Using read/write Linked Data for Application Integration – Towards a Linked Data Basic Profile

Arnaud J Le Hors
IBM
Software Standards Architect
+1 (720) 396-5228
lehors@us.ibm.com

Martin Nally
IBM
CTO, Rational Software Fellow and VP
+1 (714) 472-2690
nally@us.ibm.com

Steve K Speicher
IBM
STSM
OSLC Lead Architect
+1 (919) 254-0645
sspeiche@us.ibm.com

ABSTRACT

Linked Data, as defined by Tim Berners-Lee's 4 rules [1], has enjoyed considerable well-publicized success as a technology for publishing data in the World Wide Web [2]. The Rational group in IBM has for several years been employing a read/write usage of Linked Data as an architectural style for integrating a suite of applications, and we have shipped commercial products using this technology. We have found that this read/write usage of Linked Data has helped us solve several perennial problems that we had been unable to successfully solve with other application integration architectural styles that we have explored in the past. The applications we have integrated in IBM are primarily in the domains of Application Lifecycle Management (ALM) and Integration System Management (ISM), but we believe that our experiences using read/write Linked Data to solve application integration problems could be broadly relevant and applicable within the IT industry.

This paper explains why Linked Data, which builds on the existing World Wide Web infrastructure, presents some unique characteristics, such as being distributed and scalable, that may allow the industry to succeed where other application integration approaches have failed. It discusses lessons we have learned along the way and some of the challenges we have been facing in using Linked Data to integrate enterprise applications.

Finally, we discuss several areas that could benefit from additional standard work and discuss several commonly applicable usage patterns along with proposals on how to address them using the existing W3C standards in the form of a Linked Data Basic Profile. This includes techniques applicable to clients and servers that read and write linked data, a type of container that allows new resources to be created using HTTP POST and existing resources to be found using HTTP GET (analogous to things like Atom Publishing Protocol (APP) [3]).

General Terms

Management, Design, Standardization

Keywords

Linked Data, Usage Patterns, Application Integration, Enterprise Application, Standards, ALM, ISM, Profile

LDOW2012, April 16, 2012, Lyon, France.

1. INTRODUCTION

There is interest in Linked Data technologies for more than one purpose. We have seen interest for the purpose of exposing information – for example public records – on the Internet in a machine-readable format. We have also seen interest in the use of Linked Data for inferring new information from existing information, for example in pharmaceutical applications or IBM Watson [4]. The IBM Rational team has been using Linked Data as an architectural model and implementation technology for application integration in the Product and Application Lifecycle Management domain. This approach has been largely successful and we are pleased – even passionate – about the results but getting there has not been easy. Although related work exists [5] [6][7][8][9], as far as we can tell, there is only a very limited number of people trying to use Linked Data technologies the way we are, and the little information that is available on best practices and pitfalls remains widely dispersed. We believe that Linked Data has the potential to solve some important problems that have frustrated the IT industry for many years, or at least make significant advances in that direction, but this potential will only be realized if we can establish and communicate a much richer body of knowledge on how to exploit these technologies. In some cases, there also are gaps in the Linked Data standards that need to be addressed. To help with this process, we discuss in this paper several best practices and anti-patterns we have identified as applicable to more domains than ALM. These include accessing, updating and creating resources from servers that expose their resources as Linked Data.

2. THE INTEGRATION CHALLENGE

IBM Rational is a vendor of industry leading system and software development tools, particularly those that support the general software development process such as bug tracking, requirements management and test management tools. Like many vendors who sell multiple applications, we have seen strong customer demand for better support of more complete business processes - in our case system and software development processes - that span the roles, tasks and data addressed by multiple tools. While answering this demand within the realm of a single vendor offering made of many different products can be challenging it quickly becomes unmanageable when customers want to mix in products from other vendors as well as their own homegrown components.

We describe our problem domain here to explain that we were led to explore these technologies by our need to solve long-standing problems in commercial application development and to emphasize that our conclusions are supported by experience in

shipping and deploying real applications, but we do not believe that our experiences or these technologies are limited to our application domain. These problems are encountered in many application domains, have existed for many years, and our industry has tried several different architectural approaches to address the problem of integrating the various products these complex scenarios require. Here are a few:

1. Implement some sort of Application Programming Interface (API) for each application, and then, in each application, implement "glue code" that exploits the APIs of other applications to link them together.
2. Design a single database to store the data of multiple applications, and implement each of the applications against this database. In the software development tools business, these databases are often called "repositories".
3. Implement a central "hub" or "bus" that orchestrates the broader business process by exploiting the APIs described in option 1 above.

While a discussion of the failings of each of these approaches is outside the scope of this document it is fair to say that although each one of them has its adherents and can point to some successes, none of them is wholly satisfactory. So, we decided to look for an alternative.

