Using OData to access an RDF triple store through a semi-structured conceptual model

Linking Structured Data

Michael Hausenblas (DERI)
Michael Kerrin (DERI)
Michael Pizzo (Microsoft)
Evelyne Viegas (Microsoft Research)
Neil Wilson (The British Library)

Whitepaper

Linking Structured Data

Using OData to access an RDF triple store through a semi-structured conceptual model

December 2012

Michael Hausenblas (DERI)
Michael Kerrin (DERI)
Michael Pizzo (Microsoft)
Evelyne Viegas (Microsoft Research)
Neil Wilson (The British Library)

1 Overview

In this paper we explore how OData can be used to expose data within an RDF triple store through an end-user oriented model, and consumed by a broad range of consumer-oriented tools and applications.

The Semantic Web is all about understanding the data of the web. Not just HTML, but data. RDF describes the web by breaking it into atomic components – individual subject, predicate, object tuples that represent the individual facts. A person is represented as a set of tuples with a common subject identifier, each with a predicate and object representing an individual fact about the person such as Name, Age, Address, personal website, and so forth. This triple model provides a very flexible solution for representing the web as a graph of interconnected facts. The collection of people within the triple store may be implicitly represented by the collection of all tuples that have a common set of predicates. The atomic nature of RDF makes it extremely flexible for storing any facts or relationships.

Capturing an entity model implicitly represented within an RDF triple store and exposing through an end-user oriented conceptual model enables easier use and consumption of the data by targeted applications, common tools, and higher level libraries. In fact, a service author may choose to apply an arbitrary set of customizations when mapping an RDF store to a conceptual model in order to optimize the user-oriented perspective of the data toward one or more common usage scenarios.

OData, a RESTful web-based technology for modeling, querying, and navigating structured and semi-structured data, provides just such a user-oriented conceptual model. With a common goal as the Semantic Web, OData seeks to exposes the *data* of the web as resources that can be addressed, related, queried, and described through common semantics.

2 Real-world scenario: British Library

The British Library is the government funded national library of the UK with responsibility for globally disseminating bibliographic metadata for its collections as well as all new UK publications since 1950 via 'The British National Bibliography' (BNB). Originally the BL's metadata services were operated on a commercial basis and primarily aimed at the library community.

2.1 Open metadata for public access

However, in 2010 the Library began a new open metadata strategy designed to remove barriers such as restrictive licensing, proprietary formats and access standards that prevented wider use of its descriptive metadata. The move was a response to the growing call from government and the open data movement for increased public access to data to stimulate innovation and enable new forms of collaboration and knowledge creation. Details of the BL's open data services are available elsewhere[1].

2.2 Joining the Linking data cloud

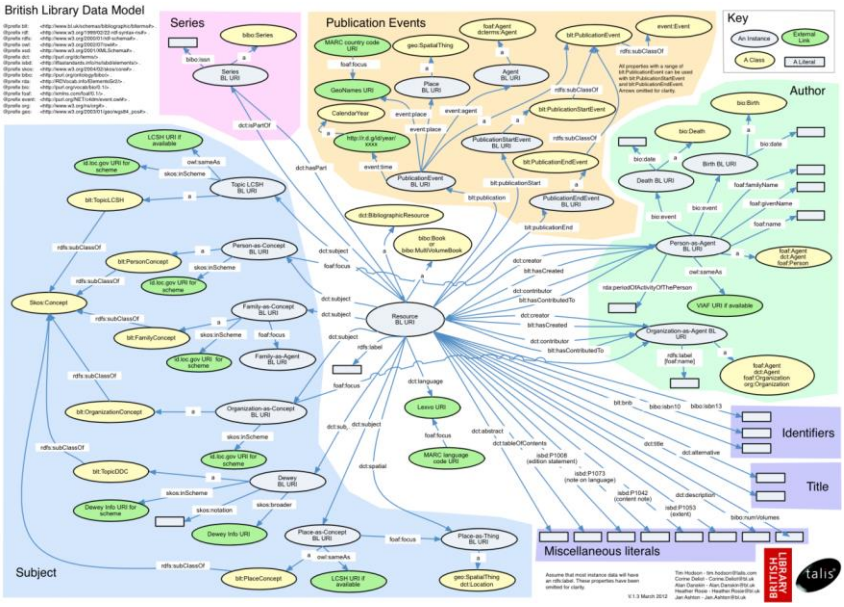
One of the BL's first steps was to re-model its data from the library MARC (MAchine Readable Cataloguing) standard to RDF and then to begin to create a linked data representation with links to six other linked data resources (e.g. Geonames) as a precursor to building an RDF triple store (with a SPARQL endpoint[2]) for the three million record BNB. A representation of the new model is shown below and proved to be highly influential among the library community in beginning to move debate from theory to practice[3].

