

Operationalizing the Semantic Web: A Prototype Effort using XML and Semantic Web Technologies for Counter-Terrorism

Submitted to the:
Second Semantic Technologies for eGov
Conference

Mike Personick, Bryan Thompson, Brad Bebee
SAIC/Advanced Systems Concepts
3811 N. Fairfax Drive, Suite 850
Arlington, VA 22203
703-469-3400
{personickm, thompsonbry, bebeeb}@saic.com

Bijan Parsia
The University of Maryland, College Park
Maryland Information and Network Dynamics Laboratory, Semantic Web Agents Group
8400 Baltimore Ave., Suite 200
College Park MD 20742
bparsia@isr.umd.edu

Curt Soechtig
ObjectSciences Corporation
6359 Walker Lane, Suite 100
Alexandria, VA 22310
curt.soechtig@objectsciences.com

August 6, 2004

1 Introduction and Business Challenge

In the domain of counter-terrorism, understanding terrorist groups and actors, along with their links and relationships, is critical to being able to discern and thus reveal terrorist group structure. Within the Intelligence Community, different organizations have different sources and repositories where each repository might contain potentially new information relating to a particular terrorist group or actor. Furthermore, each analyst can have his or her own preference or favored tool for viewing and analyzing these organizations and actors. The ability to share such information repositories using a diverse set of applications and users will enhance understanding of terrorist organizations; which will occur more quickly and in new ways. Ultimately, it is hoped that this will result in more effective intelligence targeting and subsequent disruption of terrorist operations.

In work being performed for a government customer, SAIC/ASC is developing and participating in an experimental network to enable research technologies to be tested by real users with real data. This environment represents a unique opportunity to experience—and, potentially, solve—the classic challenges that exist in facilitating information sharing within the Intelligence Community. Each new information resource brought online within the experimental network has its own schema and unique conventions for how the information is stored. Further exacerbating the situation, each of the research tools designed to help analysts visualize and understand terrorist organizations has its own internal format in which it processes information.

In bringing repositories onto the network for sharing and later analysis with a variety of research tools, the SAIC/ASC team encountered three distinct challenges. The first is the lack of description of repository schemas. This presents a barrier to information sharing in that it makes mediation between the terms and meanings of different user communities virtually impossible. The second is the tight coupling between analytical tools and a specific schema, which prevents users from persisting and contributing insights from a tool back into their respective organization's repositories. In addition, it makes it difficult to provide tools for analysts to view information from multiple, distributed repositories. Finally, within the counter-terrorism domain, it is difficult to have an *a priori* knowledge of the attributes and links that are required to be stored. As seemingly innocuous information about a terrorist group might help to make a critical link during analysis, repositories in this domain simply must be robust enough to facilitate the addition of new attribute and link information.

The team's initial approach to sharing information on the experimental network used a virtual, denormalized relational database approach for creating repositories of link and group information. This approach provided a highly flexible mechanism for persisting information contributed on the network. However, the flexibility and weak-typing also proved to be its weakness. While it did provide a separation of concerns between the applications and persistence mechanisms, applications using the schema were coupled tightly to specific instance values of information as the result of a lack of information description.

To enable a solution to this challenge of sharing information and providing a rich set of analytical tools, the SAIC/ASC team has initiated prototyping efforts using the XML specification XPointerⁱ and two Semantic Web languages: the Resource Description Framework (RDF)ⁱⁱ and the Ontology Web Language (OWL).ⁱⁱⁱ The outcome of the efforts, described in this paper, will be applied on an experimental network with real users using Semantic Web technologies with real data to analyze and better understand terrorist organizations and their relationships. The team believes that the application of Semantic Web technologies will enable the describing and mediation of semantics for information sharing where other efforts have not succeeded to date.

This paper presents an overview of the technologies that have been applied prior to the selection of the XPointer and RDF/OWL for the team’s prototyping efforts. It provides an in-depth discussion of the key design and architectural considerations as well as directions. Finally, an overview of the current status and future directions is provided.

2 The Initial Relational Strategy

The initial approach used a virtual, denormalized relational database schema known as the Evidence Database (EDB).^{iv} This schema was selected to facilitate information sharing on the experimental network. As repositories were made available on the network, each new schema was loaded into the EDB.