3. WHAT WOULD SUCCESS LOOK LIKE?

Unsatisfied with the state of the art regarding product integration in the ALM domain we decided around 2004 to have another look at how we might approach this integration problem.

Stepping back from what had already been attempted to date we started by identifying what characteristics an ideal solution would have. We came up with the following list:

Distributed – because of outsourcing, acquisitions, and the Internet, systems and work forces are increasingly distributed.

Scalable - need to scale to an unlimited number of products and users

Reliable – as we move from local area networks to wide area networks, as we move to remote areas of the world without the best infrastructures, and as users increasingly use mobile technology, we have to be reliable across a wide range of connectivity profiles.

Extensible – we need to be extensible in the sense that we can work with a wide variety of resources both in the application delivery domain but also in adjacent domains.

Simple – avoid the fragility we saw with tight coupling and keep the barrier to entry low so that it will be easy for people to interoperate with our products.

Equitable – equitable architecture that is open to everyone with no barriers to participation.

4. THE SOLUTION

When looking for a solution that had these characteristics – distributed, scalable, reliable, extensible, simple, and equitable – we realized that one such solution already existed: The World-Wide Web.

The Internet is all over the world, it supports billions of users, it's never gone down, it supports every kind of capability from web pages to video, from education to business, and anyone with an internet connection and an input device can participate in it.

One reason the Web enjoys all these characteristics is that it works in terms of protocols and resource formats rather than application specific interfaces. As an example, the web allows anyone to access any web page using whatever device and browser they like, independently of the type of hardware and system the server is running on. This is possible because the web relies on a resource format for web pages – HTML – and a protocol for accessing these resources – HTTP –.

Applying the same principle to the ALM domain integration problem meant thinking in terms of domain specific resources, such as requirements, change requests, and defects, and access to these resources rather than in terms of tools. We stopped thinking of the applications as being the central concept of the architecture and instead started to focus on the resources.

In this architecture the focus is on a web of resources from the various application domains – in our case, change management or quality management etc. - the applications are viewed as simply handlers of HTTP requests for those resources, and are not a central focus. Because each resource is identified by a URI, we can easily express arbitrary linkage between resources from different domains or the same domajn.

When we started in this direction, we were not fully aware of the linked data work – we reasoned by analogy with the HTML web, and we had understood the value of HTTP and URLs for solving our problems. For data representations, we continued to look to XML for solutions. Over time it became clear to us that to realize the full potential of the architecture we needed a simpler and more prescriptive data model than the one offered by XML, and so we started transitioning to RDF [10]. At this point we realized that what we were really doing was applying Linked Data principles to application integration.

5. LINKED DATA

We wanted an architecture that is minimalist, loosely coupled, had a standard data representation, kept the barriers to entry low and could be supported by existing applications implemented with many implementation technologies. Linked Data was just what we needed.

Linked Data was defined by Tim Berners-Lee as the following four rules [1]:

1) Use URIs as names for things
2) Use HTTP URIs so that people can look up those names.
3) When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
4) Include links to other URIs, so that they can discover more things.

RDF provides a data model that is very flexible, enables interoperability and extensibility.

With RDF we were able to model the different types of resources we needed and the relationships between themsuch that for ALM a change request becomes a resource exposed as RDF that can be linked to the defect it is to address, and a test to use to validate the change to be made. With Linked Data the change management,

defect management, and test management tools no longer connect to each other via specific interfaces but simply access the resources directly, following the Linked Data principles.

6. CONVENTIONS

As we embarked on the process of defining the various resource types we needed, their relationship, and their lifecycle it became apparent that we also needed to define a set of conventions above what is currently defined by W3C and the Linked Data standards. Some of these are simple rules that could be thought of as clarification of the basic Linked Data principles. Others are necessary because, unlike many uses of Linked Data, which are essentially read-only, our use of Linked Data is fundamentally read-write which raises its own set of challenges.

The following lists some of the categories these conventions fall in:

- Resources – a set of HTTP and RDF standard techniques and best practices that you should use, and anti-patterns you should avoid, when constructing clients and servers that read and write linked data. This includes a set of common properties leveraging existing RDF vocabularies such as Dublin Core [11]. It also includes what HTTP verb to use for creating, updating, getting, and deleting a resource as well as how to use them. In particular, in a system where tools may expand resources with additional properties beyond the core properties required to be supported by everyone it is crucial that any application that updates a resource preserves the properties it doesn't understand.
- Containers – a type of resource that allows new resources to be created using HTTP POST and existing resources to be found using HTTP GET. These containers are to RDF what APP is to XML. They answer the following two basic questions:
    1) To which URLs can I POST to create new resources?
    2) Where can I GET a list of existing resources?
- Paging – a mechanism for splitting the information in large containers into pages that can be fetched incrementally. For example, an individual defect usually is sufficiently small that it makes sense to send it all at once, but the list of all the defects ever created is typically too big. The paging mechanism provides a way to communicate the list in chunks with a simple set of conventions on how to query the first page and how pages are linked from one to the next.
- Ordering – a mechanism for specifying which predicates were used for page ordering..

