SPARQL / OData Interop

Kal Ahmed (kal@brightstardb.com, @kal_ahmed)
Graham Moore (gra@brightstardb.com, @gra_moore)

Written as a position paper for the W3C Open Data on the Web workshop:
http://www.w3.org/2013/04/odw/

Intro

We have been working on exposing SPARQL endpoints as OData endpoints. The source code is now available on GitHub at https://github.com/BrightstarDB/odata-sparql. The following is a description of our motivation and some of the technical details about how we are approaching the problem.

At some level there is no doubt that the SPARQL, RDF, Linked Data Platform[1] stack "competes" with the OData stack. Both provide access to generic data models, both attempt to play well as a RESTFul architecture, both offer up query languages for use by remote clients over HTTP. OData does a better job of exposing data as entities and OData support for containers and entity level update is more mature than the LDP effort. RDF on the other hand has a metamodel that is more accessible; the unwavering use of URIs for addressing and identification, both at the level of instance data and the level of ontology is a real benefit as is the inherent ability to merge homogenous data.

However, "competes" is in scare-quotes because it is not useful to think of public open data standards competing with one another with each trying to operate to the exclusion of the other. Rather we should be focussing on the ways in which interoperability can be achieved - as running code but also at the level of the standards-making process. We have been working with both OData and RDF now for many years and trying to help bridge the gap between these web data worlds at the level of running code. Based on that experience, we have started a project to provide an OData endpoint that sits on top of any SPARQL-compliant endpoint. Technically this is a cool thing to play with, but it is also a way to get a feel for issues in open data protocol interoperability that could benefit from further standards work.

The Approach

[1] http://www.w3.org/TR/2012/WD-ldp-20121025/

The technical approach we have taken is to implement a proxy that parses OData operations and rewrites them as equivalent SPARQL operations that can then be executed against a SPARQL endpoint. The SPARQL result set is then parsed and rewritten as an OData result set.

The main hurdle to overcome is that an OData service is driven by the underlying domain model. An OData service exposes its ontology and entity collections as a metadata document[2] with a well-known URI[3], and all queries are expressed in terms of this metadata. For a generic SPARQL endpoint, this is not necessarily the case.

As the service we are exposing is OData we have chosen to approach this problem by making use of the annotations extension point in the OData service metadata document. By taking this approach we enable the OData service that the proxy provides to be defined either:

1. As a manually configured OData service metadata document.
2. By conversion from a known RDF schema or OWL ontology
3. By introspection of the SPARQL endpoint using SPARQL queries that make use of RDF Schema / OWL types and properties.

At present we have not implemented either (2) or (3) but we are fairly confident that some level of useable OData service metadata could be automatically generated in this way. The other potential advantage is that the OData service metadata is simply another resource that can be published, so it would be possible for the owner of a SPARQL endpoint to make an official OData service description available even if they were unwilling/unable to host the OData proxy themselves.

OData Annotations

To get things working we created a small set of annotations that can be used to decorate the model described in an OData service metadata document. The annotations are used to help map OData entity and property types to their equivalent RDF types.

We use the following annotation namespace for all SPARQL OData annotations:

```
<Using Namespace="ODataSparqlLib.Annotations" Alias="Sparql"/>
```

This is just a way of saying that the Annotation *ODataSparqlLib.Annotations.Uri* can be referenced in the metadata document using the short name *Sparql.Uri*.

Identity Prefix Annotation

The Identity Prefix Annotation is used to help map simple between RDF Resource URIs to

[2] http://www.odata.org/media/30002/OData%20CSDL%20Definition.html

[3] http://www.odata.org/media/30002/OData.html#metadatadocumentrequest

OData simple identity attributes. For example in RDF we want to be using a URI such as http://www.brightstardb.com/products/1 where as in the OData entity we want to talk about this as product with Id of '1', and connected to this as */products(1)*.

NOTE: while it would be possible to use the full URI of an RDF resource as its OData identifier, it has implications for consistent URI escaping and for the length of the resulting OData URIs.

