Adding Semantics to Model-Driven Software Development: A Practical Experience Report

Andreas Nareike
University of Leipzig
Augustusplatz 10
04109 Leipzig
nareike@informatik.uni-leipzig.de

Jörg Unbehauen
University of Leipzig
Augustusplatz 10
04109 Leipzig
unbehauen@informatik.uni-leipzig.de

Johannes Schmidt
University of Leipzig
Augustusplatz 10
04109 Leipzig
jschmidt@informatik.uni-leipzig.de

ABSTRACT

Model-driven software development tools and methods allow fast bootstrapping of software applications and can improve their quality. Using a domain model and adhering to a framework specific set of conventions leverages the generation of objects and services in both frontend and backend and an accompanying database schema. However, rapid application development frameworks like JHipster or Spring Roo lack a deep integration of semantic technologies which allow exploration and analytics beyond the operations defined as services. A post-deployment integration of semantic technologies into an existing application can be cumbersome, as knowledge about the domain model can get lost and additional components require deployment. We therefore enrich UML domain models in order to generate database mappings and additional ontologies for exposing both the conceptual model and instance data as RDF. Ultimately, this allows using the expressive power of SPARQL for analytic queries while still using established development frameworks for implementing enterprise applications.

1. INTRODUCTION

Modern model-driven software development frameworks, like JHipster[1] and Spring Roo[2], can generate fully functional enterprise applications including the persistence and service layer as well as the user frontend. The generation process is controlled by a domain model described as e.g. an UML model or a domain-specific language. Those frameworks support a wide range of persistence implementations like relational, document-driven or graph-oriented databases. But, to our knowledge, none of those frameworks support semantic web data formats like RDF.

To motivate software developers to use semantic web technologies, semantic web endpoints for enterprise applications can be generated. Existing relational data can be converted into RDF using a mapping approach like the RDB to RDF Mapping Language (R2RML[3]). These mappings can be directly derived from the domain model. As a result, developers can benefit from the flexibility and power of SPARQL queries and can use reasoners and semantic rules for data analytics.

In this paper, we provide a brief overview of a software generator framework, that supports model-driven generation of web-based enterprise applications as well as a mapping component to provide a SPARQL endpoint to the data. In order to facilitate the process of generating a mapping, we augment UML classes to reuse the encoded knowledge. This hybrid application design allows us to both use mature enterprise application frameworks for creating user friendly and scalable applications and integrate an extensible Linked Data based life cycle.

2. MOTIVATING EXAMPLE

To illustrate our approach, a simplified use case is chosen from the field of power plant operation. The UML classes in Figure 1 represent top-level concepts for power plant operation. The class *PowerPlant* represents the master data of an existing plant. *Documents* like manuals, reports and bills can be assigned to different plants. The type of a Document is described as a *DocumentKind*.

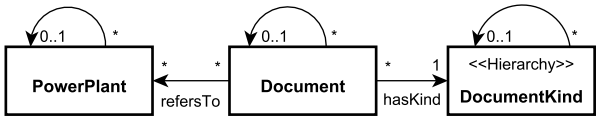

Figure 1: An annotated UML diagram

Every UML class in our example represents a hierarchy modelled by a self-relation. A power plant is divided into systems or equipment units. The different document kinds constitute a classification hierarchy. For example, data sheets and technical reports both contain technical information whereas project control documents address management information[4].

Since we are using UML for conceptual modelling, we have to deal with the underspecified semantics of this language. There have been some proposals to provide sound semantics for conceptual models build in UML. Those are largely theoretical (e.g. [5], [9]) or propose a customised UML dialect which is tied to a certain modelling style and/or limited tool support (e.g. [8], [7]). One frequent problem is multi-level modelling which occurs, when a resource can act as both an

[1]https://jhipster.github.io/
[2]http://projects.spring.io/spring-roo/
[3]https://www.w3.org/TR/r2rml/

[4]See document kinds according to the IEC 61355 standard.

SEMANTICS 2016: Posters and Demos Track
September 13-14, 2016, Leipzig, Germany

instance of a class and also as a class (often called 'power-types', e.g. [9], chapter 17.2 or 'clabjects', e.g. [1])

In our example in Figure 1, the class DocumentKind has a different meaning compared to the other classes. A document kind describes a taxonomy of typifications. So, each instance of this UML class represents a class from an ontological point of view. To distinguish the semantic of an UML class, an UML profile has been developed, that bundles a set of custom UML stereotypes. For example, the *Hierarchy* stereotype marks taxonomies. Other stereotypes can be used to control the RDF representation of UML properties and classes.

To annotate meta-data in our example, three UML profiles are included: (a) an entity description profile that defines architectural settings and persistence properties, (b) a web description profile that defines general UI settings, and (c) an ontology profile that defines semantic designations of UML classes as well as general ontology alignments.

