Proposal for a DCAT Query Engine

Joseph Carson <jcarson8@fau.edu>

COP 5859 - Final Project - Dr. Ravi Shankar

Abstract

Data distribution systems that are utilizing the Data Catalog Vocabulary (DCAT) for publishing various dataset distributions are driving initiatives to leverage seamless interoperable analysis of not only existing datasets but also various other sources of data, potentially found in the wild, which are likely to be expressed in various different formats and vocabularies. While DCAT makes datasets widely available for consumption in different formats, the distributions themselves aren't inherently related, meaning that many distribution elements in a DCAT document may contain data in different formats other than csv or RDF/XML, and as such their true meaning may be different and or even contextual to data expressed in other DCAT distributions of the same dataset. Consider a potential dataset of real estate foreclosures that contains two distributions, one being a csv listing of foreclosures across Florida and the other a map file generated by Google Maps, which lists all businesses and residences in Florida. The csv distribution will resolve into RDF triples easily while the Google Maps file will be Keyhole Markup Language (KML) and may not have a direct RDF translation. This poses the problem of a single dataset resource being associated with two semantically equivalent but syntactically incompatible resources, requiring an extensive semantic alignment process. What follows is a proposed framework and reference implementation for abstraction of processing common DCAT distribution formats into a single queryable RDF model built on Apache Jena. The reference implementation demonstrates a single query interface to a DCAT document that contains two distributions, a listing of FOAF friends and a trivial equivalent ontology called BLAH.

The DCAT vocabulary organizes multiple data streams, referred to as distributions, associated with a single dataset. These distributions need not be related to one another in format or semantics, yet are asserted to be related to a larger dataset as a whole. Querying each distribution individually is quite trivial, but querying the aggregation of many is complicated by the inherent differences in syntax and semantics of the each. The lack of a common framework for aggregating and aligning dataset distributions leads to error prone and ad-hoc alignment methods in semantic web applications. This paper proposes a common framework for aggregating, aligning and querying the resources in a DCAT document including a set of proposed feature requirements, their implementation, potential use cases, and the limitations of such a component as well as where improvement must be sought. The provided reference implementation performs a simple alignment of a FOAF document and a trivial semantically equivalent format, referred to as BLAH, to demonstrate the intended usage of such a framework through a single Java class interface known as DCATModel.

It's necessary to understand that DCAT represents an aggregation of data streams (or files) that are associated with a particular information base. It's not an arbitrary grouping of all associated data of an entire topic, but rather all information files for a particular instance of a topic. A semantic framework for DCAT must be able to resolve and deconstruct a DCAT document itself. It should support querying the aggregation of its distributions as well as querying against an inference model produced by the combining the aggregation of the distributions with loaded schemas and any arbitrary assertions made on the model. It must also support other common object

oriented design principles including reuse of the component to analyze different DCAT documents, changing the inference assertions, and even querying metadata of the DCAT document itself instead of its distributions. All of which are possible by maintaining separate Jena models for the DCAT document, all associated distributions, ontologies and assertions, and accessor/mutator methods for controlling each.

The reference implementation mimics the standard Jena Model interface without directly implementing it. The primary interface method DCATModel.load() accepts a url that must resolve to a local or remote DCAT document. Since DCAT is inherently an RDF vocabulary (usually RDF/XML), it can be loaded into a DCAT specific model and queried for either the associated accessURL or downloadURL properties, preferably downloadURL as accessURL by definition may require user interaction on a web page. The model can then be preserved for DCAT metadata related queries after each call to load. Each URL must then be loaded into a the dataset model. At this point, the framework must decide whether or not the type of data can be handled by Jena. Jena supports a few different RDF input formats including csv, ntriples, and RDF/XML but also implements a method of extending the framework with new format parsers which Jena refers to as languages. Languages are usually associated with the MIME type of a distribution, but if unavailable it can also be associated to a distribution by one of many file extensions. Attempts to load unrecognized formats into Jena result in an exception being raised, therefore the framework must attempt to resolve the Language associated with the distribution before loading it. The reference framework implements a method for resolving the installed language to each distribution by attempting an http

HEAD request to resolve the MIME type if the url if it's http. Otherwise, Jena's API is used to resolve the distribution url according to the url string itself. If an installed language can be resolved, then Jena can be instructed to invoke the appropriate language to parse the distribution into the dataset model, otherwise it's simply ignored.