To achieve this we use the *IdentifierPrefix* annotation on the property identified as the Key property for the OData EntityType. The following sample show its usage.

```
<EntityType Name="Film">
    <Key>
       <PropertyRef Name="Id"/>
    </Key>
    <Property Name="Id" Type="Edm.String" Nullable="false">
       <ValueAnnotation Term="Sparql.IdentifierPrefix"
                        String="http://dbpedia.org/resource/"/>
     </Property>
</EntityType>
```

In the above example an RDF resource with a URI like http://dbpedia.org/resource/Un_Chien_Andalou can be referenced through the OData proxy as http://example.org/odata/Films('Un_Chien_Andalou')

The identity prefix mapping is specific to an OData entity type, so each type can use a different prefix if required.

Entity Type Mapping Annotation

To indicate how types in OData map to types in RDF we use a *Uri* annotation on the OData EntityType definition. In this context the Uri annotation simply provides the full URI of the RDF resource that defines the entity type.

The following sample shows its usage:

```
<EntityType Name="Person">
  …
  <ValueAnnotation
    Term=”Sparql.Uri”
    String=”http://mappings.dbpedia.org/server/ontology/classes/Person” />
</EntityType>
```

We also allow a default namespace URI for entity types to be defined in the proxy configuration file, any EntityType without an explicit mapping receives a mapping based on appending the EntityType name to the namespace URI. We allow for different string case mappings to also be applied when resolving the name to a URI - e.g. force to lower/upper case; force to lower/upper

camel-case.

Literal Property Type Annotation

A similar approach is used to map OData properties to RDF properties.

```
<Property Name="Name" Type="Edm.String" Nullable="true">
       <ValueAnnotation Term="Sparql.Uri" String="http://dbpedia.org/property/name"/>
</Property>
```

Again, we also allow a default namespace URI for property types to be defined in the proxy configuration file (separate from the default namespace for entity types), any Property without an explicit mapping receives a mapping based on appending the Property name (with case conversion applied) to the namespace URI.

Association Property Type Annotations

In OData, properties that reference other entities are described by a NavigationProperty that defines a traversal of a separately defined Association type. This allows OData to support bi-directional traversal of the relationships between entities. In RDF resource to resource relationships are directed. To accomodate OData's bidirectionality we introduce an additional *IsInverse* annotation which can be used in conjunction with a *Uri* annotation to specify both the RDF property type and the direction in which the property is traversed (subject-to-object or object-to-subject).

```
<EntityType Name=”Place”>
  <NavigationProperty Name=”BirthPlaceOf”
                      Relationship=”DBPedia.Person_BirthPlace”
                      FromRole=”BirthPlace” ToRole=”Person”>
    <ValueAnnotation Term=”Sparql.Uri” String=”http://dbpedia.org/ontology/birthPlace”/>
    <ValueAnnotation Term=”Sparql.IsInverse” Boolean=”True” />
  </NavigationProperty>
</EntityType>
<Association Name="Person_DeathPlace">
  <End Role="Person" Type="DBPedia.Person" Multiplicity="*"/>
  <End Role="DeathPlace" Type="DBPedia.Place" Multiplicity="1"/>
</Association>
```

Note that the Association definition is currently un-annotated as all the necessary information is conveyed by the annotations on the NavigationProperty.

Once again, the NavigationProperty Name can be combined with the default ontology base URI specified in the proxy configuration to avoid the need to provide explicit *Uri* annotations.

A Larger Example

Putting this all together here is a complete OData service metadata document with annotations that we can use to expose a subset of DBPedia as OData.

