Available online at www.sciencedirect.com

ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web 6 (2008) 61–69

JOURNAL OF Web Semantics

www.elsevier.com/locate/websem

Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST)

Robert Battle*, Edward Benson

BBN Technologies, 1300 North 17th Street, Suite 400 Arlington, VA 22209, United States

Received 2 June 2007; received in revised form 7 September 2007; accepted 6 November 2007
Available online 17 November 2007

Abstract

Semantic Web technologies must integrate with Web 2.0 services for both to leverage each others strengths. We argue that the REST-based design methodologies [R.T. Fielding, R.N. Taylor, Principled design of the modern web architecture, ACM Trans. Internet Technol. (TOIT) 2 (2) (2002) 115–150] of the web present the ideal mechanism through which to align the publication of semantic data with the existing web architecture. We present the design and implementation of two solutions that combine REST-based design and RDF [D. Beckett (Ed.), RDF/XML Syntax Specification (Revised), W3C Recommendation, February 10, 2004] data access: one solution for integrating existing web services and one server-side solution for creating RDF REST services. Both of these solutions enable SPARQL [E. Prud'hommeaux, A. Seaborne (Eds.), SPARQL Query Language for RDF, W3C Working Draft, March 26, 2007] to be a unifying data access layer for aligning the Semantic Web and Web 2.0.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Semantic Web; Web 2.0; Representational State Transfer; Web Services

1. Introduction

The Web 2.0 and Semantic Web communities maintain different strategies towards a similar goal—an interlinked web of data that exposes information for easy processing, integration, and reuse. The usability-centric developments that drive Web 2.0 have resulted in an entire new economy of web sites and mash-ups over the past few years, but the absence of semantic data descriptions on Web 2.0 has put a low ceiling on the complexity that these sites can achieve. Each new mash-up requires the hand-tuning of every service involved.

The Semantic Web has solved different problems. Formats such as OWL [16] and RDF [3] provide ways to semantically describe and align data from disparate sources, but the lack of usable and agreed upon data access methods have prevented widespread use of such data. Both disciplines need each other for either to realize their ultimate goals. Web 2.0 has a wealth of data but poor semantics and difficult integration. The Semantic Web solves the integration issue but suffers from a dearth of users.

An ideal solution would bridge the two disciplines so that their access points aligned. Existing web sites could inter-operate with both OWL and RDF data while also serving as access points for semantically enabled clients or query architectures. This paper demonstrates that such an alignment is not only possible, but it can be accomplished seamlessly using the Representational State Transfer (REST) design methodology common amongst Web 2.0 developers today.

We first examine the REST methodology [2] and argue that REST-based design is both a fundamental enabling technology of Web 2.0 and a natural fit for Semantic Web operations. We then introduce our design and implementation of two complimentary strategies for REST-based alignment of the semantic world with Web 2.0. The first, the Semantic Bridge for Web Services, uses semantic tags on traditional SOAP and REST endpoints to allow their incorporation in semantic queries. The second, Semantic REST, defines a standard way through which REST-based resources on Web 2.0 can be exposed, modified, and queried in RDF alongside their existing HTML, XML, and JSON endpoints. Finally, we show an example integration of these techniques by performing a distributed query across data sources using both the Semantic Bridge for Web Services and Semantic REST.

* Corresponding author. Fax: +1 703 284 1281.
E-mail addresses: rbattle@bbn.com (R. Battle), ebenson@bbn.com (E. Benson).

1570-8268/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.websem.2007.11.002

2. REST-based Web design

Representational State Transfer is a pattern of resource operations that has emerged as a *de facto* standard for service design in Web 2.0 applications. Whereas the traditional SOAP-based approach to Web Services uses full-blown remote objects with remote method invocation and encapsulated functionality, REST deals only with data structures and the transfer of their state. REST's simplicity, along with its natural fit over HTTP, has contributed to its status as a method of choice for Web 2.0 applications to expose their data.

Resources in a RESTful service are both identified by and resolved with a URL that generally takes the form:

```
ResourceURL ::= Protocol://Host/ApplicationPath/ResourceType/ResourceID
```

The combination of protocol, host, and application path serve as a namespace for all resources contained within a site, and the resource type and resource ID uniquely identify a particular resource within that namespace. Operations concerning a resource type but not a particular resource are accessed by the URL above without the ResourceID at the end. Thus a new user on the site *semwebcentral.org* would be created using the URI http://www.semwebcentral.org/user and a particular user would be accessed via http://www.semwebcentral.org/user/ogrouch, where 'ogrouch' is the user's ID.