3. RELATED WORK

Bringing together model-driven application design and semantic web technologies has been the goal of previous works. In general, we can categorise these works by examining what the primary model is and into what model it is translated.

Object-to-RDF Approaches like Jenabeans[5] annotate object oriented classes for persisting objects into triple stores.

RDF-to-Object The opposite approach is to start with an ontology in either RDFS or OWL in order to generate Java classes. An example of such a generator is Jastor[6].

UML-to-OWL There are a number of approaches that convert UML models to OWL, see for instance [6], [12], [4] or [2].

There are multiple Semantic Web frameworks and software libraries that allow a low level access to triple stores, e.g. Apache Jena[7] or rdf4j[8]. Usually at least some abstraction is provided to help leveraging the semantics defined by RDFS and OWL. Today, enterprise application frameworks or rapid application frameworks do not integrate such libraries.

There have been attempts close this gap. Empire RDF[9] implements the Java Persistence API (JPA) and could be used as a drop in replacement for JPA implementations. Additionally, the aforementioned Object-to-RDF and RDF-to-Object mapper could be used for persistence in rapid application development frameworks.

However, most of those approaches are not well maintained and we could not find evidence of major adaption in either open-source communities or enterprise contexts.

We therefore conclude, that by offering an optional, drive-by semantification, we can reach out to developers who want to stick to their familiar tools and data models but also want to enrich their application for deeper data analysis.

[5]https://code.google.com/archive/p/jenabean/
[6]http://jastor.sourceforge.net/
[7]http://jena.apache.org
[8]http://rdf4j.org/
[9]https://github.com/mhgrove/Empire

4. APPROACH

In Figure 2 the generation approach is depicted including the generated artifacts. To illustrate the dependencies between the different elements, three interactions are shown as arrows and will be introduced in this section.

First, the generator is divided into cartridges, that address different functional requirements. The application cartridge bundles templates and functions to generate a Java based backend and a JavaScript fontend. The r2rml cartridge is responsible for the data mapping generation whereas the analytics cartridge addresses the ontology generation. Second, we have the *application operation*, in which the generated application is utilised. Third, the *analytics extraction* describes how we extract and enrich RDF data from the application.

4.1 Generation Process

In the generation process step, first, as depicted in Figure 2 (1.1) the generator reads the serialised UML domain model we previously introduced in Section 2. The software generator is implemented with the help of the Eclipse Xpand framework[10] and conducts model to text transformations defined as templates. The generator cartridges have different resulting artifacts, e.g. text files, code skeletons or even executable source code. In Figure 2 the results of the generator process are enumerated from 1.2 to 1.5 including the responsible cartridges:

1.2 Backend The domain classes are generated based on the meta-data modelled with the entity description profile. Instances of these classes are exposed via REST web services for basic listing, viewing and editing operations. Further, user-specific projections and a custom search API are included. The backend application is built using the Spring Boot [11] framework.
For persistence, JPA is employed and schema creation as well as database population is delegated to the corresponding JPA implementation. A further benefit of JPA is the support for heterogeneous data models. Relational, graph and document database support can be provided by selecting an appropriate JPA implementation.

1.3 Frontend For the frontend, JavaScript modules and associated HTML snippets are generated. Using the AngularJS[12] framework, these elements are connected to the backend services and provide a rich user interface.

1.4 OWL ontology Using the annotations expressed with the ontology profile, a rich OWL ontology comprised out of OWL classes, data and object properties is generated. This ontology contains the axioms, that cannot easily be expressed in R2RML.

1.5 R2RML Mapping To provide an RDF data endpoint, an R2RML mapping is created, which maps the database used by the application backend into (virtual) RDF. This mapping connects the instance data of the database to the OWL ontology. We use an OWL ontology to enrich the RDF of the R2RML mapping, as not all information present in the UML model can be carried over into the database and R2RML mapping.

[10]https://eclipse.org/modeling/m2t/?project=xpand
[11]http://projects.spring.io/spring-boot/
[12]https://angularjs.org/