The following sections provide further details regarding a proposal for addressing these in the form of a "Basic Profile for Linked Data" inspired by our work on Open Services for Lifecycle Collaboration (OSLC) [12].

7. TERMINOLOGY

The terminology used in this paper is based on W3C's Architecture of the World Wide Web [13] and Hyper-text Transfer Protocol (HTTP/1.1) [14].

Link : A relationship between two resources when one resource (representation) refers to the other resource by means of a URI.

Basic Profile : A specification that defines the needed specification components from other specifications as well as providing clarifications and patterns. Within the "Basic Profile for Linked Data", it is sometimes referred to as a shortened "Basic Profile".

Client : A program that establishes connections for the purpose of sending requests.

Basic Profile Client : A client that adheres to the rules defined in the Basic Profile.

Server: An application program that accepts connections in order to service requests by sending back responses. Any given program may be capable of being both a client and a server; our use of these terms refers only to the role being performed by the program for a particular connection, rather than to the program's capabilities in general. Likewise, any server may act as an origin server, proxy, gateway, or tunnel, switching behavior based on the nature of each request.

Basic Profile Server : A server that adheres to the rules defined in the Basic Profile.

8. BASIC PROFILE RESOURCES

Basic Profile Resources are HTTP linked data resources that conform to some simple patterns and conventions. Most Basic Profile Resources are domain-specific resources that contain data for an entity in some domain, which could be commercial, governmental, scientific, religious or other. A few Basic Profile Resources are defined by the Basic Profile specifications and are cross-domain. All Basic Profile Resources follow the four basic rules of Linked Data, previously laid out in section 5, to which Basic Profile adds a few rules of its own. Some of these rules could be thought of as clarification of the basic linked data rules.

1. **Basic Profile Resources are HTTP resources that can be created, modified, deleted and read using standard HTTP methods**.
   (Clarification or extension of Linked Data rule #2.)
   Basic Profile Resources are created by HTTP POST (or PUT) to an existing resource, deleted by HTTP DELETE, updated by HTTP PUT or PATCH [15], and "fetched" using HTTP GET.
   Additionally Basic Profile Resources can be created, updated and deleted using SPARQL Update [16].

2. **Basic Profile Resources use RDF to define their state.**
   (Clarification of Linked Data rule #3.) The state (in the sense of state used in the REST architecture) of a Basic Profile Resource is defined by a set of RDF triples.
   Basic Profile Resources can be mixed in the same application with other resources that do not have useful RDF representations such as binary and text resources.

3. **You can request an RDF/XML representation of any Basic Profile Resource.**
   (Clarification of Linked Data rule #3.) The resource may have other representations as well. These could be

other RDF formats, like Turtle, N3 or NTriples, but non-RDF formats like HTML and JSON would also be popular additions, and Basic Profile sets no limits.

4. **Basic Profile clients use Optimistic Collision Detection on Update.**
(Clarification of Linked Data rule #2.) Because the update process involves first getting a resource, modifying it and then later putting it back to the server there is the possibility of a conflict, e.g. some other client may have updated the resource since the GET. To mitigate this problem, Basic Profile implementations **should** use the HTTP `If-Match` header and HTTP ETags to detect collisions.

5. **Basic Profile Resources use standard vocabularies.**
Basic Profile Resources use common vocabularies (classes, properties, etc) for common concepts. Many web sites define their own vocabularies for common concepts like resource types, label, description, creator, last-modification-time, priority, enumeration of priority values and so on. This is usually viewed as a good feature by users who want their data to match their local terminology and processes, but it makes it much harder for organizations to subsequently integrate information in a larger view. Basic Profile requires all resources to expose common concepts using a common vocabulary for properties. Sites may choose to additionally expose the same values under their own private property names in the same resources. In general, Basic Profile avoids inventing its own property names where possible – it uses ones from popular RDF-based standards like the RDF standards themselves, Dublin Core, and so on. Basic Profile invents property URLs where no match is found in popular standard vocabularies. A number of recommended standard properties for use in Basic Profile Resources are listed below, in section 8.1.

6. **Basic Profile Resources set rdf:type explicitly.**
A resource's membership in a class extent can be indicated explicitly – by a triple in the resource representation that uses the rdf:type predicate and the URL of the class - or derived implicitly. In RDF there is no requirement to place an rdf:type triple in each resource, but this is a good practice, since it makes query more useful in cases where inferencing is not supported. Remember also that a single resource can have multiple values for rdf:type. For example, the dpbedia entry for Barack Obama [17] has dozens of rdf:types. Basic Profile sets no limits to the number of types a resource can have.