At the core of REST based design is a set of state transfer operations universal to any data storage and retrieval system. These operations, as commonly interpreted on the web, are referred to by the acronym CRUD [11], for "Create, Read, Update, Delete." The Web 2.0 community has adopted an informal mapping of CRUD operations onto the commands provided by the HTTP protocol: POST, GET, PUT, and DELETE, respectively. These commands identify the particular CRUD operation being requested of the resource identified by the URL endpoint.

Table 1 presents a mapping of CRUD operations onto HTTP requests. It is important to note that REST is not a standard in the W3C sense of the word, but rather a design technique under active interpretation across the web. While the table below presents one commonly used application of REST to the web, it is not the only interpretation.

One wrinkle in this provided mapping is the refusal of some public routers to forward HTTP DELETE requests. Due to the unreliable forwarding of this particular command, the delete operation is usually specified as either a URL-encoded request argument or a suffix appended to the request URL, such as the following example [11]:

```
GET http://www.semwebcentral.org/user/ogrouch;delete
```

A great benefit of REST-based web design is the ability to use HTTP Headers to provide request context around each of the CRUD operations. A request to a particular resource might result in HTML, XML, or JSON depending on the desired media type transmitted in the HTTP Accept header. This allows developers to overlay the programmatic API for a website directly on top of the site exposed to web users and reduces the cost and complexity of providing multi-format access to a site's underlying data.

Drawing on the context provided by HTTP, a web site operating under REST principles can be viewed as a web-accessible API. The function signature of a call into this API is then described by the tuple: {Resource URL, HTTP Command, Accept Header}.

3. The case for REST-based alignment

REST-based design is critical to Web 2.0 because it provides a unified way of organizing and accessing data over many different mediums, enabling mashups and structuring the Ajax-based applications we have today. From a design standpoint, this approach consists of two basic principles:

- Each user-facing component of an application is modeled as a resource.
- Each resource in a web application is identified by and resolved through a URL.

These two principles have subtle but far reaching effects that have greatly encouraged and enabled interoperability between Web 2.0 sites. In a photo sharing application, for example, a photo is not *tagged* (verb) but rather a new *tag* (noun) resource is created. Because a tag is modeled as a resource instead of an action, it can be described in a number of declarative formats and externalized to other sites. These resource-based principles also lead to site APIs that are overlaid on top of the URL structure for web users, creating predictable and, to a certain extent, self-documenting APIs.

The REST design methodology also integrates well with the resource paradigm of the Semantic Web. The Semantic Web

Table 1
REST to HTTP Mapping

CRUD operation	HTTP command	Input format	Output format
Create	POST	HTTP Form Encoded	Status 201 CREATED
Read	GET	None	Determined by request headers
Update	PUT	HTTP Form Encoded	Status 200 OK
Delete	DELETE	None	Status 200 OK

uses URIs as resource identifiers, so the URL-based identifiers of REST fit naturally into its scheme. The Semantic Web, like REST, also deals strictly with assertions describing objects and their state; no parallel exists for SOAP-like remote method invocation. Finally, all common operations on the Semantic Web with the exception of query – data fetch, insertion, and deletion – are the fundamental operations in a REST-based system. It follows that REST-based web sites are an ideal carrier of semantic data and would even provide the additional benefit of resource resolvability in human-readable HTML. The remainder of this paper demonstrates strategies for making this a reality.

4. Semantic bridge for web services

A great deal of data on the web is already available through REST and SOAP-based access points but this data carries no markup to relate it to semantic concepts. Providing such markup would enable integration of these Web services with the Semantic Web for data access and semantic query across multiple web sites. In order for a semantic application to consume this data, it has to know what the data means; it is not useful knowing that you can search for books via Amazon.com's E-Commerce Web Service if the application consuming this data doesn't know that the service returns a book or even how a book is defined.

The Semantic Bridge for Web Services (SBWS) [21] is a Java tool (part of BBN's Asio [22] suite) we have developed to integrate existing web services into the Semantic Web. It wraps around a set of Web service operations described by a WSDL [6] or WADL [10] document to create a SPARQL [18] endpoint for those services. SBWS can then analyze SPARQL SELECT or CONSTRUCT queries and determine what combination of Web service operations will provide an answer.

4.1. Semantically describing web services

There are several frameworks that add semantic information to Web services, such as OWL-S [15] and SAWSDL [8]. They provide details for each Web service parameter that describes how the value is derived from some ontology. With this combination of Web services and Semantic markup, SBWS can provide a standard way of communicating across Web 2.0 applications regardless of how the applications themselves are defined.

