AutoRDF - Using OWL as an Object Graph Mapping (OGM) specification language

Fabien Chevalier

AriadNEXT
80 avenue des Buttes de Coëmes
35700 RENNES - FRANCE
Email: fabien.chevalier@ariadnext.com

Abstract. AutoRDF is an original open source framework that facilitates handling RDF data from a software engineering point of view. Built on top of the Redland software package, it bridges the gap between semantic web ontology and legacy object oriented languages, by providing transparent access to RDF resources from within standard C++ objects. Its use of widespread C++11, Boost and Redland makes it suitable not only for the desktop and server, but also for low computing power embedded devices. This framework is a result of the IDFRAud research project, where it is used to handle complex domain specific knowledge and make it available on smartphone-class devices.

1 Introduction

IDFRAud [4] project is an industrial research project led by French ID document verification leader company AriadNEXT. One of the objectives of IDFRAud is to propose an automatic solution for ID verification that can handle documents issued from a large set of countries. The solution will be able to execute specific controls according to the ID model (type, country, generation, etc) thanks to a knowledge base. The core idea of IDFRAud project is to provide an automatic verification system for identity documents in order to replace existing manual verification processes. The different components of ID analysis and verification in IDFRAud are driven by a set of control rules. In order to guarantee an interpretable and adaptive behavior at each ID analysis step, the identity document descriptions are organized by a knowledge management module.

One of the requirements of our knowledge management module is its interoperability and portability. It is preferred to store the data in a standard way in order to be able to use other tools such as Sewelis [8] to navigate our data. Another strong requirements are that we must be able to easily extract a subset of the knowledge base to run on mobile platforms, where C/C++ language rules. After a thorough analysis of existing technologies, we decided to use RDF for its versatility/flexibility of knowledge modelling using graphs, coupled with its capability to bring formal structure to the knowledge using Web Ontology Language (OWL).

Providing an efficient, consistent, and descriptive enough model for ID documents proves to be a very challenging task. The biggest issues encountered is the very high diversity of how ID documents look like, which makes it really difficult to design a data model that fits all cases. As such it is anticipated that the model will see serious evolutions with the number of supported documents. The second biggest issue is the fast evolving fraud patterns, which makes it necessary to add document characteristic attributes very easily in the system.

As such we needed something that could make code base maintenance easy by being able to follow a constantly evolving ontology. AutoRDF can be of some use to any kind of project that needs to manipulate RDF data where an ontology exists. The use of modern C++ make it portable to a wide variety of platforms, including all mobile phone platforms, as well as most of the embedded world systems.

2 Related work

Code generators that are used in conjunction with UML design softwares such as BOUML [1] or IBM Rational Rose [3], are principally based on data modelling. Some other well known open source frameworks such as Hibernate OGM [2], or Neo4j OGM [5] provide easy read/write for Java objects to or from graph databases. They use a more code-centric approach, where annotations store the information the framework uses to know how object must be serialized. As such they are not model-centric, but more code-centric.

AutoRDF tries to promote a software design approach where data modelling is treated as a first class citizen. It shares also some ideas with Protégé code generator plugin [6], as it uses a similar code generating approach. However it goes further than all those tools, as it does not only build interface classes that would need to be implemented by the developer, but provides a ready to use object class hierarchy to read and write data to disk.

Compared to Redland [7], AutoRDF raises the abstraction level by understanding web ontologies, as well as providing a C++ object oriented design to RDF graphs API whereas Redland is a pure C library. It has some similarities with *owlcpp* [10], however this library has no C++ proxy code generation capability, and AutoRDF tries to keep the bar as low as possible for occasional users, by providing an easy to use C++11 API. It is quite similar in goal to the Automatic Mapping of OWL Ontologies described in [9], however transferred to C++/embedded world as an open source project.

3 Introducing AutoRDF

AutoRDF is an open source semantic web code generator for C/C++. It parses a Web Ontology Language file (OWL), builds an internal representation of the ontology, and generates a set of C++ classes that make it easy to read/write RDF data from within C++. It uses the well established Redland [7] library to perform all its input/output operations. AutoRDF uses a proxy approach,

no data is copied into the C++ objects. C++ objects edit the underlying RDF graph in real time when the C++ object methods are called.

AutoRDF supports a subset of RDFS and OWL. Resources of type *rdfs:Resource* or *owl:Class* are identified as candidates for class generation, *owl:subClassOf* is used to generate inheritance relationship between generated C++ classes. *owl:oneOf* allows smart mapping to C++ enums. *owl:hasKey* generates a static object loading method, taking the key as parameter. Resources of type *owl:DatatypeProperty* and *owl:ObjectProperty* are used to generate appropriate getter/setter methods. *rdfs:domain* is used to target gettter/setter generation to the right C++ class, *rdfs:range* allows more specific C++ datatype selection. *owl:FunctionalProperty*, *owl cardinality* and *qualified cardinality* are used to choose if setters/getters should handle only single items, or if list of items should be supported. *owl:Restriction* with *owl:onDataRange* or *owl:onClass* allow to further specialize datatype of a property once applied to a given C++ object. Annotations of type *rdfs:comment*, *rdfs:label*, *rdfs:seeAlso*, *rdfs:isDefinedBy* are also used to generate documented C++ classes/methods. This subset is enough to generate C++ code that is most of the time as good as what a developer would have written by hand.