7. **Basic Profile Resources use a restricted number of standard datatypes.** RDF does not by itself define datatypes to be used for property values, so Basic Profile lists a set of standard datatypes to be used in Basic Profile to increase interoperability. Here is the list:
   - Boolean: a boolean type as specified by XSD [18] Boolean.
   - Date: a Date type as specified by XSD date.
   - DateTime: a Date and Time type as specified by XSD dateTime.
   - Decimal: a decimal number type as specified by XSD Decimal.
   - Double: a double floating-point number type as specified by XSD Double.
   - Float: a floating-point number type as specified by XSD Float.
   - Integer: an integer number type as specified by XSD Integer.
   - String: a string type as specified by XSD String).
   - XMLLiteral: a Literal XML value.

8. **Basic Profile clients expect to encounter unknown properties and content.**
Basic Profile provides mechanisms for clients to discover lists of expected properties for resources for particular purposes, but also assumes that any given resource may have many more properties than are listed. Some servers will only support a fixed set of properties for a particular type of resource. Clients should always assume that the set of properties for a resource of a particular type at an arbitrary server may be open in the sense that different resources of the same type may not all have the same properties, and the set of properties that are used in the state of a resource are not limited to any pre-defined set. However, when dealing with Basic Profile Resources, clients should assume that a Basic Profile server may discard triples for properties of which it does have prior knowledge. In other words, servers may restrict themselves to a known set of properties, but clients may not. When doing an update using HTTP PUT, a Basic Profile client must preserve all property-values retrieved using GET that it doesn't change whether it understands them or not. (Use of HTTP PATCH or SPARQL Update instead of PUT for update avoids this burden for clients.)

9. **Basic Profile clients do not assume the type of a resource at the end of a link.**
Many specifications and most traditional applications have a "closed model", by which we mean that any reference from a resource in the specification or application necessarily identifies a resource in the same specification (or a referenced specification) or application. By contrast, the HTML anchor tag can point to any resource addressable by an HTTP URI, not just other HTML resources. Basic Profile works like HTML in this sense. A HTTP URI reference in one Basic Profile resource may in general point to any resource, not just a Basic Profile resource.

   There are numerous reasons to maintain an open model like HTML's. One is that it allows data that has not yet been defined to be incorporated in the web in the future. Another reason is that it allows individual applications and sites to evolve over time - if clients assume that they know what will be at the other end of a link, then the data formats of all resources across the transitive closure of all links has to be kept stable for version upgrade.

A consequence of this independence is that client implementations that traverse HTTP URI links from one resource to another should always code defensively and be prepared for any resource at the end of the link. Defensive coding by clients is necessary to allow sets of applications that communicate via Basic Profile to be independently upgraded and flexibly extended.

8.1 Common Properties

The following are some properties from well-known RDF vocabularies that are recommended for use in Basic Profile Resources. Basic Profile requires none of them, but a specification based on Basic Profile may require one of these properties or more for a particular resource type.

Commonly used namespace prefixes:

```
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix rdf:
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs:
  <http://www.w3.org/2000/01/rdf-schema#>.
@prefix bp:
  <http://open-services.net/ns/basicProfile#>.
@prefix xsd:
  <http://www.w3.org/2001/XMLSchema#>.
```

8.1.1 From Dublin Core

URI: http://purl.org/dc/terms/

Property	Range	Comment
dcterms:contributor	dcterms:Agent	The identifier of a resource (or blank node) that is a contributor of information. This resource may be a person or group of people, or possibly an automated system.
dcterms:creator	dcterms:Agent	The identifier of a resource (or blank node) that is the original creator of the resource. This resource may be a person or group of people, or possibly an automated system.
dcterms:created	xsd:dateTime	The creation timestamp
dcterms:description	rdf:XMLLiteral	Descriptive text about the resource represented as rich text in XHTML format. **SHOULD** include only content that is valid and suitable inside an XHTML element.
dcterms:identifier	rdfs:Literal	A unique identifier for the resource. Typically read-only and assigned by the service provider when a resource is created. Not typically intended for end-user display.
dcterms:modified	xsd:dateTime	Date on which the resource was changed.
dcterms:relation	rdfs:Resource	The URI of a related resource. This is the predicate to use when you don't know what else to use. If you know more specifically what sort of relationship it is, use a more specific predicate.
dcterms:subject	rdfs:Resource	Should be a URI (see dbpedia.org) "Typically, the subject will be represented using keywords, key phrases, or classification codes. Recommended best practice is to use a controlled vocabulary. To describe the spatial or temporal topic of the resource, use the Coverage element." (from Dublin Core)
dcterms:title	rdf:XMLLiteral	A name given to the resource. Represented as rich text in XHTML format. **SHOULD** include only content that is valid inside an XHTML element.