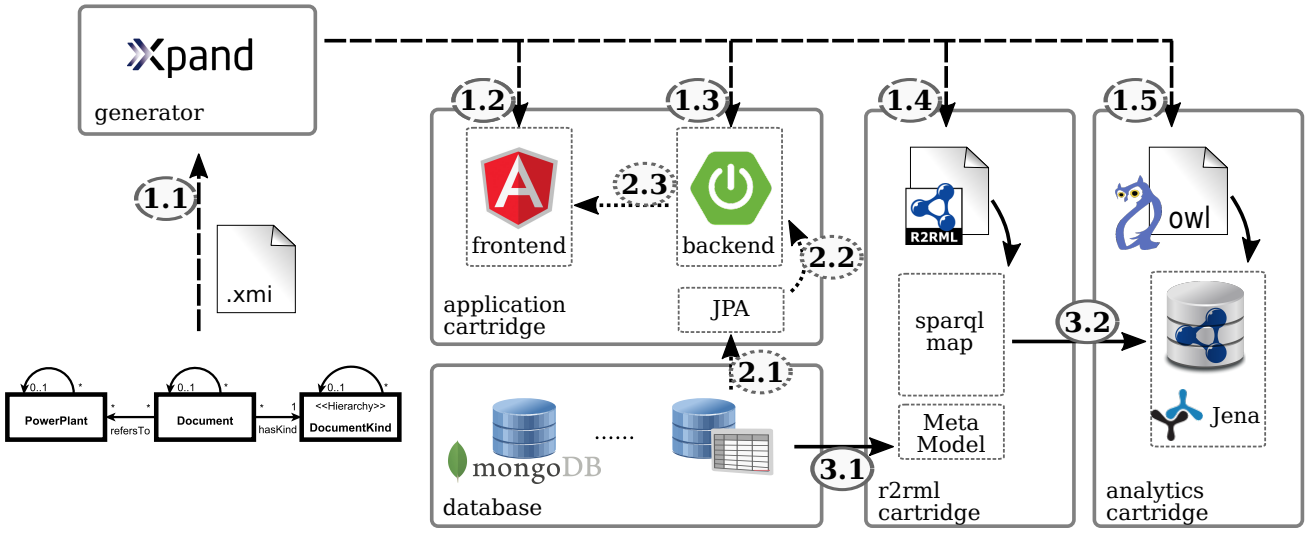

Figure 2: System overview with the main interactions: (1.1 - 1.5) addresses the generation process and its artifacts, (2.1 - 2.3) the operation of the generated application and (3.1, 3.2) how the database is extracted and enriched.

4.2 Application Operation

The typical operation and data flow of the generated application in case of a read operation is depicted in Figure 2, 2.1 - 2.3. Instance data previously stored in the database is accessed via JPA (2.1) and read into the backend application (2.2). The AngularJS UI modules accesses the REST services (2.3) provided by the application backend.

4.3 Analytics Extraction

The extraction for analytics involves two separate steps: first, a mapping into RDF and second, loading and accessing the integrated dataset.

4.3.1 RDB-to-RDF Mapping

The semantic data flow starts with RDB-to-RDF converter reading the R2RML mappings (see Figure 2, 1.4) previously generated and accessing the database (see Figure 2, 3.1). The result of this mapping is a (virtual) RDF graph that can be exposed as both dump or SPARQL endpoint.

For converting relational databases into RDF, multiple implementations have surfaced, such as D2R [3], ontop [13] [10] or SparqlMap [11]. We tested these implementations for fitness in our use case using the following requirements:

1. Compliance with R2RML, providing a standardised mapping mechanism and consequently preventing a lock in.
2. SPARQL and RDF dump facilities, for accessing the data either as virtual or materialised RDF graph.
3. Pure Java implementation, as the backend is already a Java application, the necessary runtime is present.

All three candidates support both SPARQL and RDF extraction and are all written in Java, but as D2R does not fully support R2RML, we ruled it out. We selected SparqlMap over ontop as SparqlMap offers support for NoSQL databases via Apache MetaModel[14] and therefore implies less restrictions on selecting a JPA conforming backend. While ontop offers reasoning support, we can perform reasoning enabled querying over an in-memory model, as the expected model size is expected to fit well into main memory.

4.3.2 Analytics Integration

The materialised mapping created using the R2RML mappings is enriched with the OWL ontology initially generated and loaded into an Apache Jena in-memory model (see Figure 2, 3.2). This model allows plugging reasoners which allow making use of the richer OWL constructs generated from the UML domain model.

5. GENERATED EXAMPLE

To complete our use case example, the most important generator artifacts are described according to the domain model presented in Figure 1. For every UML class, Java beans and services are generated as well as a web frontend including input forms according to their UML properties. The resulting web application is fully functional and users can create and modify data.

For validation and analytics, an ontology is generated by combining concepts derived from the UML model with instance data provided at runtime by the application user. UML associations between them are represented by OWL object properties.

Instance data corresponding to the UML classes *PowerPlant* and *Document* are mapped to typed RDF instances. However, the instances of the UML class *DocumentKind* will be mapped to a set of OWL classes that constitute a hierarchy. Consequently, the UML association *hasKind* will be mapped to the `rdf:type` property.