In practice, many data distributions found in DCAT models may not be supported by Jena, which ends up being the largest inhibitor to querying the aggregate data. A useful DCAT framework should internally install commonly found distribution languages into Jena as well as allow for the consumer to install their own languages into Jena such that the framework doesn't conflict with the consumer's installed language handlers, allowing for the consumer's installed languages to be transparently detected and used as if the framework installed them. This is exactly what the DCATModel reference implementation does by implementing utility functions to install languages, query if languages are installed, and provide for a class to associate all necessary elements that make up a language configuration including the associated file extensions, MIME type, factory objects, etc. The DCATModel class installs its own language handlers, if those it wants to install aren't already installed. It does this during a static initialization block which is only ever executed the first time the DCATModel class is accessed or an instance is created. If a consumer wants to install their own language handler that DCATModel would usually install at runtime, it needs only to do so before it interacts with DCATModel which won't override any installed languages.

Since translation of complex data to RDF can be considered a field of its own, the reference implementation only provides a small basic DCAT file with two trivial

distributions, one of which is an RDF/XML listing of FOAF objects and the other a semantically equivalent format which specifies people of type BlahPerson identified by the language BLAH. DCATModel installs a language to handle files of type .blah and associated MIME type. The common approach to implementing a language was used, which is to use the *schemagen* tool packaged with the Jena library to read an ontology and generate a Java class that aids in creating the associated RDF types when parsing the input distribution and writing to the model. Each line is read and translated to a group of RDF triples, with each resource being able to be described as a BlahPerson. Since these two languages are syntactically equivalent, they can be aligned according to making BlahPerson an equivalent class of foaf:Person, with other properties being aligned similarly.

Results

A brief test of the reference implementation yields the expected results. The reference implementation test successfully loads both the RDF/XML listing of foaf:Persons and the listing of BlahPerson objects into the dataset model. A few queries are performed, with alignment being manually added to the model between each invocation of the same query, which is essentially, “select the names of all objects that foaf:know #me.” After sufficient alignment has been applied for the type of class, e.g. foaf:knows aligned to BlahPerson:hasFriend, all expected friend objects merged from the people.blah file are returned successfully. These results show that since simple alignment operations can be applied on multiple distributions of languages aren’t

supported by default that a more capable DCAT framework can be derived by extending it in many of the same ways.

Limitations and Next Steps

The primary limitation of the design at the moment is that it doesn't already implement any useful languages. It does however facilitate a framework for installing them and aggregating distributions that use them. Implementing a language that can handle Google Maps and various other data formats that are commonly distributed with DCAT will bring a wealth of possibilities to bridging the gap between tough to align languages. An even more beneficial feature would be to leverage one of the many ontology alignment services in development today including the Align API to help generate automated schema alignments so that the consumer of the framework has to rely less on manually aligning the ontologies that represent the data in the DCAT model.

References

1. An Introduction to RDF and the Jena RDF API. (n.d.). Retrieved July 11, 2015.
2. Reading RDF in Apache Jena. (n.d.). Retrieved July 21, 2015.
3. An Introduction to RDF and the Jena RDF API. (n.d.). Retrieved July 11, 2015.
4. DuCharme, B. (2013). *Learning SPARQL querying and updating with SPARQL 1.1* (2nd ed.). Sebastopol, CA: O'Reilly Media.
5. Data Catalog Vocabulary (DCAT). (n.d.). Retrieved July 2, 2015.
6. Project Open Data Metadata Schema v1.1. (n.d.). Retrieved June 28, 2015.