Position Paper: Validation of Distributed Enterprise Data is Necessary, and RIF can Help

David Schaengold
Director of Business Solutions
Revelytix, Inc
Sept 19, 2011, Revised Oct 17, 2011

Overview

Revelytix has been investigating the use of validation rules to ensure the consistency and correctness of data exposed by our integration software. We have come to the conclusion that whenever multiple data sources are integrated at run time for enterprise applications, a distinct, abstraction-level set of validation rules should be applied to the integrated data, beyond whatever store-level validation is already in place. In order to accomplish this, we use a forward-chaining rules engine and the Rules Interchange Format (RIF).

Background

Revelytix uses the semantic web standards (OWL, RDF, and SPARQL) as the backbone of its model-driven software tools whose primary function is enterprise information integration. The software allows multiple structured data stores, which are typically existing operational or legacy systems, to act as the ultimate persistence layer while exposing their data as a single queryable dataset. The software provides a SPARQL 1.1 query engine that is abstracted from any particular store, as well as a tool for translating between SPARQL and native structured query languages dynamically. In combination, the software allows users to query existing distributed enterprise data using SPARQL as a query language and an OWL ontology, typically containing enterprise-level concepts of the kind found in taxonomies and business glossaries, as a schema. The data is translated from its original schema and format at run time.

In initial pilots demonstrating this solution, we did not make any attempt to validate the data exposed by the query engine post-integration (that is, query result sets) beyond checking that the data originated in the correct underlying sources. However, we quickly realized that this approach was insufficient because it is possible for errors to emerge only after integration, even without technical failures such as garbled packets or unparsed frames. In most cases these errors actually reflect errors in the underlying data that are undetectable given the information available to a particular data store. As an example, consider a plausible data validation rule stating that all persons must have no more than one social security number. For a given person, one data store might contain an assertion indicating that a particular person P's social security number is a value A, while another contains an assertion indicating that the P's social security number is a different value, B. In each data source, a validation mechanism checks to make sure that all persons have a maximum of one social security number, and in each data source the check passes. However, the integrated data set contains both values, A and B, for P's social security number. A validation mechanism checking for violations on the integrated dataset will catch this error, which could not have been detected by the data sources themselves. The example given is extremely simple, but much more complex cases that are similar in structure have presented themselves.

Validation Rules in Practice

Exposing integrated enterprise data as RDF presented both opportunities and challenges for validation. The principal opportunity was the ability to write rules against data of a single format, in terms of enterprise-level concepts. This contrasts with traditional validation mechanisms, which must address themselves to data in the format of the local application, and must do so using whatever schema is

used by the local application. Because the schemas used by traditional systems tend to be relatively fixed over time, these validation mechanisms are often created without significant allowance for the evolution of validation rules, and the concepts used in writing the rules must be local. Because the schema of the data being validated in our situation is an OWL ontology representing an integrated view, however, it is possible to write rules using a much more general method. For a rules language we elected to use a W3C standard, the Rules Interchange Format (RIF), and to write rules using RIF frames to represent RDF triples in the ontology, and including only constants that were URIs or literals found in the ontology or expected from the data. The most important criteria in selecting a rules language were that the language be able to interoperate effectively with RDF data and be widely implemented, used, or promoted. This effectively limited our choices to three rule languages: RIF, SWRL, and SPIN. RIF was ultimately chosen for four reasons: (1) RIF is a W3C standard, while SWRL and SPIN at present are not; (2) RIF has a larger and more expressive set of built-in predicates; (3) RIF is designed to be interchangeable with other rule formats, including formats in wide industry use; (4) The RIF specification includes a framework for extending the language in cases where the existing specification is insufficient.

While SPIN does have the advantage of a SPARQL-based syntax, which is more familiar to users of the semantic web standards than the RIF presentation syntax, in the long run the similarity can be confusing, as there are subtle differences between SPARQL CONSTRUCT statements and executable rules which can at times be significant. In these cases the similarity of SPIN syntax to SPARQL is a disadvantage, not an advantage.

