Querying RDF: SPARQL 1.2/2.0 and imperative query languages

Position statement by Adrian Gschwend, CEO of Zazuko GmbH in Switzerland for the *W3C Workshop on Web Standardization for Graph Data* in Berlin, March 2019.

I am CEO of a Swiss based Linked Data consulting company, founded in 2014. I first heard of RDF and Linked Data in 2008 and found it intriguing. One year later I started using it for first projects. I am an engineer and always thought of Linked Data and RDF as a way to solve real world problems. While I enjoy reading papers in the domain and play around with prototypes, I strongly believe Linked Data is the most powerful tooling available for the real world as of now. Our company helps government organizations and large enterprises to solve complex problems based on the Linked Data stack.

We are known for many Open Source libraries and standards we co-develop mainly within the W3C RDF JavaScript community group, which I initiated years back. My personal focus is on data pipelining and SPARQL. I enjoy writing SPARQL queries and publish a lot of them around public data sets in Switzerland. A good selection of queries based on our data can be found in the Open Data Advent Calendar which was last active in December 2018.

Within discussions on Twitter and with specific people I advocated the need for further development on the SPARQL query language. My idea is to define a few low-hanging fruits that could go into a SPARQL 1.2 release and then think about a more radical revision which could lead to SPARQL 2.0. SPARQL is a great query language that already found its place in the real world. A SPARQL 1.2 release could address some of the easier to solve issues people have with SPARQL without breaking anything in SPARQL 1.0 and 1.1. I am by far not alone in that regard, see for example this blog post and some of the features and extensions Apache Jena implements as well.

The discussion about SPARQL 2.0 may be more open to results. In my opinion it would be allowed to break things in such a release if it brings the RDF stack further. There are some things that cannot be queried easily if at all in SPARQL 1.x. A good example of ideas in that direction is for example `PATH` queries in Stardog. This is the kind of discussions I would like to start for a 2.0 version.

From a graph point of view it could also be useful to talk about other ways to query RDF data. SPARQL is a great declarative way to access RDF and works for many use cases. While developing our RDF JavaScript stack we also started working with other approaches, one implementation is clownface, a simple but powerful graph traversing library inspired by Apache ThinkerPop. The imperative way we use in clownface is very useful in client side JavaScript for example, where we want to create user interfaces. We think there might be room for a standard of a more imperative oriented query language for RDF as well, for such use cases.

Some specific SPARQL issues I have:

- Many SPARQL endpoints have extensions for text search based on Lucene or alike. The API

is pretty much always the same but there is no standard for it, so queries across stores are not portable.

- `IF` can only be used as `IF/ELSE`.
- Handling of RDF Lists is really painful, some stores have ideas/implementations of how this could be improved with some syntactic sugar. It is currently almost impossible to update lists with SPARQL where the amount of elements is not always the same.
- There are some simple functions that are extremely useful in data pipelining but not part of the standard. A good example is the string `split` function implemented in Jena.
- Some things that would be useful in the real world are surprisingly hard. For example getting the top X results that fulfill a certain condition. See for example this archived SemanticOverflow posting (main site is no longer online).
- Property paths only return the start and the end of the path. This makes it impossible to get all the nodes of a chain for example, as it can be found in tree-based structures like directories. So it is cumbersome to fetch all the data needed to properly show such a tree in a user interface.
- Once we would have a way to get the full path we also need a way to represent it in a `CONSTRUCT` query.