4.1.1. SOAP and WSDL

SBWS can be configured to query SOAP Web Services [4] with the combination of a WSDL document describing the service mechanics and an OWL-S document describing the service semantics of the WSDL components. The OWL-S document maps each operation and message defined in the WSDL definition to an ontology. As most Web services return plain old XML, there needs to be a conversion from this XML data to RDF. The OWL-S document contains either an inline XSLT document or the URI of a document to use. This is important as the Semantic Bridge for Web Services can only use RDF data. A point of future work for SBWS is to integrate WSDL 2.0 [5] and SAWSDL to provide more flexibility for SBWS and WSDL based Web services. Using OWL-S with SOAP based web services provides access to traditional web services. SOAP, however, does not fit well with the REST principles of Web 2.0; it has one endpoint and many actions on that endpoint, whereas REST provides an endpoint for each resource that will be acted upon. To more closely align the Semantic Web with Web 2.0, SBWS needs to work with REST services.

4.1.2. REST and WADL

While SOAP based Web services have a WSDL document that defines their operations, there is no standard equivalent for REST services. Needing such a description language for SBWS to function, we chose the Web Application Description Language (WADL), from Sun Microsystems, as a specification for describing REST services. A WADL document is designed to be a simple alternative to WSDL for use with XML/HTTP Web applications. It provides a description of Web applications in a simpler format than WSDL while also defining how to generate the URI for each operation and defining the format of the input and output parameters. Since WADL was not designed for Semantic Web interoperability, it does not provide placeholders for semantic definitions of the operations, parameters, and results described within a WADL document. SBWS provides custom annotations to the WADL document [27], similar to SAWSDL, that describes the semantics of the REST service. There are three annotations for the WADL document that SBWS uses to add semantics to the input and output of a REST method: modelReference, valueProperty, and schemaMapping. The "modelReference" annotation tells SBWS the OWL class of the parameter. The "valueProperty" tells SBWS which data property in the model to use as a value for the parameter. The "schemaMapping" annotation gives SBWS an XSL transformation to apply to the model before setting it as the value of the parameter, which can be used instead of the "valueProperty" annotation. For an output parameter, the "schemaMapping" annotation tells SBWS how to convert the XML result into RDF. Eventually this should be replaced by a set of standard annotations, perhaps call SAWADL. With these annotations in hand, SBWS is able to breakdown SPARQL queries into a series of HTTP GET requests that satisfy a given SELECT or CONSTRUCT query. WADL and the SBWS annotations provide the integration that allows for the Semantic Web to utilize existing REST services.

4.2. Example usage with Amazon.com

Once SBWS has been configured for a particular web service, it enables that service to be used as if it were a SPARQL endpoint. As an example, consider Amazon.com's REST-based E-Commerce service [19] which allows developer to search its catalog for product information. This service takes item query fields as part of a search URL and returns an XML response describing a product. We would like to query this information semantically and have it returned in RDF format.

The first step required is to define an ontology on which to map the data provided by the Web Service. The example Amazon.com ontology used by this paper is available at [29]. Next, an annotated WADL document, available at [26] must be created

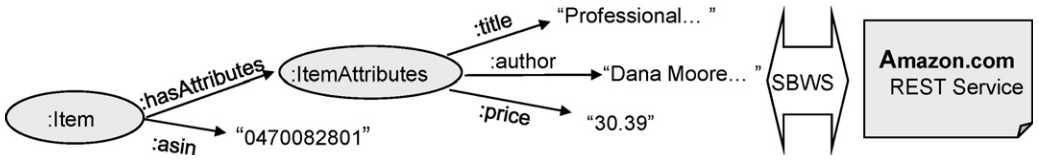

Fig. 1. Conceptual mapping from ontology to Amazon.com Web Service.

to align each element of the ontology with the request parameters and response structures of the Web Service. With these elements in place, SBWS can be used as a SPARQL proxy for Amazon.com's E-Commerce platform. Fig. 1 contains a representative depiction of our configuration.

This particular example of SBWS was configured to use Amazon.com's E-Commerce REST service with definitions for Amazon.com's ItemLookup and ItemSearch operations. The ItemLookup service accepts an Amazon Standard Identification Number (the ISBN of a book) in exchange for information, and the ItemSearch service accepts the author and title values as input.

Using the described configuration, we can submit the following query to SBWS to search for the title, price, and authors of the book with ISBN "0470082801":

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://semanticrest.org/example/amazon-ontology#>