Using enterprise concepts defined in OWL in order to write validation rules in RIF proved to be a powerful combination. However, challenges also arose. The first challenge we encountered was in deciding how to express rule violations in RIF. Two possibilities presented themselves. The first possibility was to invent an RDF predicate, or set of predicates, which would be asserted of data violating rules. Using the example given above, for instance, a rule might be written in RIF as follows:

```
Forall ?x ?y1 ?y2 (
       ?x :violatesRule "true"^^xsd:boolean :-
       And(
              ?x[:socialSecurityNumber->?y1]
              ?x[:socialSecurityNumber->?y2]
              External(pred:notEqual(?y1 ?y2))
       )
)
```

This rule would generate an entailment asserting that the bindings for ?x are in violation of a rule. This approach proved to unsatisfactory, however, for two reasons. First, it seemed perilous to assert that whatever was represented by a particular URI was in violation of a rule. In our example, person P has not violated a data rule. Rather, the URI *representing* her is the guilty party. But our entailment states that the bindings of ?x violates a rule, and person P is named by a URI in those bindings. In other words, this approach violates the implicit use-mention distinction that underlies the use of URIs in RDF, and could lead to significant problems if the generated entailment were to be used by reasoners. Secondly, we found it to be frequently the case that errors were not generated by the bindings of a single variable, but rather required a concert of triples. In that case it is obviously deficient to assert that the bindigns for a single variable violate a rule, and confusing to assert this of the entire solution set involved in the error.

As a consequence, we elected to use only rules that generated no entailments for validation. The W3C document defining a RIF implementation of OWL-2 RL[1] introduces a zero-argument predicate, `rif:error()`, which we used exclusively as the consequent in validation rules. In principle, any zero-argument predicate would serve the same function, however, as long as it could be correctly interpreted by the rules engine that executes the rules, and in the future if `rif:error()` comes to be widely used to express only logical errors, and not errors more broadly, we expect to begin using two zero-place predicates: `rif:error()` for logical errors, and a different predicate for non-logical errors. Using `rif:error()`, the same validation rule as written above would look like this:

```
Forall ?x ?y1 ?y2 (
      rif:error() :-
      And(
            ?x[:socialSecurityNumber->?y1]
            ?x[:socialSecurityNumber->?y2]
            External(pred:notEqual(?y1 ?y2))
      )
)
```

This solves the problem with violating the use-mention distinction, but not the problem of capturing complex violations. Ultimately this challenge had to be addressed by the software itself, and so the rules engine that was built to execute RIF is designed to generate a SPARQL query that reproduces the precise set of triples that generated `rif:error()` in a particular case.

As it happened, using a zero-argument predicate helped us surmount another challenge, which was the absence of negation in the RIF Basic Logic Dialect (RIF-BLD). In the examples above, all the necessary negation is handled by a built-in predicate, `pred:notEqual(`Arg_1 Arg_2`)`. However, in many cases we needed a more general negation operator, in cases when the erroneous state of affairs can only be represented by a graph. We realized that validation rules had the following very general form:

> X (a fact about a domain) must be in Y (a set of possible facts about a domain)

Where Z is the subset of Y relevant for the rule, and translated into RIF, that general form would be expressed as follows:

```
rif:error() :-
And(X ~Z)
```

Sometimes the required negation of Z could be accomplished by a built-in predicate, but very often a general negation operator was necessary. Unfortunately, the `NOT()` operator, which is the general-purpose negation operator in RIF, is only available in the PRD dialect, and not the BLD dialect.

We had desired to use only RIF-BLD rules because of their greater simplicity compared to RIF Production Dialect (RIF-PRD) rules. RIF-PRD is aimed at a very different set of cases than RIF-BLD, and it is simpler to build a rule execution engine if the executed rules are written in RIF-BLD. One of the specific difficulties in building a RIF-PRD rather than a RIF-BLD execution engine is that in the latter case the entailments generated by rules are strictly monotonic – that is, the set of assertions in the graph only increases or remains the same as a result of running the rules, and never decreases.

