allows for the easy construction of wrappers (also called screen scrapers) for web pages. The server for XSLT files and the cleaner itself were exposed as web services via jREST with a minimal overhead compared to non-web application development: Both services offer a get-method in which the parameters are annotated. With three more lines of code, a web server is started and the two objects are registered as web services.

5. CONCLUSIONS

In this paper we presented semweb4j, a semantic web application development framework that requires almost no knowledge about neither semantic nor web to create fully functional, interoperable semantic web applications.

RDF$_2$Go provides a triple store abstraction, RDFReactor generates a domain-centric object-oriented API from an RDF Schema, and jRESTexposes annotated objects as web services. Taken together, these frameworks help creating robust semantic web applications easier and have been used already in real-world applications (SemVersion, SemWiki). For the future, we look into ways to streamline the development and deployment process even further.

6. REFERENCES

[1] T. Berners-Lee. *Weaving the Web*. Texere Publishing Ltd., November 1999.

[2] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[3] M. Hammer and D. McLeod. The semantic data model: a modelling mechanism for data base applications. In *SIGMOD '78: Proceedings of the 1978 ACM SIGMOD international conference on management of data*, pages 26–36, New York, NY, USA, 1978. ACM Press.

[4] M. Völkel, C. F. Enguix, S. R. Kruk, A. V. Zhdanova, R. Stevens, and Y. Sure. SemVersion - versioning RDF and ontologies. KnowledgeWeb Deliverable D2.3.3.v1, Institute AIFB, University of Karlsruhe, June 2005.

[5] M. Völkel and Y. Sure. RDFReactor - from ontologies to programmatic data access. In *Proceedings of the International Semantic Web Conference - Demo Session*, Galway, Ireland, NOV 2005.

[13] `http://semwiki.ontoware.org`