SELECT ?title ?price ?author
WHERE {
 ?item a :Item;
  :asin "0470082801"^^xsd:integer;
  :hasItemAttributes ?attributes.
 ?attributes a :ItemAttributes;
  :price ?price;
  :author ?author;
  :title ?title.
}
```

Listing 1 - Example SBWS-powered SPARQL Query

SBWS will analyze all of the triples in the SPARQL query and see that the only triple that provides any data is the triple? item amazon:Item.ASIN "0470082801"^^xsd:integer. By looking at all of the semantically described operations in it's configuration, SBWS can see that the ItemLookup operation takes it's input from the amazon:Item.ASIN property and that it provides values for the? price,? author, and? title variables. Since invoking the ItemLookup operation solves the query, no further action is necessary. If SBWS encounters a query isn't satisfied by the output of a single web service call, SBWS can chain together web service calls into a hierarchy that can satisfy the query (assuming that the query can be satisfied by some combination of web service calls). A more detailed example using the live Amazon.com web service is detailed below.

4.3. Benefits of semantically enabled web services

There is a tremendous benefit that can be attained by integrating existing Web services with the Semantic Web. By adding semantics to existing Web services, many disparate sets of data on the Web can be integrated with little work. Later in the paper, we present an example using our distributed query architecture called the Semantic Query Decomposer (SQD) [14,20]. SQD provides a unified SPARQL endpoint over several individual data sources and automatically federates queries between them. With a component such as SQD, several instances of SBWS can be configured to work together. For example, one instance of SBWS can be configured to use Amazon.com's E-Commerce Service and another to use Facebook's Web Service API. With this system and a SPARQL SELECT or CONSTRUCT query, users can find out the price and ratings for each book in their friends profiles. This allows SBWS to answer questions from the entire set of configured Web services that could not be answered by an individual Web service. A popular theme in Web 2.0 is combining different Web services together to form a "mashup." Distributing a query across a set of semantically enabled Web services as described above allows for the services to be combined into a semantic "mashup."

Even within the context of a single Web service and its operations, SBWS provides a benefit for consumers of Web services. Its ability to analyze a query and chain together the output from one Web service invocation to the input of another Web service call gives it the ability to consider the entire set of Web service operations to answer questions that are not limited by the API presented by the developers of the Web service.

The data that is presented by conventional REST and SOAP web services is more accessible when it is semantically enabled and integrated into SBWS. By using SBWS with semantically described Web services in this way, it enables SPARQL to become the standard language for querying Web services and makes it easy to combine several Semantic Web services and query them as if they were a single semantic mashup. More traditional Web 2.0 mashup's rely heavily on programming directly to the Web service API. SBWS (in conjunction with SQD) abstracts this away by presenting a single SPARQL interface.

5. Semantic REST

Semantic REST is a different approach to Web 2.0 and Semantic Web integration that merges existing RDF operations with REST access points. While SBWS provides a way to adapt existing REST and SOAP-based Web services for semantic query, Semantic REST provides a new implementation for REST-based web sites to integrate fully into the Semantic Web. Semantic REST takes the existing set of standard REST operations and defines constraints and patterns for use with SPARQL and RDF. The result is an ability to query, retrieve, modify, delete, and add RDF data directly from the endpoints already in use in a REST-based web application.

Using this model, existing Semantic Web applications that operate over SPARQL endpoints may continue to perform normally, but new functionality is possible through the expanded set of operations the REST approach provides, and semantic resources double as web-resolvable pages. The Semantic REST approach is defined in two parts. First we describe the characteristics of Semantic REST requests, including a data constraint to enforce the REST mindset and an extended SPARQL syntax proposed by Hewlett Packard's Jena team [25]. Next we map HTTP REST operations onto the semantic world, describing what each combination of endpoint and HTTP command maps to in terms of SPARQL and RDF.

5.1. *Request characteristics*

5.1.1. *Graph constraints*

One of the fundamental principles behind REST as used in Web 2.0 is the idea of using an URL as a resolvable resource identifier. All operations about a particular resource can be handled through requests to its resolvable identifier. This also means that all operations to a resource's URL should be concerning that resource alone. Because RDF provides a descriptive flexibility that HTML Form-encoded data does not, constraints must be placed upon the data sent to a Semantic REST endpoint to prevent arbitrary statements and queries from being made. Doing so ensures that all RDF data sent as part of a request is relevant to the resource implied by the request endpoint.

To place this constraint, Semantic REST requires that all requests containing an RDF graph reference the endpoint URI in every disconnected sub-graph included in the request. If a request contains a graph that does not include the endpoint as an RDF node, then the request is rejected by the server and a response of HTTP 406 Not Acceptable is returned.

Additionally, Semantic REST operations that carry a graph payload that is to modify the graph on the server may not reference resources on that server that do not exist (thereby implicitly creating them). It is up to the server to verify that each individual resource within its resolvable namespace referenced in an INSERT or UPDATE request is a pre-existing resource in its local graph-store. Note that this rule means that the open world hypothesis does not apply to resources within the local namespace that a Semantic REST server controls; the server should be able to report whether they exist or not. Finally, to ensure that all operations are graph-related and can thus be subjected to these constraints, Semantic REST only supports the SELECT, CONSTRUCT, MODIFY, INSERT, and DELETE operators of SPARQL and SPARQL/Update (described below).

5.1.2. *Extended SPARQL syntax*

A standard for the insert, update, and delete operations necessary for any data system has not yet emerged for RDF, but it is reasonable to expect that SPARQL, as a proposed RDF query language, will be a medium for these operations when they are standardized. As such, Semantic Rest uses an extension of SPARQL called SPARQL/Update defined by the Jena team at Hewlett-Packard. This extension to SPARQL adds two additional commands, INSERT and DELETE, as well as a transactional operator MODIFY that allows deletion and insertion to be chained together to create the equivalent of an SQL update statement.

SPARQL/Update draws on the flexible syntax of SPARQL to allow variables in the data for INSERT and DELETE commands so that one command may be issued for many resources in the knowledge base. This leads to INSERT and DELETE queries immediately familiar to anyone who has used SPARQL, such as the example in Listing 2, which deletes all statements about users whose subscription expires before 1 January 2007. The endpoint of this example request is the class-level endpoint for a User, http://example.org/somewebsite/User.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX subscription: <http://example.org/somewebsite/subscription/>
PREFIX somewebsite: <http://example.org/somewebsite/>

DELETE {
 ?user ?p ?v
}
WHERE {
 ?user a somewebsite:User .
 ?user subscriptions:subscriptionStops ?finalDate .
   FILTER ( ?finalDate < "2007-01-01T00:00:00"^^xsd:dateTime )
}
```