Figure 1 shows a simple UML diagram based on a simple geometry Ontology, and an corresponding OWL/RDF fragment is reproduced below.

```
@prefix geom: <http://example.org/geometry#> .

geom:topLeft a owl:ObjectProperty ;
    rdfs:domain geom:Rectangle .

geom:Point a owl:Class ;
    ...
    rdfs:subClassOf [ a owl:Restriction ;
        owl:onProperty geom:x ;
        owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
        owl:onDataRange xsd:double ] ;
    ... .

geom:Rectangle a owl:Class ;
    rdfs:subClassOf geom:Shape ;
    rdfs:subClassOf [ a owl:Restriction ;
        owl:onProperty geom:topLeft ;
        owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
        owl:onClass geom:Point ] .
```

AutoRDF provides a simple way to manipulate an RDF dataset from C++ code, as seen below:

```
geo::Rectangle r("http://example.org/myfancyrectangle");

// Set one of my rectangle coordinates - the long way
geo::Point tl;
tl.setX(1.0);
tl.setY(2.0);
```

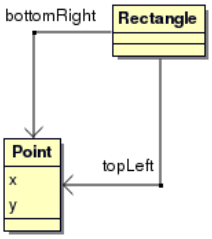

Fig. 1. Geometry ontology. This model is used as example to showcase AutoRDF code generation capabilities.

```
tr.setTopLeft(tl);

// Set one of my rectangle coordinates - the short way
tr.setBottomRight(geo::Point().setX(11).setY(12));
```

Those lines of code makes it very natural for a C++ developer to manipulate RDF data without even knowing its RDF.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix geo: <http://example.org/geometry#> .

<http://example.org/myfancyrectangle>
    geo:bottomRight [
        geo:x "11" ;
        geo:y "12" ;
        a geo:Point
    ] ;
    geo:topLeft [
        geo:x "1" ;
        geo:y "2" ;
        a geo:Point
    ] ;
    a geo:Rectangle .
```

4 Demonstration content

The demonstration will run on a laptop and will feature Protege for ontology visualization, the geometry ontology and AutoRDF as code generator, and a simple load/save example. The demonstration will also be available with a more complex scenario, a draft of the IDFRAud project ontology.

This demonstration using demo geometry ontology is available as a screencast on the AutoRDF github page at `https://github.com/ariadnext/AutoRDF`.

5 Future Work

AutoRDF is still very young. It will be used in the coming years by the IDFRAud project, and as such we will add some more features, towards making working with RDF datasets easier:

- A data quality assessment API - Checking consistency of a given RDF dataset towards a reference ontology is of crucial importance to ensure correct applicative behaviour and avoid faults that are due to poor data quality.
- Some kind of C++ OWL API, in order to make creation of other tools on top of AutoRDF easier. For instance we plan to create a graphical document model editor for IDFRAud projet, with user interface components generated automatically from the underlying ontology.

References

1. Bouml. `http://www.bouml.fr/`
2. Hibernate OGM - The power and simplicity of JPA for NoSQL datastores. `http://hibernate.org/ogm/`
3. IBM Rational Rose. `http://www.ibm.com/software/products/fr/enterprise/`
4. IDFRAud: An Operational Automatic Framework for Identity Document Fraud Detection and Profiling - Joint research project with AriadNEXT, IRISA, ENSP and IRCGN funded by ANR grant ANR-14-CE28-0012. `http://idfraud.fr/`
5. Neo4j OGM - An Object Graph Mapping Library for Neo4j. `http://neo4j.com/docs/ogm/java/stable/`
6. Protégé code generator. `http://protegewiki.stanford.edu/wiki/Protege-OWL_Code_Generator`
7. Redland rdf libraries. `http://librdf.org/`
8. Ferré, S., Hermann, A.: Reconciling faceted search and query languages for the Semantic Web. Int. J. Metadata, Semantics and Ontologies 7(1), 37–54 (2012)
9. Kalyanpur, A., Pastor, D.J., Battle, S., Padget, J.A.: Automatic mapping of owl ontologies into java. In: SEKE. vol. 4, pp. 98–103. Citeseer (2004)
10. Levin, M.K., Cowell, L.G.: owlcpp: a c++ library for working with owl ontologies. Journal of biomedical semantics 6(1), 1 (2015)