Data Segmenting in Anzo

Cambridge Semantics Position Paper for the Linked Enterprise Data Patterns W3C Workshop

Cambridge Semantics believes that support for named graphs, careful choice in segmenting data, and some minor extensions to SPARQL are quite sufficient for using Semantic Web software to tackle a wide variety of enterprise data challenges.

Our Anzo software combines data from a variety of diverse sources and then serves it up into familiar user interfaces such as Web dashboards or Microsoft Excel. To support the application of Anzo to broad classes of enterprise challenges, a single Anzo database will often store data for many, many applications. And, of course, in this era of Big Data, many applications will involve millions of data instances or more.

To deal with the diversity and volume of data that our customers encounter, Anzo uses a variety of approaches to segment data and transfer only the required data over the network to client applications.

First, all RDF data is stored in named graphs. Anzo uses named graphs as the primary data-manipulation unit for actions including:

- Versioning and data lineage (provenance)
- Access control
- Notifications (eventing)
- Replication

(Anzo primarily uses the TriG serialization format when serializing named graph data outside of Anzo or when exchanging named graph data with 3rd-party tools.)

To manage large amounts of data, Anzo must decide how to segment data across named graphs. At one extreme, all data could be put into a single named graph (effectively mimicking a triple store); at the other extreme, graphs could consist of a single RDF triple (effectively giving per-triple granularity for things like permissions). While the Anzo tools support a variety of approaches between these two extremes for partitioning data, for *most* solutions the Anzo tools find that the best strategy is simply to put all triples about a particular resource into a single named graph.

The resulting profile for an Anzo database is a system with many, many small graphs. Each company is described in its own named graph; each employee in his/her own graph; each protein in its own graph; etc. In general, this straightforward partitioning scheme corresponds nicely with the granularity at which the operations above (versioning, security, etc.) are logically performed.

For particularly sparse resources (those with few properties), the overhead of placing each resource in its own named graph[1] is significant, and for these cases the software can be configured to place the triples describing multiple resources into a single named graph. In these cases, however, a new problem arises: how do you locate the graph that describes a particular resource (e.g., when following a hyperlink in a Web dashboard)? Anzo provides an extensible *graph resolution framework* that lets solutions use various methods (ranging from syntactic URI inspection to explicit registries to full-database SPARQL queries) to map resource URIs to named graph URIs.[2]

To access data from the server, client applications use either the Anzo replication service or SPARQL queries. The replication service simply pulls down the contents of one or more named graphs into a local replica on the client. This local replica is kept up-to-date with changes to the server data via a JMS-based notification (push) system. Anzo clients can maintain as many local replica graphs as they want, but to keep client memory footprints manageable, they tend to cache only data being actively used by the end user. To this end, it is beneficial for the replication service for data to be segmented into many small named graphs rather than fewer large named graphs.

This segmentation scheme leads to practical considerations for issuing SPARQL queries. The SPARQL standard uses either FROM and FROM NAMED clauses within the query or else default-graph-uri and named-graph-uri parameters in the protocol to let query authors specify the graphs that make up the RDF dataset for query execution. Alternatively, a SPARQL endpoint can query against a fixed RDF dataset. Anzo requires that query authors be able to specify a dataset, particularly given that an Anzo Server typically holds information for many applications. But creating a query that lists millions of graphs in FROM and FROM NAMED clauses (or the equivalent protocol URIs) is impractical, even for queries generated by software applications.

Instead, Anzo extends SPARQL with the concept of a named dataset. Named datasets are lists of Anzo graphs that make up the default graph and the named graphs of an RDF dataset. The definition of a named dataset is itself stored in a graph in the Anzo database. Anzo's SPARQL implementation supports a FROM DATASET clause which is used in the same manner as the standard FROM clauses, but instead references the graphs to query against via reference to the definition as stored in the Anzo database. We find this concept of a named dataset useful for other operations, such as exporting a full set of related data at once or tracking changes to a certain subset of data.

Anzo is used regularly by customers for applications as diverse as manufacturing quality control to supply chain KPI tracking to assay lifecycle management to cross-geography advertising performance reporting. Oftentimes, multiple of these applications store large amounts of data on a single Anzo database. Thoughtful yet straightforward data segmentation is, in our experience, one key to supporting these use cases for our enterprise customers.

---

[1] Anzo maintains one *metadata graph* per named graph, which contains security and provenance information. Anzo does provide capabilities to roll up some of this metadata to the data set level, which is another coping strategy employed.

[2] In the straightforward case of one resource per named graph, Anzo uses the same URI for the resource and for the named graph, which makes graph resolution trivial.