InfoQ interviews Jans Aasman on how MongoGraph helps NoSQL Developers

MongoGraph from AllegroGraph team brings semantic web features to MongoDB developers. They implemented a MongoDB interface to AllegroGraph database to give Javascript programmers both joins and the semantic web capabilities. Using this approach JSON objects are automatically translated into triples and both the MongoDB query language and SPARQL work against these objects. Another goal of MongoGraph is to make the freetext engine of their graph database easy to search as Solr/Lucene.

AllegoGraph CEO Jans Aasman gave a presentation and talked about working on the level of objects instead of individual triples. InfoQ spoke with Jans about this new approach and how it helps the NoSQL developers.

Infoq: What are the advantages of representing JSON objects as RDF triples in a graph database?

Jans: Well, the most direct answer is that you can use JSON to model complex schemas and then perform complicated joins over your data without writing map-reduce queries. One can approach the JSON objects stored in MongoGraph both as JSON objects (using the MongoDB query language) or as more fine grained RDF triples that allow for complex models and complex joins (using the SPARQL query language). Also you can use all the other advanced features of an RDF Database (aka – TripleStore). One can apply the query language SPARQL or apply rules using mechanisms like SWRL, RIF, or Prolog.

You can also now link the data structures in your application that you represent as JSON seamlessly with RDF triples in the Linked Open Data Cloud.

InfoQ: How do you access the data stored in a MongoGraph type of database?

Jans: A MongoGraph query like the example below will return all books for ‘Jans’ ‘Aasman’ as JSON objects.

```
db.authors.find({firstName: 'Jans', lastName: 'Aasman'})
```

But, assuming that we have a collection of authors, books, publishers and stores, one could also write a join heavy SPARQL query like:

```
select * where {
?x fr:firstName Jans; fr:lastName Aasman ; fr:authorOf ?book .
?book hasPublisher ?publisher .
?store fr:outletFor ?publisher; fr:located 'San Francisco' .
}
```

InfoQ: What are the limitations of using a solution like this?

Jans: Currently we implement 90% of the MongoDB API. However, we do not emulate the clustering mechanisms of MongoDB. For this capability we rely on the clustering mechanisms built in to AllegroGraph.

InfoQ: What are the emerging trends in combining the NoSQL data stores?

Jans: From the perspective of a Semantic Web – Graph database vendor what we see is that nearly all graph databases now perform their text indexing with Lucene based indexing (Solr or Elastic Search) and I wouldn’t be surprised that most vendors soon will allow JSON objects as first class objects for graph databases. It was surprisingly straightforward to mix the JSON and triple/graph paradigm. We are also experimenting with key-value stores to see how that mixes with the triple/graph paradigm.

InfoQ: What best practices and architecture patterns should the developers and architects consider when using a solution like this one in their software applications?

Jans: If your application requires simple straight joins and your schema hardly changes then any RDBM will do.

If your application is mostly document based, where a document can be looked at as a pre-joined nested tree (think a Facebook page, think a nested JSON object) and where you don’t want to be limited by an RDB schema then key-value stores and document stores like MongoDB are a good alternative.

If you want what is described in the previous paragraph but you have to perform complex joins or apply graph algorithms then the MongoGraph approach might be a viable solution.