Applying rules to form generation from OWL models

Jean-Marc Vanel[1],

[1] JMV, 19 avenue Mirabeau, 78000 Versailles, France
jeanmarc.vanel@gmail.com

Abstract. Starting from domain models in OWL, we apply rules in N3 logic with a RETE rule engine to generate fully functional GUI's in Java Swing. Form generation is just a particular case of object graph transformation through logical rules; this technique is also applied to transform UML into OWL. Widgets and user events are translated to a platform independent GUI model, thus enabling platform independence, even for Web applications. This logic-based framework promises user-friendliness by inferring logical behavior, i.e. navigation and CRUD, from the OWL model and specific business rules.

Keywords: rule engine, GUI generation, object graph transformation, N3.

1 Introduction

In order to facilitate development of business applications there are three main ways: reuse domain models and rules, reuse existing software components; and ease authoring. In this paper we will concentrate on the first issue.
The highest accomplishment would be to reproduce the reasonings in a designer's mind leading from the specifications to the application design. The actual goal is to automatize as much as possible of the software development, starting from domain models and rules, while leaving room for customization. The approach consists in modeling not only the business domain, but also the software concepts, including application specification, and to reason about both.
The tools presented here are able to generate from an OWL model a simple Java Swing application with these features: edit datatype property values, create or link resources for object properties. For a given RDF resource R, it displays all properties implied by the OWL model, plus all RDF statements about R. To enable link to user chosen resource, it has some generic navigation features within an RDF database. Moreover, it can implement application specific behavior from N3 rules at runtime.
The run-time rule engine is able to generate a complete input form in milliseconds, directly from the OWL model. This opens the way for using rule and logic based techniques for complex presentation and behavior exploiting the model, the data and the user actions (present and past).

2 GUI generation : state of the art

Lots of tools use ad-hoc code: Ruby On rails, PHP frameworks. The Model Driven Architecture (MDA) [1], from the Object Management Group relies on meta-modeling and Object Constraint Language. OWL and the Description Logics are a way of modeling the real world that allows reasoning. On the other hand MDA techniques are based on UML and MOF that are little more than structural descriptions.
Graph rewriting techniques are often used [2].
Then there are the logic programming approaches [3] [4] [5] that are the most similar to ours.
The GUI's generated are not only forms and tree based input, but can also be graph based (i.e. showing vertices and edges) [6].
Some use the Fresnel [7] RDF presentation vocabulary to select the properties being displayed, and the CSS like style applied.

3 Overall architecture

The Data flow, from an application designer point of view, is depicted in figure 1. At design time the inputs are a domain model in OWL and an application specification in N3. A rule engine is then used to obtain a list of N3 (RDF) inferred statements. Not shown on this diagram are N3 rules expressing the knowledge to build the application from the designer inputs (see paragraph “Object graph transformation though rules”). From these inferred statements an object graph of Java objects is generated, relying on simple N3 patterns for Java objects creation, property assignment and method calls. More precisely these Java objects result in a Swing Graphical User Interface (GUI) showing forms and fields for the end-user data (i.e. instances of the OWL domain model). This end user application can be run in a dedicated Integrated Development Environment (IDE) called EulerGUI [8]; or be exported and deployed on any Java enabled platform.
A runtime the end-user application relies on two generic services. First, an RDF in-memory storage holds the end user data. It also includes a forward chaining RETE [9] inference engine (Drools [10]) running much the same rules as the design time engine; we can call this the Knowledge Base (KB). Second, a GUI – KB adapter forwards the relevant user GUI events to the KB. This is done generically via Java introspection. There is no decisions made in the Java code of the GUI – KB adapter, just verbatim forwarding of the event properties.

We think that Object Oriented (OO) storage for the business data has lived, like when Java classes are generated from ontologies [11]. It's time for knowledge bases! In OO implementations, the data model, the business rules, and the infrastructure are easily

mixed. It's a Copernican revolution ! Note that OO remains fit for the infrastructure code, though.