Listing 2 - Example SPARQL/Update Query

5.1.3. *Specifying semantic REST*

As the Semantic REST operations are designed to be performed over the existing REST endpoints of a web site, HTTP Accept headers must be used to specify that a particular request is a Semantic REST one. In all of the operations detailed below, the response type will either be application/rdf+xml or application/sparql-results+xml. Clients should know by the nature of the request which type to expect in response, but may simply include both accept types if they wish to avoid having to alternate between two header templates based on this knowledge. Either of the two accept types in Listing 3 specify a Semantic REST operation.

Resource creation and statement insertion are kept completely separate to remove ambiguity from the creation request. The HTTP Post that creates a new resource carries no payload – it simply causes the server to generate a new resource handle and return it to the client. The resource handle uses the class-level endpoint as its namespace, so a POST to/users would result in a new resource with a URI of the form/users/resource_id being created. On the server side, resource creation causes a single triple to be inserted into the graph stating that thex new resource is of the class type defined by the endpoint used to create that resource.

```
Accept: application/rdf+xml
Accept: application/sparql-results+xml
```

Listing 3 - HTTP Accept Headers for Semantic REST Operation

5.2. *REST syntax*

The central concept behind Semantic REST is a mapping between the commonly accepted HTTP REST operations and actions appropriate for the Semantic Web. We focus this mapping primarily on SPARQL and its proposed extension as it represents an existing standard on which many Semantic Web applications are already based.

Semantic REST operations can occur at two types of endpoints: class-level and resource-level. Class-level endpoints represent an entire class of resources on the remote server, such as all users. These endpoints provide access to broad queries that affect or return a number of resources of that type, as well as resource creation. The URI of each resource in Semantic REST is also a resolvable endpoint. These resource-level endpoints provide information about the resource in RDF as well as the ability to add, remove, or modify information about that resource.

The operations in Table 2 describe the class-level endpoint functionality that may be used for operations concerning a particular resource type.

In this early version of Semantic REST, the handle created by the server is automatically generated, so it is like an auto-incrementing integer in the style of database row IDs. A method for specifying a custom handle (such as a 'ebenson') is a desirable addition for future versions of Semantic REST. The requirement on the server to generate this handle also indicates that some indexing mechanism must sit alongside the triple-store to keep track of existing handles and generate new unique ones.

As an example, an HTTP POST might be sent to the http://www.semwebcentral.org/user endpoint to create a new User resource on SemWebCentral. The server would create a unique resource handle (user:1234), insert a statement declaring its type (user:1234 a:User), and finally respond with the URI representing that new resource, http://www.semwebcentral.org/user/1234. The client could then interact with this new resource-level endpoint using the following resource-level Semantic REST conventions, defined in Table 3.

The Insert and Remove operations operate with the behavior and restrictions specified above in this paper, and the Read statement returns all triples for which the URI endpoint is either the subject or object.

