Thoughts on Validating RDF Healthcare Data

2013 W3C RDF Validation Workshop
David Booth
KnowMED, Inc.

Healthcare data often involves combining several datasets from different sources – any of which can contain source-specific data errors that are (in principle at least) often detectable. This position paper briefly outlines some of the features I'd like to see in RDF validation tools.

1. **SPARQL-based framework.** I favor SPARQL-based approaches because there is a large management economy that is obtained by using a widely used common-denominator tool. I am most drawn toward approaches that either build on SPARQL as a component, or can be used from SPARQL (such as SPARQL extension functions).

2. **Validation pipelines.** A single SPARQL ASK or CONSTRUCT query can be overly complex and difficult to debug. Often it is simpler to write a series of SPARQL operations – such as INSERT operations that operate on named graphs – to create intermediate forms of the data that are easier to check than the original data. At present there does not seem to be a standard way to specify a series of SPARQL Update operations followed by a SPARQL Query. (The RDF Pipeline Framework (http://rdfpipeline.org/) is one approach that can be used, but certainly not the only one.) It would be helpful to have standard ways to define validation pipelines.

3. **Better URI pattern matching and munging.** RDF applications routinely generate new URIs from natural keys in the data. It would be nice to have easier mechanisms for checking URI patterns and detecting misspellings.

4. **Validation like automated regression testing.** In any serious software development effort it is essential to have a regression test suite that grows as the software evolves, features are added and bugs are fixed. Since RDF is by nature a very unconstrained language, validation becomes similarly essential. Therefore, I prefer validation approaches that are amenable to modularity, such that new, independent tests can be conveniently added without affecting existing tests, much as one might add a new regression test when a feature is added. To be concrete: for ease of automation and maintenance I prefer an approach in which each validation test (or group of related tests) can be naturally expressed in a separate file and run independently from other validation tests.

5. **Operational versus declarative.** For very simple tests (such as simple pattern matching) a declarative style can be convenient. But in my experience, for more complex tests regular programmers are usually more comfortable with an operational style ("Do A first, then B, then C, and then the result should be X") than a declarative style. I note that SPARQL Update operations are amenable to an operational style, as a series of updates can be specified.