For the example model in Figure 1, this yields the following RDF (some triples have been omitted for brevity):

[13]http://ontop.inf.unibz.it/
[14]http://metamodel.apache.org/

```
:Document a owl:Class .
:DocumentKind a owl:Class .
:PowerPlant a owl:Class .

:refersTo a owl:ObjectProperty ;
    rdfs:domain :Document ;
    rdfs:range :PowerPlant .
```

Listing 1: RDF Turtle snippet[15]

The details of the mapping, i.e. which RDF resources are used for the UML classes and associations, are controlled by an UML stereotype. Therefore, it is also possible to reference external ontologies. Cardinality restrictions on the association will be mapped to according OWL restrictions. Properties can be further specified (e.g. as `owl:transitiveProperty` by annotating the UML association accordingly.

The relational data will be transformed by an R2RML mapping which produces additional triples. A point of special interest is that the different hierarchies do not share the same semantics. The power plant hierarchy is built using a part-whole-relationship, whereas the document kind hierarchy is best represented by a usual subclass hierarchy. The second case must be treated differently since the instances of this UML class must be mapped to OWL classes. Whereas for power plants, the property `rdf:type` is used, the property `rdfs:subClassOf` will be used instead. For our use case example, these triples are produced (some are omitted for brevity):

```
:Technical_report rdfs:subClassOf :DocumentKind .
:Inspection_report rdfs:subClassOf
    :Technical_report .
```

Listing 2: RDF Turtle mapped with R2RML

Even without any reasoning with respect to the semantics of RDFS/OWL, this provides the benefit of running SPARQL queries on the data. By using established top-level ontologies, the semantics can further be clarified. For more extensive analytics, we have tested augmenting the resulting ontology with rule sets.

6. CONCLUSION

Today, many enterprise applications do not benefit from the power of semantic web tools and frameworks. In the area of model-driven software development, semantification of (existing) applications can be achieved by using an OWL cartridge as well as mapping techniques for RDF data.

The semantics of UML classes can differ. It is hard to distinguish whether an instance of a UML class is an instance from an ontological view. Using an ontology profile, it is possible to annotate the intended semantic of an UML class. Based on these information, a formal ontology can be generated that is used to define relation data mappings into RDF.

With our approach, the development of classical enterprise application is supported as well as semantic web applications. In a first practical use case, we used the semantic web endpoint to analyse complex relations between strongly interlinked data in the application area of sustainable energy power plants.

7. ACKNOWLEDGMENTS

This work was partly supported by the following grants from the German Federal Ministry of Education and Research (BMBF) for the LEDS Project under grant agreement No 03WKCG11C as well as the CVtec research project as grant 01IS14016C.

8. REFERENCES

[1] C. Atkinson and T. Kühne. In defence of deep modelling. *Information and Software Technology*, 64:36–51, Aug. 2015.

[2] A. Belghiat and M. Bourahla. From UML class diagrams to OWL ontologies: a Graph transformation based Approach. *International Journal of Computer Applications*, 41(3):41–46, 2012.

[3] C. Bizer and R. Cyganiak. D2r server-publishing relational databases on the semantic web. In *Poster at the 5th International Semantic Web Conference*, volume 175, 2006.

[4] S. Brockmans, R. Volz, A. Eberhart, and P. Löffler. Visual modeling of OWL DL ontologies using UML. In *The Semantic Web–ISWC 2004*, pages 198–213. Springer, 2004.

[5] M. Clavel and M. Egea. An Algebraic Semantics for UML+ OCL Class Diagrams. *Universidad Complutense de Madrid, Spain*, 2006.

[6] D. Gasevic, D. Djuric, V. Devedzic, and V. Damjanovi. Converting UML to OWL ontologies. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 488–489. ACM, 2004.

[7] G. Guizzardi, A. P. das Graças, and R. S. Guizzardi. Design patterns and inductive modeling rules to support the construction of ontologically well-founded conceptual models in OntoUML. In *International Conference on Advanced Information Systems Engineering*, pages 402–413. Springer, 2011.

[8] G. Guizzardi, G. Wagner, N. Guarino, and M. van Sinderen. An ontologically well-founded profile for UML conceptual models. In *Advanced Information Systems Engineering*, pages 112–126. Springer, 2004.

[9] A. Olivé. *Conceptual Modeling of Information Systems*. Springer, Berlin ; New York, 2007 edition edition, Aug. 2007.

[10] M. Rodrıguez-Muro, J. Hardi, and D. Calvanese. Quest: efficient sparql-to-sql for rdf and owl. In *ISWC*, 2012.

[11] J. Unbehauen and M. Martin. Executing sparql queries over mapped document stores with sparqlmap-m. In *Proceedings of the 12th International Conference on Semantic Systems*, pages 485–490, 2016.

[12] J. Zedlitz, J. Jörke, and N. Luttenberger. From UML to OWL 2. In *Knowledge Technology*, pages 154–163. Springer, 2012.

[15] For all Turtle examples, we omit this prefix declaration:

```
@prefix : <http://example.org> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
```