Table 2
Class-level endpoints of semantic REST

Operation	HTTP command	Request data format	SPARQL command	Response
List	GET	None	n/a	SPARQL Select result of all resources of the type specified by the URL
Query	GET	SPARQL	SELECT or CONSTRUCT	Query Results
Create	POST	None	n/a	Status 201 CREATED
Insert	PUT	SPARQL/Update	INSERT	Status 200 OK
Remove	DELETE	SPARQL/Update	DELETE or MODIFY	Status 200 OK

Table 3
Resource-level endpoints of semantic REST

Operation	HTTP command	Request data format	SPARQL command	Response
Read	GET	Empty	n/a	RDF/XML
Insert	PUT	SPARQL/Update	INSERT	Status 200 OK
Remove	DELETE	SPARQL/Update	DELETE or MODIFY	Status 200 OK

Continuing the example of User 1234, a client could use the resource-level Semantic REST conventions to add information with the HTTP Put operation in Listing 4. The endpoint for this request would be the URI of the new resource just created, http://www.semwebcentral.org/user/1234.

```
PUT
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:swc="http://semwebcentral.org/example/semwebcentral-ontology#"
   xmlns:user="http://semwebcentral.org/user/"
   xml:base=" http://semwebcentral.org/user/">
<swc:Developer rdf:about="http://www.semwebcentral.org.org/user/1234">
  <user:first_name>Oscar</user:first_name>
  <user:last_name>Grouch</user:last_name>
</swc:Developer>
</rdf:RDF>
```

Listing 4 - Creating a resource with Semantic REST

The server validates that the one graph contained in this request references the resource defined by the endpoint serving the request, and then adds the new triples to its triple-store. It replies to this request with a Status 200 OK response indicating that the graph concerning resource http://www.semwebcentral.org/user/1234 had successfully been updated.

Several themes are apparent in this organization of operations, and while information security is outside the scope of this paper, this organization is chosen with future security enhancements in mind. The class-level endpoints provide the generalized administrative operations for the resource type: the ability to create new instances and the ability to perform sweeping queries, insertions, and deletions (modification is accomplished by chaining an insertion after a deletion). The resource-level endpoints offer capabilities with a narrower focus; they provide only the ability to read, insert, and delete statements about the particular resource implied by the URL.

Integrating the class-level and resource-level Semantic REST capability into an existing REST-based web site is a straightforward task, as these sites already contain code that routes incoming HTTP requests based on the desired response format (HTML, XML, etc). Once Semantic REST capability is added to a site, it becomes a full-featured member of the Semantic Web while maintaining its existing strengths as a web site. With the widespread use of this functionality, Web 2.0 sites such as the photo sharing site Flickr could benefit from better semantic understanding of its user data, and the Semantic Web community would benefit immensely from the tie-in to the vast wealth of information available on the web.

6. Example: distributed, semantic query over Web 2.0

To test the ideas presented in this paper, we performed a distributed semantic query across three separate data sources, each employing a different method of data exposure:

(1) A DAML DB-based [24] RDF triple-store containing mock employee data from a payroll system connected to traditional SPARQL endpoint.
(2) A Semantic REST compliant replica of SemWebCentral.org.
(3) Amazon.com's live REST-based service that allows users to search its database but not in RDF format.

We created a SBWS wrapper for Amazon.com's live REST-based service so that it could be treated as a semantic RDF data source. This wrapper contained the annotations necessary to map book and author data from Amazon's REST service to elements of the publication ontology at [25].

Performing a distributed semantic query is not a trivial task even when operating strictly within a Semantic Web environment. We employed the Semantic Query Decomposer (SQD) to assist us with this operation. SQD wraps around multiple semantic data sources with their own data source ontology defining the concepts in the data source and provides a unified SPARQL

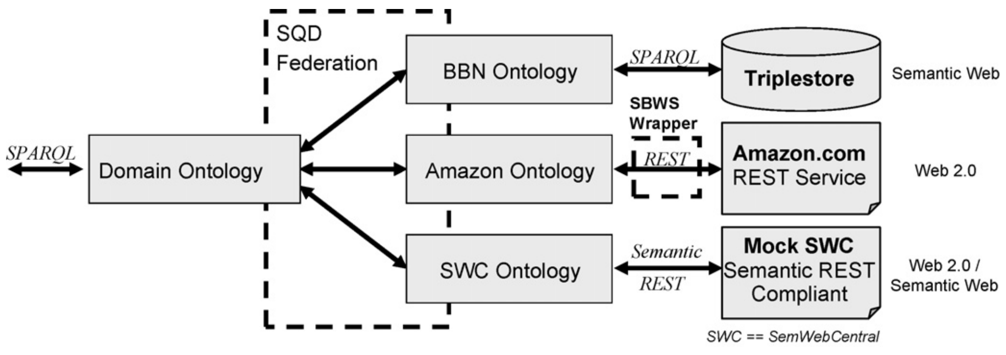

Fig. 2. System diagram of demonstration.