In this example, a base type namespace URI of http://dbpedia.org/ontology/ and a base property namespace of http://dbpedia.org/property/ is used and names are mapped to URI components by forcing them to lower camel-case. In this example, this means that many properties and entity types do not require a *Uri* annotation to be mapped.

```
<?xml version="1.0" encoding="utf-8" ?>
<edmx:Edmx xmlns:edmx="http://schemas.microsoft.com/ado/2009/11/edmx" Version="3.0">
  <edmx:DataServices
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
m:DataServiceVersion="2.0">
    <Schema xmlns="http://schemas.microsoft.com/ado/2009/11/edm" Namespace="DBPedia">
      <Using Namespace="ODataSparqlLib.Annotations" Alias="Sparql"/>

      <!-- Thing: http://www.w3.org/2002/07/owl#Thing -->
      <EntityType Name="Thing">
        <Key>
          <PropertyRef Name="Id"/>
        </Key>
        <Property Name="Id" Type="Edm.String" Nullable="false">
          <ValueAnnotation Term="Sparql.IdentifierPrefix"
String="http://dbpedia.org/resource/"/>
        </Property>
        <ValueAnnotation Term="Sparql.Uri" String="http://www.w3.org/2002/07/owl#Thing"/>
      </EntityType>

      <!-- Work: http://dbpedia.org/ontology/Work -->
      <EntityType Name="Work" BaseType="DBPedia.Thing">
        <!-- Title: Gets default URI mapping: http://dbpedia.org/property/title -->
        <Property Name="Title" Type="Edm.String" Nullable="true"/>
        <!-- Director: Gets default URI mapping: http://dbpedia.org/property/director -->
        <NavigationProperty Name="Director" Relationship="DBPedia.Work_Director"
FromRole="Work" ToRole ="Director"/>
      </EntityType>

      <!-- Film: http://dbpedia.org/ontology/Film -->
      <!-- Derived from Work -->
      <EntityType Name="Film" BaseType="DBPedia.Work">
        <Property Name="Name" Type="Edm.String" Nullable="true">
          <!-- Not strictly necessary, but shown as an example -->
          <ValueAnnotation Term="Sparql.Uri" String="http://dbpedia.org/property/name"/>
        </Property>
        <Property Name="Runtime" Type="Decimal" Nullable="true" />
        <Property Name="ImdbId" Type="Edm.String" Nullable="true"/>
```

```
        <!-- Not strictly necessary, but shown as an example -->
        <ValueAnnotation Term="Sparql.Uri" String="http://dbpedia.org/ontology/Film" />
      </EntityType>

      <!-- http://mappings.dbpedia.org/server/ontology/classes/Person -->
      <EntityType Name="Person" BaseType="DBPedia.Thing">
        <Property Name="Name" Type="Edm.String" Nullable="true">
          <!-- Here we use FOAF vocab for a property -->
          <ValueAnnotation Term="Sparql.Uri" String="http://xmlns.com/foaf/0.1/name"/>
        </Property>
        <Property Name="BirthDate" Type="Edm.DateTimeOffset" Nullable="true">
          <ValueAnnotation Term="Sparql.Uri"
String="http://dbpedia.org/ontology/birthDate"/>
        </Property>
        <Property Name="DeathDate" Type="Edm.DateTimeOffset" Nullable="true">
          <ValueAnnotation Term="Sparql.Uri"
String="http://dbpedia.org/ontology/deathDate"/>
        </Property>
        <NavigationProperty Name="BirthPlace" Relationship="DBPedia.Person_BirthPlace"
                            FromRole="Person" ToRole="BirthPlace"/>
        <NavigationProperty Name="DeathPlace" Relationship="DBPedia.Person_DeathPlace"
                            FromRole="Person" ToRole="DeathPlace"/>
        <NavigationProperty Name="RestingPlace"
Relationship="DBPedia.Person_RestingPlace"
                            FromRole="Person" ToRole="RestingPlace"/>
      </EntityType>

      <!-- Place: http://dbpedia.org/ontology/Place -->
      <EntityType Name="Place" BaseType ="DBPedia.Thing">
        <Property Name="Abbreviation" Type="Edm.String" Nullable="true"/>
        <Property Name="Abstract" Type="Edm.String" Nullable="true"/>
        <Property Name="AnnualTemperature" Type="Edm.Decimal" Nullable="true"/>
        <Property Name="Elevation" Type="Edm.Decimal" Nullable="true"/>
        <Property Name="PopulationTotal" Type="Edm.Int32" Nullable="true"/>
      </EntityType>


      <Association Name="Work_Director">
        <End Role="Work" Type="DBPedia.Work" Multiplicity="*"/>