8.1.2 From RDF

URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#

Property	Range	Comment
rdf:type	rdfs:Class	The type or types of the resource. Basic Profile recommends that the rdf:type(s) of a resource be set explicitly in resource representations to facilitate query with non-inferencing query engines

8.1.3 From RDF Schema

URI: http://www.w3.org/2000/01/rdf-schema#

Property	Range	Comment
rdfs:member	rdf:Resource	The URI (or blank node identifier) of a member of a container.
rdfs:label	rdf:Resource	"Provides a human-readable version of a resource name." (From RDFS)

9. BASIC PROFILE CONTAINER

Many HTTP applications and sites have organizing concepts that partition the overall space of resources into smaller containers. Blog posts are grouped into blogs, wiki pages are grouped into wikis, and products are grouped into catalogs. Each resource created in the application or site is created within an instance of one of these container-like entities, and users can list the existing artifacts within one. There is no agreement across applications or sites, even within a particular domain, on what these grouping concepts should be called, but they commonly exist and are important. Containers answer two basic questions, which are:

1. To which URLs can I POST to create new resources?
2. Where can I GET a list of existing resources?

In the XML world, APP has become popular as a standard for answering these questions. APP is not a good match for Linked Data - this specification shows how the same problems that are solved by APP for XML-centric designs can be solved by a

simple Linked Data usage pattern with some simple conventions on posting to RDF containers. We call these RDF containers that you can POST to Basic Profile Containers. Here are some of their characteristics:

1. A Basic Profile Container is a resource that is a Basic Profile Resource of type bp:Container.
2. Clients can retrieve the list of existing resources in a Basic Profile Container.
3. New resources are created in a Basic Profile Container by POSTing to it.
4. Any resource can be POSTed to a Basic Profile Container - a resource does not have to be a Basic Profile Resource with an RDF representation to be POSTed to a Basic Profile Container.
5. After POSTing a new resource to a container, the new resource will appear as a member of the container until it is deleted. A container may also contain resources that were added through other means - for example through the user interface of the site that implements the Container.
6. The same resource may appear in multiple containers. This happens commonly if one container is a "view" onto a larger container.
7. Clients can get partial information about a Basic Profile Container without retrieving a full representation including all of its contents.

The representation of a Basic Profile Container is a standard RDF container representation using the rdfs:member predicate or another predicate specified by bp:membershipPredicate. For example, if you have a container with the URL http://example.org/container1, it might have the following representation:

```
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix rdfs:
   <http://www.w3.org/2000/01/rdf-schema#>.
@prefix bp:
   <http://open-services.net/ns/basicProfile#>.
<http://example.org/container1>
  a bp:Container ;
  dcterms:title "A very simple container";
  rdfs:member
     <http://example.org/container1/member1>,
     <http://example.org/container1/member2>,
     <http://example.org/container1/member3>.
```

Basic Profile does not recognize or recommend the use of other forms of RDF container such as Bag and Seq because they are not friendly to query. This follows standard linked data guidance for RDF usage (see *RDF Features Best Avoided in the Linked Data Context* [5]).

Sometimes it is useful to use a subject other than the container itself as the membership subject and to use a predicate other than rdfs:member as the membership predicate, as illustrated below.

```
# The following is the representation of
#   http://example.org/netW/nw1/assetCont
@prefix rdfs:
   <http://www.w3.org/2000/01/rdf-schema#>.
@prefix bp:
   <http://open-services.net/ns/basicProfile#>.
@prefix o: <http://example.org/ontology/>.
<http://example.org/netW/nw1/assetCont>
  a bp:Container;
  bp:membershipSubject
   <http://example.org/netW/nw1>;
  bp:membershipPredicate o:asset.


<http://example.org/netW/nw1>
  a o:netW;
  o:asset
   <http://example.org/netW/nw1/assetCont/a1>,
   <http://example.org/netW/nw1/assetCont/a2>.
```

The essential structure of the container is the same, but in this example, the membership subject is not the container itself – it is a separate net worth resource. The membership predicate is o:asset – a predicate from the domain model. A POST to this container will create a new asset and add it to the list of members by adding a new membership triple to the container. You might wonder why we didn't just make http://example.org/netW/nw1 a container and POST the new asset directly there. That would be a fine design if http://example.org/netW/nw1 had only assets, but if it has separate predicates for assets and liabilities, that design will not work because it is unspecified to which predicate the POST should add a membership triple. Having separate http://example.org/netW/nw1/assetCont and http://example.org/netW/nw1/liabilityCont container resources allows both assets and liabilities to be created.

In this example, clients cannot simply guess which resource is the membership subject and which predicate is the membership predicate, so the example includes this information in triples whose subject is the Basic Profile Container resource itself.