Table 4
Distributed, semantic query results using SBWS and Semantic REST

?Name	?Position	?Title
Dana Moore	Division Scientist	Jabber Developer's Handbook
Dana Moore	Division Scientist	Professional Rich Internet Applications: AJAX and Beyond (Programmer to Programmer)
Edward Benson	Software Developer	Professional Rich Internet Applications: AJAX and Beyond (Programmer to Programmer)
Dana Moore	Division Scientist	Peer-to-Peer: Building Secure, Scalable, and Manageable Networks

access point that accepts queries targeted for a provided domain ontologythat defines the concepts that the query refers to. SQD decomposes these queries into workflows of sub-queries specific to each configured data source and its particular ontology. Decomposition decisions are based on SWRL [12] rules that map data source ontologies into the master domain ontology. Finally, SQD merges its sub-query results and provides a unified query response in the language of the domain ontology.

Fig. 2 depicts the overall system architecture we used for our test. This architecture allows us to send SPARQL queries to SQD using a single domain ontology (available at [28]) and have those queries automatically distributed across the three data sources described above.

With SQD configured for our three test data sources, we posed our SPARQL query: who are all of the BBN employees who are also Semantic Web developers, and what books have they authored? This query relies on our mock employment information from the SPARQL endpoint, project membership data from SemWebCentral, and authorship information from Amazon.com's REST service.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://semanticrest.org/example/domain-ontology#>
SELECT ?name ?position ?title
WHERE
{
  ?person a :SemanticWebDeveloper ;
     :employedAt :BBN ;
     :position ?position ;
     :name ?name .
 ?book a :Book ;
     :author ?name ;
     :title ?title .
}
```

Listing 5 - Semantic query across multiple data sources

Performing the query causes SQD to divide work among the three services based on our configuration file and ontology mappings. Mock employee data is fetched from an endpoint powered by the Joseki SPARQL server [23]. The SemWebCentral developer information is stored in a mock version of SemWebCentral and is accessible for query from the class-level Semantic REST endpoint for users. Finally, the book information is fetched from Amazon.com's live REST service through an SBWS proxy. All together, the information that returns is reproduced in part in Table 4.

This success of this query clearly demonstrates the viability and potential of integration between Web 2.0 and Semantic Web. It took two data sources from the Web 2.0 community, an open-source development site and Amazon's REST service, and it seamlessly merged them with a strictly Semantic Web data source for a unified query. Further it made use of two integration methods: the client-side SBWS approach for Amazon.com's existing REST-based service and the server-side Semantic REST approach for the mock version of SemWebCentral.

7. Related work

The Semantic Bridge for Web Services functions somewhat like a semantic web service matchmaker. There have been several service matchmakers that have been developed such as OWLS-MX [13] and the OWL-S IDE [17]. These, however, focus on more generic service matchmaking than SBWS. SBWS is more of a query engine using a Web service as a knowledge base than a discovery tool. SBWS uses signature matching as it extracts concepts and data from the query to match and ultimately invoke a set of Web service operations to provide an answer, while the more conventional service matchmakers take a request of desired inputs and outputs and provide a corresponding service using signature matching, reasoning, or some other hybrid mechanism to match the service.

Past approaches to the topic of merging the semantic web with the existing world-wide web have often focused on the extraction of semantic content from HTML-based pages rather than the coexistence of HTML and RDF services as two different access mediums for the same data. MIT's Piggy Bank plugin for Firefox uses pre-defined "scrapers" to extract semantic information out of the DOM structures of popular web sites such as Flickr and Amazon.com, for example [7]. Other examples use a combination of user-guided training and tree-based algorithms to learn how to scrape data from a site that presents multiple data objects of the same type with the same basic DOM layout [9][solvent].

Microformats [1] are a way for web scrapers and HTML developers to meet in the middle. Developers embed lightweight semantic markup into the class attribute of HTML elements (generally used for CSS styling). These markers provide a standard context through which to interpret the contents of the HTML tag. Semantic REST offers a different approach than the above tools because it focuses on the co-existence of HTML and RDF rather than a method for extracting RDF from HTML content. This different approach attempts to retain the capabilities and flexibility that make RDF and HTML attractive data formats in the first place.

8. Conclusion and future work

The returns that will come from integrating the data modeling of the Semantic Web with the user participation of Web 2.0 will be far greater than either of the two component parts. Semantic Web users will be able to perform queries and analysis across a wealth of live data from various web sites that they likely use every day. Web applications will also benefit from the ability to use semantic mapping to incorporate external data into their own services.