---

[1] http://www.w3.org/TR/2009/WD-rif-owl-rl-20091001/#Inconsistency_rules

Because `rif:error()` never leads to the assertion of new information or the deletion of existing information, using a zero-place predicate as the consequent of our validation rules allowed us to use the `NOT()` syntax of PRD in our rules while at the same time preserving a monotonic relationship between the input and output of the rule engine. While the Revelytix rules engine now does support non-monotonic entailments, being able to guarantee that RIF validation rules will never remove assertions makes them more broadly adoptable.

A third challenge was presented by the semantics of OWL and RDF. These standards follow what is called an "open-world assumption," meaning that one may not deduce from the absence of a piece of information that the piece of information is false. And yet many of our validation rules seemed to imply just this. For instance, a validation rule requiring every person to have at least one social security number could be valuable in many cases, but it would require treating the absence of a triple linking a person with a social security number as erroneous.

Once again, it was realized that the apparent dilemma was a case of confusing use and mention. RDF and OWL make use of the open-world assumption in drawing new inferences from existing facts, but these inferences are themselves facts about the part of the world represented by the graph or ontology, not about the presence or absence of particular triples. In fact, it became clear that open world semantics were an ally of data validation, because OWL inferences could expose errors that would otherwise remain hidden in integrated data. For instance, suppose that person P was represented by two distinct URIs, and that the URIs were linked with an owl:sameAs assertion[2]. Suppose further that each of these URIs had a distinct value for :socialSecurityNumber. In Turtle syntax, the data might look like this:

```
<http://www.example.com#person12345> :socialSecurityNumber "123-45-6789" .
<http://www.example.com#person54321> :socialSecurityNumber "987-65-4321" .
<http://www.example.com#person12345> owl:sameAs <http://www.example.com#person54321> .
```

These three triples, by themselves, would pass the validation rule written above, even though a human can easily see that it violates the spirit of the law, if not its letter. What is needed to make the triples correctly trigger a validation error is the OWL inference rules. Fully inferenced under standard OWL-RL rules, these three triples would generate three further assertions, leading to a total of six:

```
<http://www.example.com#person12345> :socialSecurityNumber "123-45-6789" .
<http://www.example.com#person54321> :socialSecurityNumber "987-65-4321" .
<http://www.example.com#person12345> owl:sameAs <http://www.example.com#person54321> .
<http://www.example.com#person12345> :socialSecurityNumber "987-65-4321" .
<http://www.example.com#person54321> :socialSecurityNumber "123-45-6789" .
<http://www.example.com#person54321> owl:sameAs <http://www.example.com#person12345> .
```

Because two values of :socialSecurityNumber are asserted for each URI representing person P, the validation rule above would trigger.

---

[2]Using standard owl:sameAs rules. In RIF the relevant rule is as follows:

```
Forall ?p ?o ?s ?s2 (
   ?s2[?p->?o] :- And(
       ?s[owl:sameAs->?s2]
       ?s[?p->?o]  ))
```

Here the value of building a general-purpose rules engine became apparent. The OWL-RL inference rules can be written as RIF, and executed by the Revelytix rules engine at the same time as the validation rules are invoked. The results, a set of zero or more entailments from the OWL rules and zero or more errors from the validation rules, are asserted into a graph kept separate from the input graph. This keeps the (possibly incorrect!) entailments from contaminating the underlying data drawn from the original enterprise data stores. Any errors that arise, including errors that can only be found by checking the entailed data against the original data, are identified and reproduced by a SPARQL query provided by the software, leading to a simple report on the integrity and correctness of the underlying data that could otherwise have been provided only with great effort.

Conclusion

The W3C semantic technology standards have proved a firm foundation for potentially transformational data integration techniques. To the core group of standards – OWL, RDF, and SPARQL – Revelytix has found RIF a helpful addition for the use-case of data validation. While there are difficulties to be surmounted, data validation at the abstraction level is essential, beyond validation at the individual store level. The combination of RIF rules and OWL ontologies for abstraction-level validation provides a powerfully general, standards-based way of enforcing standards of data quality and correctness across all of an enterprise's data.