9.1 rdfs:Container Properties

Because a Basic Profile Container is a Basic Profile Resource the same set of common properties described in section 8.1 applies. In addition, Basic Profile Containers have the following specific properties:

Property	Occurs	Range	Comment
bp:membershipPredicate	zero or one	rdfs:Property	Indicates which predicate of the container should be used to determine the membership when it is not rdfs:member.
bp:membershipSubject	zero or one	rdfs:Property	Indicates which resource is the subject for the members of the container when it is not the container itself.

9.2 Retrieving non-member properties

The representation of a container that has many members may be large. When we looked at our use cases, we saw that there were several important cases where clients needed to access only the non-member properties of the Container. [The dcterms properties listed in this page may not seem important enough to warrant addressing this problem, but we have use cases that add other

predicates to containers - for providing validation information and associating SPARQL endpoints for example.] Since retrieving the whole container representation to get this information may be onerous, we were motivated to define a way to retrieve only the non-member property values. We do this by defining for each Basic Profile Container a corresponding resource, called the "non-member resource", whose state is a subset of the state of the container. The non-member resource's HTTP URI can be derived in the following way.

If the HTTP URI of the container is {url}, then the HTTP URI of the related non-member resource is {url}?non-member-properties. The representation of {url}?non-member-properties is identical to the representation of {url}, except that the membership triples are missing. The subjects of the triples will still be {url} (or whatever they were in the representation of {url}), not {url}?non-member-properties. Any server that does not support non-member-resources should return an HTTP 404-NotFound error when a non-member-resource is requested.

This approach can be thought of as being analogous to using HTTP HEAD compared to HTTP GET. HTTP HEAD is used to fetch the response headers for a resource as opposed to requesting the entire representation of a resource using HTTP GET.

Here is an example:

Request:

```
GET /container1?non-member-properties
HOST: example.org
Accept: text/turtle
```

Response:

```
@prefix rdfs:
   <http://www.w3.org/2000/01/rdf-schema#>.
@prefix dcterms: <<http://purl.org/dc/terms/>.
<http://example.org/container1>
  a bp:Container;
  dcterms:title
    "A Basic Profile Container of Acme Resources";
  bp:membershipPredicate rdfs:member;
  dcterms:publisher <http://acme.com/>.
```

9.3 Design motivation and background

The concept of non-member-resources has not been especially controversial, but using the URL pattern {url}?non-member-properties to identify them has been controversial. Some people feel it's an unacceptable intrusion into the URL space that is owned and controlled by the server that defines {url}. A more practical objection is that servers respond unpredictably to URLs they do not understand, especially those that have a "?" character in them. For example, some servers will return the resource identified by the portion of the URL that precedes the "?" and simply ignore the rest. This problem could perhaps be mitigated by using a character other than "?" in the URL pattern. An alternative design that was discussed uses a header field in the response header of {url} to allow the server to control and communicate the URL of the corresponding non-member-resource - presence or absence of the header field would let clients know whether the non-member-resource is supported by the server. The advantages of this approach are that it does not impinge on the server's URL space, and it works predictably for servers that do not understand the concept of a non-member-resource. The disadvantages are that it requires two server round-trips - a HEAD and a GET - to retrieve the non-member-resources, and it requires the definition of a custom HTTP header, which to some people at least seems comparatively heavyweight.

9.4 Paging

Basic Profile Containers may support a technique called Paging which allows the representation of large containers to be transmitted in chunks.

Paging can be achieved with a simple RDF pattern. For each container resource, <containerURL>, we define a new resource <containerURL>?firstPage. The triples in the representation of <containerURL>?firstPage are a subset of the triples in <containerURL> - same subject, predicate and object.

Basic Profile Container servers may respond to requests for a container by redirecting the client to the first page resource – using a HTTP-303 "See Other" HTTP redirect to the actual URL for the page resource.

Continuing on from the member information from the JohnZSmith net worth example, we'll split the response across two pages. The client requests the first page as http://example.org/netW/nw1/assetCont?firstPage:

```
# The following is the representation of
# http://example.org/netW/nw1/assetCont?firstPage
@prefix rdf:
   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix bp:
  <http://open-services.net/ns/basicProfile#>.
@prefix o: <http://example.org/ontology/>.

<http://example.org/netW/nw1/assetCont>
  a bp:Container;
  dcterms:title "The assets of JohnZSmith";
  bp:membershipSubject
    <http://example.org/netW/nw1>;
  bp:membershipPredicate o:asset.

<http://example.org/netW/nw1/assetCont?firstPage>
  a bp:Page;
  bp:pageOf
    <http://example.org/netW/nw1/assetCont>;
  bp:nextPage
    <http://example.org/netW/nw1/assetCont?p=2>.

<http://example.org/netW/nw1>
  a o:netW;
  o:asset
    <http://example.org/netW/nw1/assetCont/a1>,
    <http://example.org/netW/nw1/assetCont/a4>,
    <http://example.org/netW/nw1/assetCont/a3>,
    <http://example.org/netW/nw1/assetCont/a2>.

<http://example.org/netW/nw1/assetCont/a1>
  a o:Stock;
```

```
  o:value 100.00.
<http://example.org/netW/nw1/assetCont/a2>
  a o:Cash;
  o:value 50.00.
# server initially supplied no data for a3 and a4
in this response
```