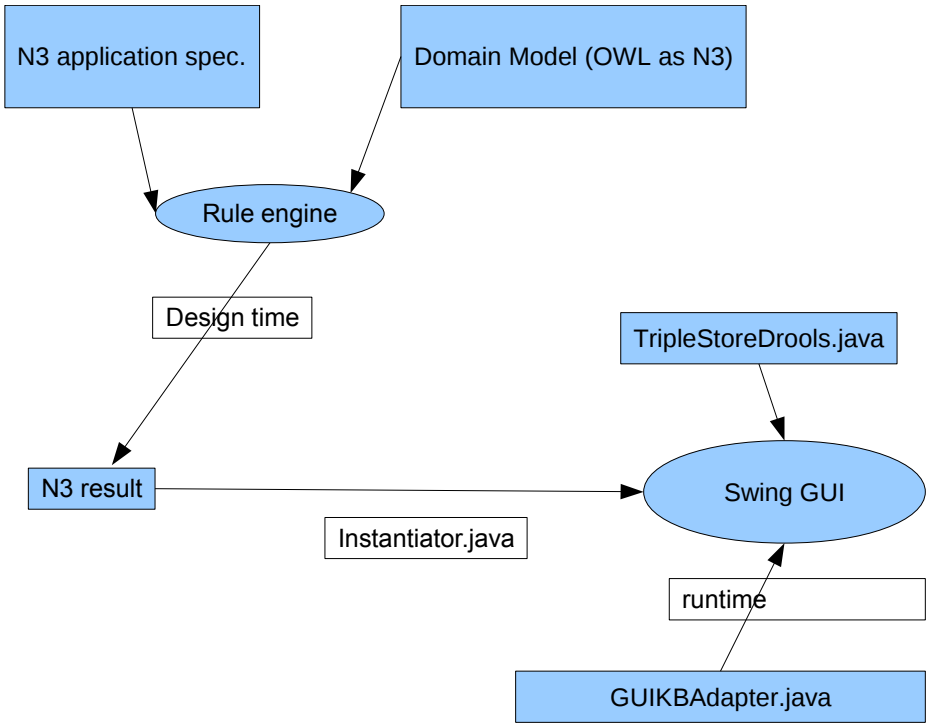

Fig. 1. The data flow, from an application designer point of view

3.1 Rule engines

For GUI generation, we began by using the Euler engine [12] in Prolog, but it is not well suited for generating object graphs steps by step. So we reused Drools, an efficient forward chaining engine in Java, using the famous RETE algorithm. Drools can store in its Working Memory any Java object, but we use just one Java class called Triple, with three string properties: subject, predicate, object. We implemented a simple translator from N3 rules to Drools rule language. The translation is quite straightforward, with an exception. In a simple case like this, we would create 2 blank nodes ?Y :

```
:x :p :fact .
{
 :x :p ?V .
} => { ?Y :p :inferred } .
```

So, in the translator to Drools rule language, we added the negation of the consequent in the antecedent.

3.2 N3, ontologies and rules

We chose the N3 file format, because it is quite human readable, compatible with RDF, and it can express facts, models, rules and queries. It was created by W3C for its own use, but is not yet a W3C recommendation. It is readable by Protégé, Jena, and many tools; a syntax colored editor exists (Vim). We use the same rule language for infrastructure (form generation) and for business rules.
N3 is quite similar to SWRL in its semantics, and similar to SPARQL in its syntax.

4 Tools

EulerGUI actually started as a GUI for the Euler inference engines [12]. Then we added other N3 rules engines: the reference implementation CWM, Fuxi another Python implementation, and Drools.

EulerGUI can open any number of RDF / OWL / N3 documents, run the rules using the 4 rule engines (Drools, Euler, CWM, Fuxi), and generate an application. The rules for application generation are in a separate Open Source Software (OSS) project, Déductions.
There is a reader for UML / XMI / eCore models and data, that translates directly in N3. Then UML class models expressed in N3 can be transformed in OWL with N3 rules. Euler can chain inferences in pipelines, like this one:

```
UML model → OWL model → Java application
```

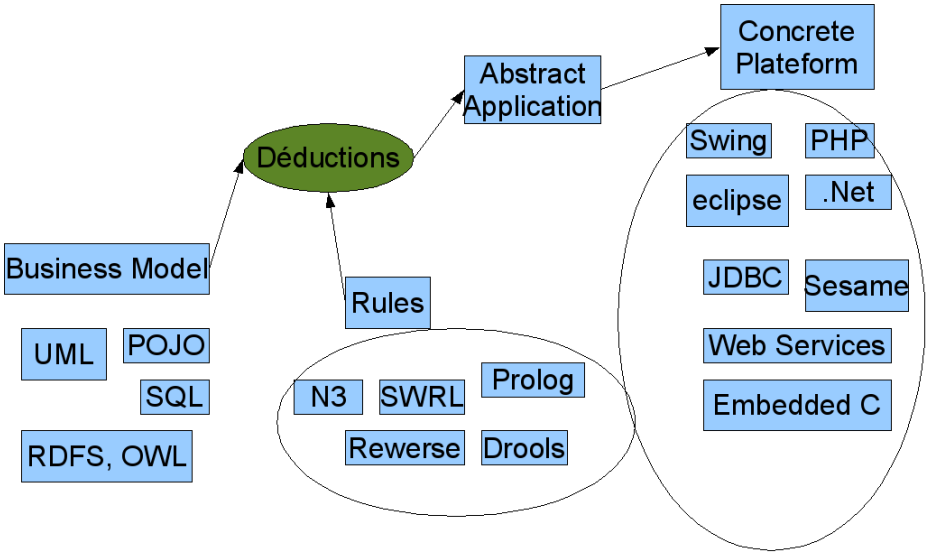

Fig. 2. File formats and platforms foreseen for EulerGUI and Déductions

5 Object graph transformation though rules

There are at least two architectures. The first is by creating objects step by step, rule by rule (see following examples). If one has a one to one correspondence, for example when creating OO classes from SQL tables, it is simpler to annotate existing objects without creating objects.