        <End Role="Director" Type="DBPedia.Person" Multiplicity="1"/>
      </Association>
      <Association Name="Person_BirthPlace">
        <End Role="Person" Type="DBPedia.Person" Multiplicity="*"/>
        <End Role="BirthPlace" Type="DBPedia.Place" Multiplicity="1"/>
      </Association>
      <Association Name="Person_DeathPlace">
        <End Role="Person" Type="DBPedia.Person" Multiplicity="*"/>
        <End Role="DeathPlace" Type="DBPedia.Place" Multiplicity="1"/>
      </Association>
      <Association Name="Person_RestingPlace">
```

```
        <End Role="Person" Type="DBPedia.Person" Multiplicity="*"/>
        <End Role="RestingPlace" Type="DBPedia.Place" Multiplicity="1"/>
      </Association>

      <EntityContainer Name ="Contents" m:IsDefaultEntityContainer ="true">
        <EntitySet Name="Films" EntityType="DBPedia.Film"/>
        <EntitySet Name="Persons" EntityType="DBPedia.Person"/>
        <EntitySet Name="Places" EntityType="DBPedia.Place"/>
        <AssociationSet Name="Film_Director" Association="DBPedia.Film_Director">
          <End Role="Film" EntitySet="Films"/>
          <End Role="Director" EntitySet="Persons"/>
        </AssociationSet>
      </EntityContainer>

    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

Request Transforms

Typical OData requests are for either a single entity, a set of entities or a select format that retrieves a result that appears as a datatable. To map this into a SPARQL query we use the above annotations and then depending on the target result use either the SELECT or CONSTRUCT keywords.

OData select queries work in a similar fashion to SELECT in SPARQL by returning a tabular result. So we generate a SPARQL SELECT query and map the resulting SPARQL table to an OData results set.

When processing OData queries that result in an entity or collection of entities, we use SPARQL CONSTRUCT queries. This allows us to retrieve an RDF graph from the SPARQL endpoint that represents the entities, their properties and relationships. The graph is then processed by the proxy into a list of entities and, if necessary the list is sorted by the sort criteria specified in the original OData query.

Update

For now update is out of scope for this project, but it could potentially be implemented using the same set of annotations we use for read. The main difference is that we need to make use of SPARQL UPDATE, which is a separate specification and more importantly usually lives in a different place on the server, most likely, behind the firewall. We see the initial benefit of this work as opening up existing RDF triple stores with public SPARQL endpoints to OData applications. Very few of these, for obvious reasons, offer a writeable SPARQL update endpoint. For enterprises with RDF stores the additional update option may be of interest.

Future work

In its current state, the library is best described as a minimal proof of concept with basic support for selecting entities by their ID or with a few simple property operators such as Equals and GreaterThan. We also have some basic sorting implemented.

The main work to do is to create a complete implementation of all OData path semantics, filters options, and projection operators. In addition we need to find a way of implementing OData paging - ideally without requiring the proxy to retrieve the complete unpaged set of entities from the SPARQL endpoint. Finally we would like to provide tools for generating an OData service metadata document from a SPARQL endpoint or its ontology and to generate / write mappings for some of the existing public SPARQL endpoints.

Discussion Points

- Proxying as a way to bridge open data standards. Seems to make sense for read/query, but perhaps less so for write/update. Is this going to be a problem in the long run ?
- OData / Linked Data Platform interop - this seems like it could be an easier path to follow, but does it make sense for the development of these standards to happen in isolation. What could LDP learn from OData and vice-versa ?
- Is there a meta-model for an entity-oriented view of resources that both OData and LDP could share ?