From: **Irene Polikoff** irene@topquadrant.com
Subject: Position statement
Date: 14 January 2019 at 03:35
To: group-data-ws-pc@w3.org

IP

I have been working with RDF since 2002. The company I co-founded, TopQuadrant (www.topquadrant.com), is offering information governance solutions built using RDF and associated technology stack. I was a co-chair of the W3C WG that developed SHACL.

The bulk of my position statement focuses on the opportunities for making RDF easier to use by the majority of developers. Having said this, I also want to note that we would like to see changes to RDF that offer better support for reification - to make it easier to add information to statements or edges in the property graph speak. And better support for lists. We see these two requirements as the main remaining items that property graph data models address better than RDF. Further, we believe that a simplified RDF specification should be based on a minimal data structure (nodes, triples, graphs) only, and allow equal usage of open-world and closed-world semantics so that languages like SHACL can truly sit in parallel to RDF Schema and OWL.

Returning to the issues surrounding application developers use of RDF, it is our experience that approaches for accessing RDF data can be the main obstacles to its adoption:

1. Some issues have to do with technical challenges and peculiarities of RDF (e.g., perception or complaint that RDF is hard).
2. Some issues have to do with the mainstream developers wanting to use the tools, techniques and skills they know and that are in are demand in the industry as opposed to developing more niche skills.
3. Finally, applications typically require UI. Thus, for building applications, developers typically need JSON objects (often nested) returned from the server while SPARQL returns flat structures

To address these challenges, we have implemented GraphQL interface to RDF that is driven by the model in SHACL. In our experience so far, this has been a very effective solution. Our developers are reporting meaningful productivity gains with the following factors contributing to the gains:

- Because GraphQL is very popular in the developer world, finding educational resources and tools is very easy. There are a number of libraries to choose from that are very mature and greatly assisted in the dev process (we use graphqljs, GraphiQL and ApolloClient).
- There is no 'maybe' with GraphQL, a class definition exists or it doesn't, data exists or it doesn't. This makes it very straight forward in both schema processing and data processing. As one developer says, "I've learned to rely on this when debugging the 'where is my data/model' issue, if the endpoint can't resolve it, there is something wrong with the model." Another developer says, "The biggest impact for me is on time saved writing and debugging complex SPARQL queries. We no longer need to mess with order of operations/triple patterns to improve performance etc. Along with this, it has simplified the separation between client and server - heavier processing can more easily be pushed to the client, reducing overall server load."
- There is no data transformation. GraphQL was born of the browser, and the browser likes JSON. While queries are not valid JSON, responses will always be valid JSON and because of this we can rely on browser developer tools to inspect/debug payloads.
- GraphQL can process multiple queries in a single request. This feature makes it especially nice when needing information about more than one type of resource.
- The schema can be used to validate queries on the client side before a network request is even attempted.
- Strictly speaking, GraphQL is not a query language, it is a replacement for REST. This means developers can spend more time developing features and less time working on the queries that back them.
- What graph or a combination of graphs is being queried is already defined in the endpoints graphid

The approach we chosen was to use SHACL to generate GraphQL Schemas. One benefit of this architectural decision from the developer perspective is that the approach encourages modelers to make discrete choices when building their schemas. Ultimately, when the model is more concrete the UI can be more structured. It's more difficult to build model-driven UI when a value, for example, can be either a Literal OR a Resource.

Overall, our developers report much less of a learning curve required to work with RDF data and semantic technologies as a result of the introduction of GraphQL. From the 2018 developer's perspective, they just need to know how to work with JSON data, which just about every developer should know at this point. We are using GraphQL for both, read and write/update access.

Within the TQ development team we do not see the full and bigger benefits of GraphQL because most of our code is on the meta level, i.e. generic algorithms that start with introspection. Our queries are produced programmatically. We believe that the real simplicity of GraphQL offers big pays off for writing typical queries against specific models, where you can use auto-complete etc.

Today, there is no standard mechanism for mapping RDF to GraphQL. A future W3C CG could tackle that, which is why we provide our SHACL-based architecture as input. For more details see the documentation links on this page www.topquadrant.com/technology/graphql/ .

Regards,

Irene Polikoff