The following example is the result of retrieving the representation for the next page:

```
# The following is the representation of
# http://example.org/netW/nw1/assetCont?p=2
@prefix rdf:
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix bp:
  <http://open-services.net/ns/basicProfile#>.
@prefix o: <http://example.org/ontology/>.

<http://example.org/netW/nw1/assetCont>
  a bp:Container;
  dcterms:title "The assets of JohnZSmith";
  bp:membershipSubject
    <http://example.org/netW/nw1>;
  bp:membershipPredicate o:asset.

<http://example.org/netW/nw1/assetCont?p=2>
  a bp:Page;
  bp:pageOf
    <http://example.org/netW/nw1/assetCont>;
  bp:nextPage rdf:nil.

<http://example.org/netW/nw1>
  a o:netW;
  o:asset
   <http://example.org/netW/nw1/assetCont/a5>.

<http://example.org/netW/nw1/assetCont/a5>
  a o:Stock;
  dcterms:title "Big Co.";
  o:value 200.02.
```

In this example, there is only one member in the container in the final page. To indicate this is the last page, a value of rdf:nil is used for the bp:nextPage predicate of the page resource.

Basic Profile Container guarantees that any and all the triples about the members will be on the same page as the membership triple for the member.

9.5 Ordering

There are many cases where an ordering of the members of the container is important. Basic Profile Container does not provide any particular support for server ordering of members in containers, because any client can order the members in any way it chooses based on the value of any available property of the members. In the example below, the value of the o:value predicate is present for each member, so the client can easily order the members according to the value of that property. In this way, Basic Profile Container avoids the use of RDF constructs like Seq and List for expressing order.

Order only becomes important for Basic Profile Container servers when containers are paginated. If the server does not respect ordering when constructing pages, the client is forced to retrieve all pages before sorting the members, which would defeat the purpose of pagination. In cases where ordering is important, a Basic Profile Container server exposes all the members on a page with a higher sort order than all members on the previous page and lower sort order than all the members on the next page. The Basic Profile Container specification provides a predicate - bp:containerSortPredicates - that the server may use to communicate to the client which predicates were used for page ordering. Multiple predicate values may have been used for sorting, so the value of this predicate is an ordered list.

Here is an example container described previously, with representation for ordering of the assets:

```
# The following is the ordered representation of
# http://example.org/netW/nw1/assetCont
@prefix rdf:
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix bp:
  <http://open-services.net/ns/basicProfile#>.
@prefix o: <http://example.org/ontology/>.

<http://example.org/netW/nw1/assetCont>
  a bp:Container;
  dcterms:title "The assets of JohnZSmith";
  bp:membershipSubject
    <http://example.org/netW/nw1>;
  bp:membershipPredicate o:asset.

<http://example.org/netW/nw1/assetCont?firstPage>
   a bp:Page;
   bp:pageOf
     <http://example.org/netW/nw1/assetCont>;
   bp:containerSortPredicates (o:value).

<http://example.org/netW/nw1>
   a o:netW;
   o:asset
      <http://example.org/netW/nw1/assetCont/a1>,
      <http://example.org/netW/nw1/assetCont/a3>,
      <http://example.org/netW/nw1/assetCont/a2>.

<http://example.org/netW/nw1/assetCont/a1>
   a o:Stock;
   o:value 100.00.
<http://example.org/netW/nw1/assetCont/a2>
```

```
   a o:Cash;
   o:value 50.00.
<http://example.org/netW/nw1/assetCont/a3>
   a o:RealEstateHolding;
   o:value 300000.
```

As you can see by the addition of the bp:containerSortPredicates predicate, the o:value predicate is used to define the ordering of the results. It is up to the domain model and server to determine the appropriate predicate to indicate the resource's order within a page, and up to the client receiving this representation to use that order in whatever way is appropriate, for example to sort the data prior to presentation on a user interface.

10. CONCLUSION

We have shipped a number of products using the Linked Data technology as a way to integrate ALM products and are generally pleased with the result. We now have more products in development that use these technologies and are seeing a strong interest in this approach in other parts of our company.