---

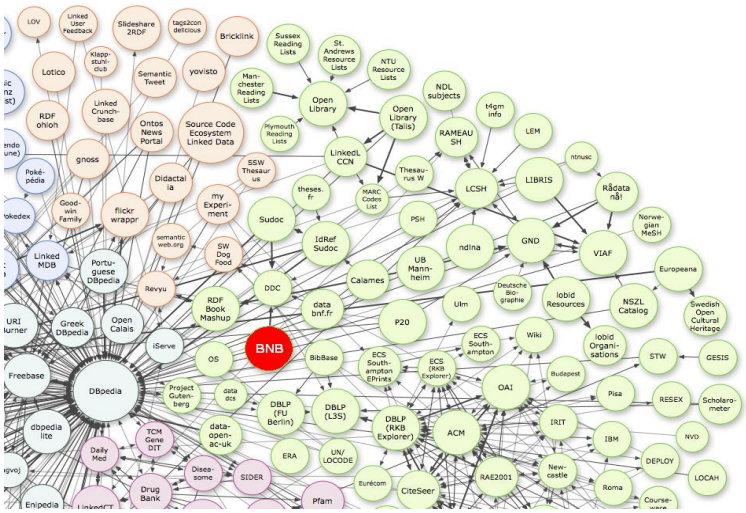
[1] http://www.bl.uk/bibliographic/datafree.html

[2] http://bnb.data.bl.uk/sparql

[3] http://www.bl.uk/bibliographic/pdfs/bldatamodelbook.pdf



The BL believes open metadata is a logical evolutionary step for the long established library principle of freedom of access to information. The creation of a global pool of easily reusable, and semantically rich metadata elements would enable the BL and similar knowledge organizations to concentrate increasingly scarse resources on adding unique value. Participation in the development of such infrastructure should also enable the improved integration and visibility of institutional holdings and begin to take the possibilities indicated by the current linked open data cloud to new levels of utility.



2.3 Enabling commercial use of BL collections

In addition to technical innovation, a vital element of the BL's open strategy was collaboration with commercial partners, researchers and developer groups to promote experimentation with the metadata to both increase its community value and improve access to information. It was this strategic objective that led to the BL's involvement with Microsoft and DERI. The BL was particularly interested in exploring three areas where its expert partners might offer unique and valuable insights:

- How to build upon the initial work with RDF and enable wider developer communities to manipulate and consume BL metadata in new and flexible ways?
- How to create new forms of visualisation of the metadata to stimulate imaginative new uses and greater interest in the BL's resources?
- How to use new techniques to further enrich the BL's basic descriptive metadata with further related information from open online resources?

The technical results of the collaboration are described below and promise to offer some useful indicators for further work in these areas.

3 Making Linked Data More Broadly Consumable

A lot of data, like the British Library metadata, is being published on the Web according to the Linked Data principles[4]. We refer to this data as the Linked Open Data cloud (LOD)[5]. However, as of today there are not many common tools and consumer-oriented applications to browse, query, and analyze this RDF-based data.

---

[4] http://www.w3.org/DesignIssues/LinkedData.html

[5] http://lod-cloud.net

Companies such as Microsoft, on the other hand, have a number of high quality tools to perform a variety of useful analysis tasks, with wide deployment bases and a large number of users already familiar with the interfaces. They also have rich, integrated development environments for working with external sources of data. The fact that these broadly adopted tools and applications cannot directly import and/or access the RDF-based Linked Data represents a limitation we would like to overcome.

Enter OData.

3.1 Using OData to Expose Consumer-Oriented views of Data

OData[6], short for The Open Data Protocol, is a set of well documented conventions that build on REST principles and industry standards such as JavaScript Object Notation (JSON)[7], XML, and IETF's Atom Publishing Protocol (AtomPub)[8].

OData is based on an Entity-Relational model that allows service implementers to help consumers visualize and consume the data.