The structure, shown below in Figure 2-1, is reasonably robust for representing entity and link information. It contains first-class data objects consisting of “entities” and “links,” along with sets of metadata “attributes” for those entities and links. Beyond attribute sets, other metadata present includes type information and provenance/workflow information, including the data’s source and report (date and author). As the original schema was very weakly typed, the demand for stronger typing led to the development of six subclasses of entities to represent common types such as people and places. This strong typing severely compromised performance, since a seven-way join became required to obtain almost any piece of information (any query involving an entity, which is most all of them).

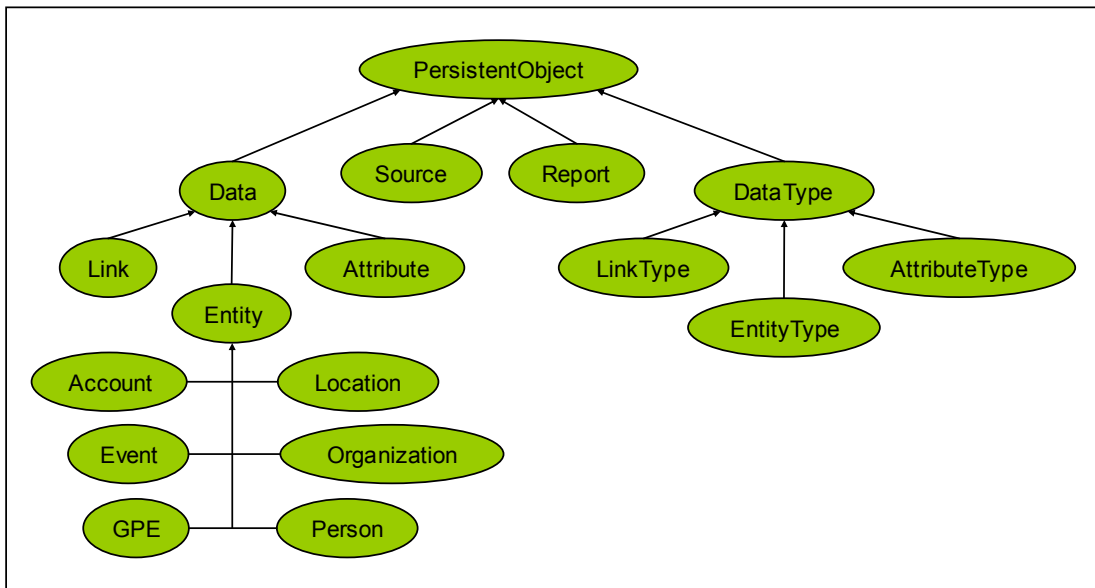


Figure 2-1. An Overview of the Initial Relational Approach Used For Entity and Link Information

As the usage of this schema has progressed, it has become increasingly difficult to achieve a separation of concerns between applications analyzing the information and the instances stored in the schema. The flexibility provided by denormalization results in an extremely tight binding between application and instance information. Initial attempts to address the challenges used object-relational mapping technologies such as Hibernate.^v However, rather than *solving* the problem, this approach simply *shifts* it from the database layer into object layer.

Performance problems necessitated changes to the database schema, but tight coupling between application code and the database made changing the relational schema very difficult. An abstraction layer in between the application and persistence layer was required.

3 Solution Architecture

Based on the challenges presented by using a relational repository, the SAIC/ASC team set out to conceive a solution architecture to address three primary concerns:

1. Promote separation of concerns between application layer and data persistence layer.
2. Dramatically improve query performance.
3. Maximize overall system scalability.

The key elements within the solution architecture are a Web services mechanism for accessing resources, a language for representing the semantics of the resources, and a rich mechanism to enable the querying of distributed resources as Web services. A conceptual overview of the solutions architecture is depicted in Figure 3-1.

