Joseph Carson <jcarson8@fau.edu>
COP 5859 Midterm Exam (Final Project Proposal)

Proposal of a Simple Java Based DCAT Driven SPARQL Engine

In an effort to make government related studies and analytics more accessible to the general public, government agencies in the United States and United Kingdom have adopted open standards for data interchange formats. The intention is to improve our quality of life by making data accessible and understandable in an efficient manner such that acquiring and making use of relevant government statistics isn't prohibitively difficult, opening the throttle on the domain of studies that are possible relating to public health and safety, economics, law enforcement, etc. Quality of life can ultimately be improved by giving the people more visibility into how our governing system performs. To that end, schemas like the Resource Description Framework (RDF) and Web Ontology Language (OWL) have been developed to structure data more semantically in an open ended fashion to allow for reasoning algorithms to understand it. These schemas can often described in a number of different formats, dialects, and locales. In addition to semantic data formats, so much data exists in other legacy formats and has many slow moving adopters depending on those formats, that it just isn't feasible to wait until it's all updated to make use of it. Therefore, a common semantic interface was necessary to aggregate this data across the many different sources and formats out there, such that meaningful information can still be harvested in a common fashion, regardless of where it it's at or how published. This common interface is the Data Catalog Vocabulary (DCAT).

A common problem with DCAT adoption is the fact that it's so open ended on the format of the actual data being searched. The data payloads could be semantically formatted, though they could also be comma separated value (csv) or various different XML and JSON formats. In practice, this data often won't be semantically formatted as it may have been collected long before the open standards were adopted. For a query to be sufficiently useful, it may need to perform logic across many datasets with differing schemas, vocabularies, and semantics. Algorithms may need to be specific and often could be inundated with detail. Apache Jena already provides many different components for building models based on differing formats that can be aggregated together and queried against, but a generic framework must be developed to handle the intermediary task of retrieval and aggregation of all models via DCAT. What follows is a proposal of a simplistic implementation for a reusable Java component suitable for performing SPARQL queries against multiple datasets with differing schemas.

DCAT defines a simple catalog object model that describes metadata about a group of datasets (in practice it's often just one with many distribution formats), including its publisher, modification date, and various other tracking information that may be of use. It's most relevant features are the dataset and distribution objects,

which list each associated dataset, the endpoints they can be downloaded from, and the different formats that each dataset is available in. With this knowledge, the component can determine whether or not it supports understanding one of those formats or decide that it prefers one format over another. The figure below shows a reduced macroscopic view of the DCAT object model, described in JSON-LD format for brevity. Most government systems deliver dataset catalogs in an RDF XML format, but the specification is pushing to adopt JSON-LD instead.

DCAT Object Model

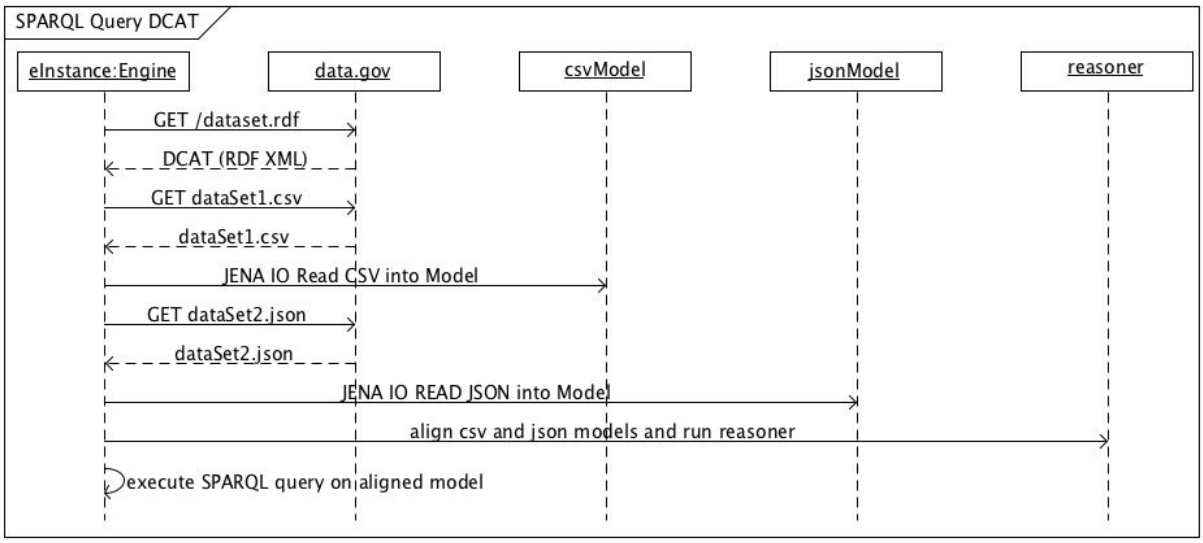
```
CatalogType {
    dataset: [
        DataSetType {
            distribution: [
                DataSetDistributionType {
                    downloadURL: “http://whatever.org/volcaniceruptions.csv”,
                    mediaType: “text/csv"
                }, ...
            ]
        }, ...
    ]
}
```

The DCAT schema specifies that more than one dataset object may exist per catalog document, as the dataset property is a list.[1] In order to abstract away the processing of the different formats, the query engine must internalize the resolution of the format type of different data sets and build a Jena model for each of them using Jena’s I/O formatting features.[2] It’s tough to tell whether or not each dataset in the catalog adheres to a different schema, if so, the query engine itself must support an interface for schema alignment and must invoke the appropriate reasoner to generate an inferred model. Whether this feature is actually necessary or even feasible in the allowed time will require research and more hands on experience. When the engine has resolved all datasets in the given catalog document and built a model for each, the given SPARQL query string can be executed against the aggregate model with the results returned just as they were from Jena’s QueryExecution interface.

This design will have a few limitations. The first of which is that the supported RDF formats are at the mercy of what Jena supports. The reason being that Jena internalizes the parsing of RDF in different formats. The engine would also be limited to the scope of one DCAT document per query. This is largely due to the fact that relevant DCAT document resources must be discovered using another method. Most data sharing systems run server software called the Comprehensive Kerbal Archive

Network (CKAN). The CKAN server software exposes an API that allows for searching for available DCAT catalogs based on query strings. Ideally this SPARQL query engine would be just one component in a larger component that allows for searching multiple URI's resolved based on an input query. Though that would further complicate the matter of schema alignment.

To properly validate the query engine, suitable datasets must be sought out as well. An ideal test case would be a catalog that defined multiple datasets per catalog, each with different distribution formats. Unfortunately, we haven't come across any yet. Testing against this model would be the best case scenario. If possible, I'll try to create one myself just to test the component. The execution timeline should look something like the below sequence diagram. A precise object model still needs to be built based on the best practices of using of Jena and the extensibility of the engine itself.



References

1. Project Open Data Metadata Schema version 1.1. https://project-open-data.cio.gov/v1.1/schema/
2. Working with RDF Streams in Apache Jena. http://jena.apache.org/documentation/io/