OData data is inherently linked, and linkable. OData uses URLs to address collections of resources, resource instances, relationships between resources, and even properties of resources.

Now under standardization at OASIS and supported by a growing ecosystem of tools, applications, development environments, and libraries, OData enables modeling, querying, and navigating structured and semi-structured data.

---

[6] http://odata.org

[7] http://json.org/

[8] http://tools.ietf.org/html/rfc5023

Some of the applications that can consume OData are:

- PowerPivot

  PowerPivot is an add-on to Microsoft Excel that enables clients to perform local sorting, filtering and pivot analysis on data through the familiar Excel user interface. PowerPivot supports the use of OData to retrieve data from external web sources.

- Sesame browser

  The Sesame browser client allows you to browse through data exposed by OData services by searching, navigating, viewing, and paging through the data. You can view the feed in a tabular view, you can view the raw OData feed as XML, or you can superimpose data that includes spatial information over a map view.

- Tableau

  Tableau desktop is a business intelligence tool from Tableau Software that uses breakthrough technology from Stanford University that lets you drag and drop to analyze data. You can use it to connect to OData sources in a few clicks, then visualize and create interactive dashboards with a few more clicks.

- ODataExplorer

  ODataExplorer is a web application that allows you to dynamically explore an OData Service; viewing and querying collections and navigating relationships.

OData is also well supported by client and server libraries on a variety of platforms including Java, JavaScript, PHP, Ruby, Silverlight and .NET. Client libraries make it easy for developers to write custom applications on the web, personal computer, or on Apple, Android or Windows Phone devices to query and update data exposed through an OData service.

4 Implementation

In order to better understand how RDF data could be exposed and consumed by OData clients in a real world example, we sought to identify a set of interesting RDF data, in this case the British Library data, and see if we could consume that data through the Microsoft Excel PowerPivot add-in as an OData feed.

Our solution took British Library data, already exposed as RDF triples, and applied a user-defined map to generate OData feeds corresponding to a fixed schema. Generalizing this approach identifies a few interesting components that could be used in exposing an OData model over an arbitrary RDF triple store:

- A language for describing a mapping from a triple store to an end-user oriented OData entity model.
- An algorithm for generating a default model and mapping from a set of RDF triples
- A process for extracting data and relationships from an RDF triple store according to a defined model

4.1 A Generalized RDF2OData converter

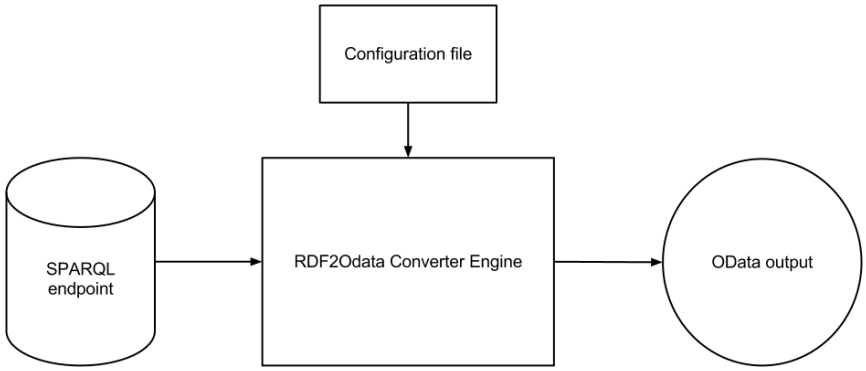
To explore the design space and better understand possibilities and limitations, we have developed an RDF2OData converter that takes RDF-based Linked Data and maps it to OData. This way, we can take advantage of the consumer-oriented model and rich tool set offered by the OData ecosystem.

There are several implementation models for exposing the RDF triples as OData:

- Generate static documents formatted according to OData.
- Expose an OData service over a schematized store populated from an RDF triple store.
- Expose an OData service that dynamically evaluates model-oriented requests against data stored in the triple store, for example by converting OData requests to SPARQL queries.

In this initial phase we took the first approach of defining a user-defined mapping file to convert the RDF triples to a set of static documents, formatted and exposed according to the OData specification. Because our target scenario was consuming the data in PowerPivot, which has minimal requirements on the functionality of the OData Service, this level of implementation seemed sufficient to satisfy our initial exploration.

As a starting point, we assume that the RDF dataset can be accessed via a SPARQL endpoint, and perform the conversion as shown in the following diagram.