Comparison with XSLT With RDF properties and logic rules one is independent of the structure, whereas XML is mainly structured data.

Comparison with QVT instead of being logic based, QVT is partly declarative and procedural.

Layered transform Instead of directly creating Java object, we use an intermediary generic GUI model (see fig.3). With some more work, this would enable platform independence, even for Web applications.

```
# add a field in the form for each property of a class:
{ ?CLASS gui:hasForm ?FORM .
  ?PROP rdfs:domain ?CLASS .
} => {
 ?FORM gui:hasField ?FIELD .
 ?FIELD gui:inputWidgetSpecification ?PROP .
} .
```

Fig. 3. A typical rule creating a blank node ?FIELD, and connecting it to existing nodes ?FORM and ?PROP

```
# the type of the field depends on the type of the
# property: ObjectProperty or DatatypeProperty
{
 ?FIELD gui:inputWidgetSpecification ?PROP .
 ?PROP a owl:DatatypeProperty .
} => {
 ?FIELD a gui:DatatypeInputWidget .
} .
```

Fig. 4. A rule adding a property (here an RDFS type) to an existing node

As part of the Euler project, a library of N3 rules implements the logic of OWL and RDF Schema (transitive property, inheritance, etc), and other goodies, see fig.5 and owl* N3 rules in http://eulersharp.svn.sf.net/viewvc/eulersharp/trunk/2003/03swap/

```
{ ?P a owl:TransitiveProperty.
  ?S ?P ?X.
  ?X ?P ?O. } => { ?S ?P ?O }.
```

Fig. 5. A rule implementing in N3 logic the OWL Transitive Property concept

Figure 6 show typical business rules.

```
  # For an order, there is an associated payment :
  { ?C a Order } =>
  { ?C hasPaymentRequest ?PR }.

  # If the holder had a payment incident, then the order
is cancelled :
  { ?C hasHolder ?H .
    ?H hasPaymentIncident ?PI } =>
  { ?C hasStatus cancelled }.
```

Fig. 6. Typical business rules.

6 Conclusion

Form generation is a showcase application; the main point is that we have a framework (Open Source) allowing to mix ontologies and a RETE forward chaining engine at runtime. The sources can be N3 (Notation 3), RDF, OWL, UML or eCore, but everything is converted in N3. The rules are in N3 too; so the same rule language is used for infrastructure (form generation) and for business rules.
Although the framework is here, much remains to do to implement OWL 2, Humane interface (user-friendlyness), multi-platform .
Automatize application building will allow IT projects to concentrate on essential matters: domain model and business rules.

References

1. OMG Model Driven Architecture http://www.omg.org/mda/
2. Calleros, J.M.G., Stanciulescu, A., Vanderdonckt, J., Delacre, J.P., and Winckler, M. A Comparative Analysis of Graph Transformation Engines for User Interface Development. In: Proceedings of the 4th International Workshop on Model-Driven Web Engineering, MDWE pages 16-30.(2008)
3. Appeltauer M., Kniesel G., Towards Concrete Syntax Patterns for Logic-based Transformation Rules. In: Electronic Notes in Theoretical Computer Science (ENTCS), Volume 219 , Pages 113-132 (2008)
4. Chun Yip Leung, N3 Rules Processing and HTML Translation for the Semantic web, Digital Media Systems Laboratory, HP Laboratories Bristol, HPL-2003-199, October 2nd , 2003*
5. Gassert, H. and Harth, A., From graph to GUI: displaying RDF data from the Web with Arago. In: ESWC'05 SFSW, Heraklion, 29 May–1 (2005)

6. Ehrig K., Ermel C., Hänsgen S., Taentzer G., « Towards Graph Transformation Based Generation of Visual Editors Using Eclipse ». In: Electr. Notes Theor. Comput. Sci, 2005.
7. Fresnel - Display Vocabulary for RDF http://www.w3.org/2005/04/fresnel-info/
8. Eulergui, an N3 centered rules IDE http://sourceforge.net/projects/eulergui
9. Charles Forgy, "On the efficient implementation of production systems." Ph.D. Thesis, Carnegie-Mellon University, 1979.
10. the Drools RETE expert system http://jboss.org/drools/drools-expert.html
11. A Semantic Web Primer for Object-Oriented Software Developers http://www.w3.org/TR/sw-oosd-primer/
12. Euler Proof Mechanism, an inference engine supporting logic based proofs http://sourceforge.net/projects/eulersharp