This paper presented the case for integration between these two worlds and the reasons why Representational State Transfer is an ideal common ground on which to perform that integration. We then provided two alternative and complimentary strategies for performing this REST-based integration. Together these two techniques provide a way to update existing web sites and to construct new hybrid endpoints, as shown by our experiments with a sample query.

Several real-world issues are not addressed with this work and must inevitably be solved before these technologies gain widespread use. Authentication and authorization is absent from both Semantic REST and SBWS as presented here. A web site or a data repository must control both who can access information and what type of information that user can access. Addressing these issues is a significant task and should be the focus of future work so that a SemWebCentral user could not make the assertion that, for example, they were of rdf:type semwebcentral:Admin. The Semantic REST model outlined here also uses a suggested addition to SPARQL that has not yet been added to the proposed SPARQL standard. For the full breadth of operations necessary for Semantic REST to serve as a complete data endpoint, SPARQL/Update or some equivalent must be officially adopted.

Acknowledgements

We sincerely thank and acknowledge Doug Reid and Matt Fisher for their review of this work, Dave Kolas for his assistance with SQD and our test configuration, and Troy Self for his ideas and help in experimenting with semwebcentral.org.

References

[1] J. Allsopp, Microformats: Empowering Your Markup for Web 2.0, Friends of ED, Berkeley, CA, 2007.
[2] R.T. Fielding, R.N. Taylor, Principled design of the modern web architecture, ACM Trans. Internet Technol. (TOIT) 2 (2) (2002) 115–150.
[3] D. Beckett (Ed.), RDF/XML Syntax Specification (Revised), W3C Recommendation, February 10, 2004.
[4] D. Box, D. Ehnebuske, G. Kakivaya, et al. Simple Object Access Protocol (SOAP) 1.1, W3C Note, May 8, 2000.
[5] R. Chinnici, J. Moreau, A. Ryman, S. Weerawarana (Eds.), Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Proposed recommendation, May 23, 2007.
[6] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, Web Services Description Language (WSDL), W3C Note, March 15, 2001.
[7] H. David, M. Stefano, K. David, Piggy Bank: experience the semantic web inside your web browser, in: Proceedings of the International Semantic Web Conference (ISWC), 2005.
[8] J. Farrel, L. Holger (Eds.), Semantic Annotations for WSDL and XML Schema, W3C Working Draft, April 10, 2007.
[9] H. Geng, Q. Gao, J. Pan, Extracting content for news web pages based on DOM, IJCSNS Int. J. Comput. Sci. Network Security 7 (2) (2007).
[10] M. Hadley, Web Application Description Language (WADL), November 9, 2006, available: http://wadl.dev.java.net/wadl20061109.pdf.
[11] D. Heinemeier Hansson, World of Resources, RailsConf 2006 Keynote Presentation.
[12] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission, May 21, 2004.
[13] M. Klusch, F. Benedikt, K. Sycara, Automated semantic web service discovery with OWLS-MX, in: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, 2006.
[14] D. Kolas, Query Rewriting for Semantic Web Information Integration, in: Proceedings of the Information Integration on the Web workshop, Twenty-Second AAAI Conference on Artificial Intelligence, 2007.
[15] D. Martin, A. Ankolekar, M. Burstein, OWL-S 1.1 Release, November 2004, available: http://www.daml.org/services/owl-s/.
[16] L. McGuinness, L. Deborah, F. van Harmelen (Eds.), OWL Web Ontology Language, W3C Recommendation, February 10, 2004.
[17] N. Srinivasan, M. Paolucci, K. Sycara, Semantic Web Service Discovery in the OWL-S IDE, in: Proceedings of the 39th Hawaii International Conference on System Sciences, 2006.
[18] E. Prud'hommeaux, A. Seaborne (Eds.), SPARQL Query Language for RDF, W3C Working Draft, March 26, 2007.
[19] Amazon Web Services, Amazon.com, available: http://aws.amazon.com.
[20] Asio Distributed Semantic Query, BBN Technologies, available: http://asio.bbn.com/sqd.html.
[21] Asio Semantic Bridge for Web Services, BBN Technologies, available: http://asio.bbn.com/sbws.html.
[22] Asio Tool Suite, BBN Technologies, available: http://asio.bbn.com/.
[23] Joseki: A SPARQL Server for Jena, available: http://www.joseki.org/.
[24] M. Dean, P. Neves, DAML DB, September 2001, available: http://www.daml.org/2001/09/damldb/.
[25] A. Seaborne, G. Manjunath, SPARQL/Update: a language for updating RDF graphs, Version 2, April 24, 2007, available: http://jena.hpl.hp.com/∼afs/SPARQL-Update.html.
[26] R. Battle, Semantic Web + Web 2.0 Example Amazon WADL, May 20, 2007, available: http://semanticrest.org/restful-rdf/paper-example/amazon.wadl.
[27] R. Battle, Semantic Web + Web 2.0 WADL Annotations, August 31, 2007, available: http://semanticrest.org/restful-rdf/paper-example/sbws-wadl.xsd.
[28] E. Benson, Semantic Web + Web 2.0 Example Domain Ontology, May 20, 2007, available: http://semanticrest.org/restful-rdf/paper-example/domain-ontology.
[29] E. Benson, R. Battle, Semantic Web + Web 2.0 Example Amazon Ontology, May 20, 2007, available: http://semanticrest.org/restful-rdf/paper-example/amazon-ontology.