As more data gets exposed using Linked Data we believe we will be able to do even more for our customers, with a set of integration services with richer capabilities such as traceability across relationships, impact analysis and deep querying capabilities. Additionally, we will be able to develop higher level analytics, reports, and dashboards providing data from multiple products across different domains. We will be able to answer questions such as: what enhancements in today's build address requirements that need to be tested with certain test cases?

We believe that Linked Data has the potential to solve some important problems that have frustrated the IT industry for many years, or at least make significant advances in that direction, but this potential will only be realized if we can establish and communicate a much richer body of knowledge on how to exploit these technologies.

It has taken us a number of years of experimentation to achieve the level of understanding that we have today, we have made some costly mistakes along the way, and we see no immediate end to the challenges and learning that lie before us. As far as we can tell, there is only a very limited number of people trying to use Linked Data technologies in the ways we are using them, and the little information that is available on best practices and pitfalls is widely dispersed. In some cases, there also are gaps in the Linked Data standards that need to be addressed.

We believe that defining a simple basic profile will enable broader adoption of Linked Data principles for application integration. Additional development of some of the concepts will be needed to complete such a basic profile. We are encouraged by the work started at the W3C Linked Enterprise Data Pattenrs workshop [19] and look forward to participating in subsequent activities. [20]

By sharing information on how we use these technologies we hope to help the industry move forward on these issues.

11. ACKNOWLEDGMENTS

This paper contains material provided by Bill Higgins from IBM, and several of the concepts discussed here come from our work in the OSLC.

Thanks to Arthur Ryman and John Arwe (as well as others) for review, feedback, and some content.

12. REFERENCES

[1] Tim Berners-Lee. *Linked Data Design Issues,* 2006 http://www.w3.org/DesignIssues/LinkedData.html

[2] *Linked Data – Connect Distributed Data across the Web* http://linkeddata.org/

[3] J Gregorio, B. de hOra. *Atom Publishing Protocol (APP)*, IETF RFC5023, 2007 http://www.ietf.org/rfc/rfc5023.txt

[4] *IBM Watson* http://www.ibm.com/innovation/us/watson

[5] Tom Heath, Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space,* 2011. http://linkeddatabook.com/editions/1.0/

[6] Leigh Dodds. Ian Davis. *Linked Data Patterns*, 2011. http://patterns.dataincubator.org

[7] Tetlow, Phil, Jeff Z Pan, Daniel Oberle, Evan Wallace, Michael Uschold, and Elisa Kendall. *Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering,* W3C, 2006. http://www.w3.org/2001/sw/BestPractices/SE/ODA/.

[8] De Cesare, Sergio, Guido L Geerts, Grant Holland, Mark Lycett, and Chris Partridge. *Ontology-driven software engineering.* Ed. Regina Bernhaupt, Peter Forbrig, Jan Gulliksen, and Marta Lrusdttir. 2010. October 6409: 279-280. http://portal.acm.org/citation.cfm?doid=1639950.1639983.

[9] Hesse, Wolfgang. *Engineers Discovering the Real World From Model-Driven to Ontology-Based Software Engineering.* 2008. In Information Systems and eBusiness Technologies, ed. Will Aalst, John Mylopoulos, Norman M Sadeh, Michael J Shaw, Clemens Szyperski, Roland Kaschek, Christian Kop, Claudia Steinberger, and Gnther Fliedl, 5:136-147. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-540-78942-0_16.

[10] Graham Klyne, Jeremy J. Carroll. *Resource Description Framework (RDF)*, W3C, 2004 http://www.w3.org/TR/rdf-concepts/

[11] *Dublin Core Metadata Initiative* http://dublincore.org

[12] *Open Services for Lifecycle Collaboration (OSLC)* http://open-services.net

[13] Ian Jacobs, Norman Walsh. *Architecture of the World Wide Web*, W3C. 2004. http://www.w3.org/TR/webarch/

[14] L Dusseault, J. Snell, *PATCH Method for HTTP*. IETF RFC5789, 2010 http://tools.ietf.org/html/rfc5789

[15] Paul Gearon, Alexandre Passant, Axel Polleres. *SPARQL 1.1 Update*, W3C 2012 http://www.w3.org/TR/sparql11-update/

[16] R. Fielding and al. *Hyper-text Transfer Protocol (HTTP/1.1)*, IETF RFC2616, 1999. http://tools.ietf.org/html/rfc2616

[17] *Dbpedia entry for Barack Obama* http://dbpedia.org/page/Barack_Obama

[18] Paul Biron, Ashok Malhotra. *XML Schema Part 2: Datatypes,* Second Edition, W3C, 2004 http://www.w3.org/TR/xmlschema-2/

[19] *W3C Linked Enterprise Data Patterns Workshop* http://www.w3.org/2011/09/LinkedData/

[20] *Linked Data at W3C* http://www.w3.org/standards/semanticweb/data