The converter takes as input a configuration file and a SPARQL endpoint. It reads the data from the SPARQL endpoint and converts it into the OData Atom format according to the rules described below.

4.2 Basic, single feed

In order to generate a simple, single feed of OData entities from the entire RDF graph, the converter works as follows:

1. Resources in the RDF graph get converted to OData entities (represented as an ATOM entry in the OData ATOM format).

2. RDF statements whose object is a RDF Literal get converted to properties in the corresponding entity. The name of the property in the OData data model is either configured in the configuration file or determined by parsing the path in the predicate URI and taking the last path name as the name of the property.

The configuration file may look like so:

```
{
    'map': {
            'http://purl.org/dc/terms/title': 'title'
    }
}
```

This will map the Dublin Core *title* predicate to the **title** key.

3. RDF statements whose object is a URI reference get converted to relationship links from the corresponding entity to the object value. The rel attribute of the link is set to the statement predicate value.

Example

For example, take the following RDF data – derived[9] from the British Library LOD dataset:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix blterms: <http://www.bl.uk/schemas/bibliographic/blterms#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix bibo: <http://purl.org/ontology/bibo/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix : <http://bnb.data.bl.uk/id/resource/>

:007387664  rdf:type        dct:BibliographicResource,
                            bibo:Book ;
            dct:language <http://lexvo.org/id/iso639-3/eng> ;
            rdfs:label "The strongest man in the world / Ann Spano" ;
            blterms:bnb "GB7609547" ;
            dct:subject
            <http://bnb.data.bl.uk/id/concept/ddc/e18/823.91> ,
                    <http://bnb.data.bl.uk/id/concept/lcsh/Childrensstories>
        ;
            owl:sameAs <http://bnb.data.bl.uk/id/resource/GB7609547> ;

.
```

[9] http://bnb.data.bl.uk/doc/resource/007387664.ttl

Running the above RDF data through the converter, with the above configuration, yields the following OData Atom entry:

```
<entry xmlns="http://www.w3.org/2005/Atom"
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
>
  <id>http://bnb.data.bl.uk/id/resource/007387664</id>
  <title>The strongest man in the world</title>
  <link rel="http://purl.org/dc/terms/subject"
href="http://bnb.data.bl.uk/id/concept/lcsh/Childrensstories" />
  <link rel="http://purl.org/dc/terms/subject"
href="http://bnb.data.bl.uk/id/concept/ddc/e18/823.91" />
  <content type="application/xml">
    <m:properties>
      <d:Id>http://bnb.data.bl.uk/id/resource/007387664</d:Id>
      <d:bnb>GB7609547</D:bnb>
      <d:title>The strongest man in the world</d:title>
    </m:properties>
  </content>
</entry>
```

4.3 Multiple feeds

We can extend the configuration file to enable us to configure multiple OData feeds to come out of the converter from a single RDF graph.

This can make it easier to use the tools mentioned above. Each feed will generally correspond to a concept (represented as an entity type) that the end user knows about. For example for the British Library dataset we generated feeds for Books, Authors, Publication Events, and Subjects.

To do this we add a "feeds" key to the configuration file. The "feeds" value is a list of SPARQL selects to make against the source. One of the variables created by the SPARQL query must be called entry and it must be a resource in the RDF graph that we want to generate an OData entry from.

For this, we can extend the configuration file to include the following:

```
'feeds': {
        'book': '?entry rdf:type < http://purl.org/ontology/bibo/Book>',
        'authors': '?entry rdf:type <AUTHOR_URL>',
}
```

This will generate two feeds from the source. The converter will run the SPARQL query for each configured feed and for every result it will look up the entry variable and generate one OData entity from this value via the process above. If we have two complicated SPARQL queries in our configuration and some resources end up in two feeds then we generate one entry in each feed via the process above.

4.4 Metadata

OData services are described by an Entity Data Model (EDM). Common Schema Definition Language (CSDL) defines an XML-based representation of the entity model exposed by an OData service. OData clients can request the metadata of the service they provide via the */$metadata* URL. Many of the clients mentioned above require that a metadata document is provided for them to work.

After converting the RDF data to OData the converter will generate a metadata file for the data that it has just created. The converter will assume a default **Edm.String** type for each data model property. This will be assumed to be nullable, and will have no fixed length. All the clients will then be able to read this metadata document and understand the basics of the OData model.

We will be able to customize this via the configuration file. So if we know that a certain property has a certain type that we want to expose to the clients this can be done via the metadata file. Within the **map** section of the configuration file where we map RDF predicates to property names we will also be able to specify the type of the property.

For example:

```
'map': {
        'http://example.com/age': {
                'name': 'age',
                'prop': {
                        'type': 'Edm.Int32',
```

```
                    'nullable': 'true'
                }
            }
        }
```

This will map the http://example.com/age predicate to the **age** property. This is all straight forward like above. But we are also specifying a prop key. The value of this key is dictionary containing all the attributes of the property type.

When the metadata document is generated it will contain the following property:

```
<Property Name="age" Type="Edm.Double" Nullable="true"/>
```

The mapping file used above defines a filter of the dataset, which we include in the generated feeds. For example with the British Library dataset we want to generate Book, Author, Publication, etc. feeds from the dataset. So when generating the Book feed we don't want the author information showing up in it. This flexibility allows mapping file authors to expose customized views over whatever subset of data they wish, while omitting unmapped data.

5 Findings

This initial investigation enabled us to better understand some of the approaches, implementation options, and potential issues in exposing RDF triples as an OData service.

5.1 Implementation Options

In our investigation we encountered a number of different implementation options, the most significant of which are described below.

For an initial implementation we generally chose the most direct route for our particular scenario; getting data into PowerPivot, which requires very little in terms of functionality from the OData service. Building a service to support a broader range of clients would likely have led to different implementation choices along the way.

In particular, converting the RDF to a static OData document, while it works well for PowerPivot, doesn't take into account that OData is more than just a data format. It is a full protocol that enables discovery of new data via the metadata features, filtering and searching within large datasets, and data modifications where appropriate. Supporting operations like filter and paging through simple processing of results prior to returning to the user, by putting the converted data into a store, or dynamically generating SPARQL queries from the OData requests, would greatly increase the set of scenarios for consuming the OData service.

5.1.1 Referencing Individual Entities

OData defines conventions for referencing individual entities within a model using key properties of each entity. These references are explicitly specified through edit-links (used for updating or deleting) or self-links (use for retrieving) contained within each entity.

Because our initial implementation is read-only, and does not support retrieving a single entity, we omit these links from the entries. This did not have a significant impact on our target scenario of consuming the data in PowerPivot, but does limit the general applicability of the solution.

A first step at addressing this would be to support the simple OData syntax for requesting an entity by ID, for example as:

```
http://vmdai05.deri.ie/category.xml/Books(007387664)
```

Having a way to address individual entities would allow us to include that request URL as the self-link in the atom entries.

A simple implementation could search through the document for the entity with the specified ID, perhaps aided by an index, and return just that entity as a single OData Atom entry. A more robust implementation might load the data into a store that supported richer query semantics, or convert the request into a dynamic SPARQL query directly against the RDF source.

5.1.2 Exposing Relationships

For RDF statements whose object is a URL, our translator generates simple ATOM links with rel values equal to the predicate value of the statement. These links, while legal in ATOM, do not contribute to the OData model. So, for example, when reading Books, Authors, Publication Events, and Subjects from the RDF graph we do not create relationships that allow OData navigation between the constructed entities.

While PowerPivot allows the user to define relationships based on values within the data, the inherent connectedness of the data is lost by not exposing these relationships through the OData model.

5.1.3 Reducing the Amount of Data Returned Per Request

OData defines a rich query syntax for filtering, ordering, paging, and limiting the set of fields returned in each entry. The ability to page or filter data is particularly useful for large data sets, like the British Library, which are better retrieved in chunks or according to user-defined criteria.

OData defines a server-driven paging model through providing a "next link" at the end of a partial result to fetch the next set of partial results. OData clients know to follow these links in order to retrieve large results in a number of smaller chunks.

OData also defines $top and $skip query options to allow the client to page through results. Our implementation could be extended, again perhaps aided by an index, to support these options.

Simple filtering by evaluating the entries against a simple criteria expressed through the $filter query option prior to returning them could limit the number of results returned to the client. Similarly, the set of fields returned per request could be filtered according to a specified $select OData query option.

5.1.4 Choice of OData Formats

OData defines two formats; one based on ATOM and one based on JSON.

Because the data model for the two formats is the same, we could have chosen an implementation that could return either format once converting the RDF triples to a common OData Entity Model. The OData Atom format is the most broadly used today, and supported by PowerPivot, so it was a natural choice for our implementation.

5.2 Issues

Applying our solution to the target scenario raised a set of issues described below.

5.2.1 Schema changes

As an RDF dataset evolves over time statements are added and removed.

Because OData exposes a model to the client, changing that model due to added or removed statements can affect clients that rely on that model.

In PowerPivot this was a problem because if a new column was added then in order to see this new data we needed to delete and re-import the feed. This became problematic if a user manually created many relationships from or to this feed. This was particularly problematic since we were evolving the schema at the same time we were building reports, and would be less of an issue once a stable mapping had been defined.

We could have chosen to expose newly added statements as "dynamic properties". Dynamic properties appear as regular properties in the payload but are not advertised in $metadata. While modeling properties in $metadata makes the individual properties more discoverable, dynamic properties provide a level of schema flexibility.

Where the schema of an OData service changes in backwards incompatible ways, services are encouraged to version the endpoint. Thus, for example, we could have chosen the following version-specific service endpoint:

http://vmdai05.deri.ie/V1/category.xml

A change to the schema could lead us to define a new service endpoint:

http://vmdai05.deri.ie/V1/category.xml

A client that was not strongly bound to a particular schema (for example, a dynamic OData browser) could query a version-independent endpoint and be redirected to the latest version through a standard HTTP redirect.

5.2.2 Predicates with multiple values

Some predicates in the RDF dataset, like the DC subject, can be present multiple times for the same resource. For example a book can be about multiple subjects.

In the converter mentioned here, we add the subjects in as ATOM links with a custom rel attribute. OData clients like PowerPivot don't recognize arbitrary link elements that are not part of the model (i.e., are not modeled as navigation properties or named streams).

To get around this we added new columns with the names subject1, subject2, etc. for each subject. This introduced some problems in trying to create relationships inside PowerPivot. We couldn't automatically create the relationship so in certain circumstances the end user would have to create lots of relationships, one per column.

Exposing multi-valued statements whose objects were URIs as relationships in the OData Entity Model would have allowed general OData clients to understand and navigate these relationships as first class concepts without having to create duplicate columns. Multi-valued statements whose object are not URIs, could be exposed through a single collection property within the individual entity.

5.2.3 Processing

Sometimes there is a need to process the data as we are converting it. For example the need arose in the British Library dataset to split up the Library of Congress Subject Headers into multiple subjects based on a rules internal to the British Library. Generalizing such processing would require additional syntax in the configuration file, as well as processing logic in the converter.

6 Final Conclusions and future directions

In this paper we looked at how data within an RDF triple store can be exposed through an end-user oriented application model and consumed using OData; a technology for modeling, querying, and navigating structured and semi-structured data through a RESTful, web-based interface.

We went from theory to practice by focusing on some scenarios from the British Library which publishes its metadata on the Web according to the Linked Data principles[10]. However, as of today there are not many common tools and consumer-oriented applications to browse, query, and analyze this RDF-based data. This is exactly this limitation that this paper tries to overcome by allowing rich and integrated development environments to directly import and/or access the RDF-based Linked Data.

In this paper, we have covered the read-only case showing promising results on how OData can be used with an RDF triple store. Further explorations should look into the following:

- Support for addressing individual entities within the OData service.
- Support for defining first class relationships between extracted entities.
- Support for functionality like paging, filtering, and select to reduce the amount of data returned to the client.
- Support for additional operations like expand, ordering
- Support for read-write functionality over the data (cf. also LDP/SPARQL update).
- Enhanced support for automatically generating a model from the RDF triple store.
- Enhanced mapping support for customizing how certain data is handled.

[10] http://www.w3.org/DesignIssues/LinkedData.html

- Use of RDF schema to enhance the creation of the OData Entity Model and/or annotate the model with common semantics.

Some approaches to support this functionality include:

- Building simple filter/paging support processing into the service when returning the data.
- Exposing an OData service over a schematized store populated from an RDF triple store.
- Exposing an OData service that dynamically evaluates model-oriented requests against data stored in the triple store.
- Explore the ability to automatically generate a model from an RDF triple store, similar to the RDB2RDF Direct Mapping.
- Explore the role of schema.org as the top-level vocabulary

We believe many real-world applications can use OData to access RDF triple stores.