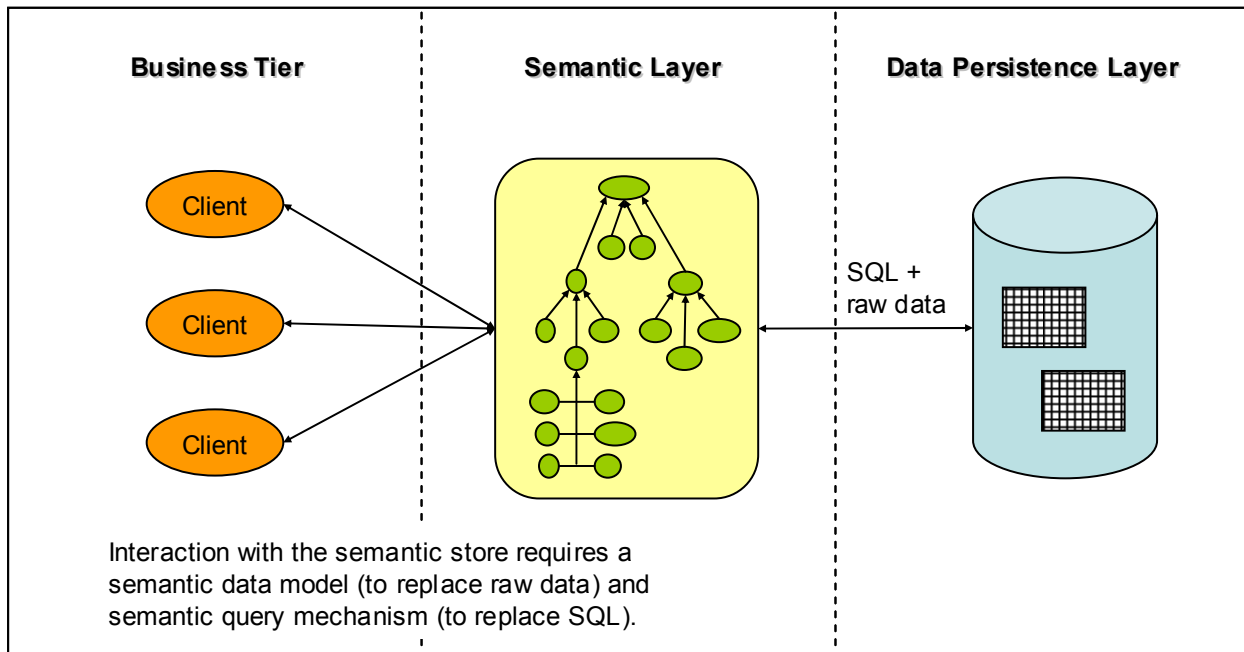


Figure 3-1. An Overview of the Solution Architecture

Initial design efforts are focused on the architecture of the semantic layer. This can be considered the binding between the description of the information and the persistence mechanism. While the specifics of the implementation will differ depending upon the nature of the persistence mechanism, the interface to the application layer will always be loosely coupled. The semantic layer can be thought of as a Web service that acts as an intermediary between the application layer and the underlying relational store. For this prototype, the architectural style known as REpresentational State Transfer (REST)^{vi} was chosen in designing this Web service layer.

The REST architectural style has emerged based on the simple concepts that have made the World Wide Web such a successful large-scale architecture. Where other federated systems have failed to achieve scalable interoperability, the Web has provided a means of exchanging information whereby clients access servers using a very simple protocol (HTTP) that supports the basic resource manipulation operations (GET, POST, PUT, DELETE). A resource-centric interface with simple semantics makes anarchic scalability possible, as all servers and clients agree to a uniform and simple interface definition. This forms the foundation of the REST architectural style.

Another key component of the REST architectural style is the "Simple Uniform Interface." Following Fielding in the HTTP RFC 2616, there are five primary interface characteristics that make up the "REST Uniform Interface."^{vii} These are:

1. Resource is the unit of identification (document centric).
2. Resource state is manipulated through the exchange of representations (document interchange).
3. Generic interaction semantics (create, update, read, and delete).
4. Self-descriptive messaging (supporting processing by intermediaries, e.g., caches and security firewalls).
5. Hypermedia is the engine of application state (hyperlink traversal plus form-based data submission). This is shown in Figure 3-2.

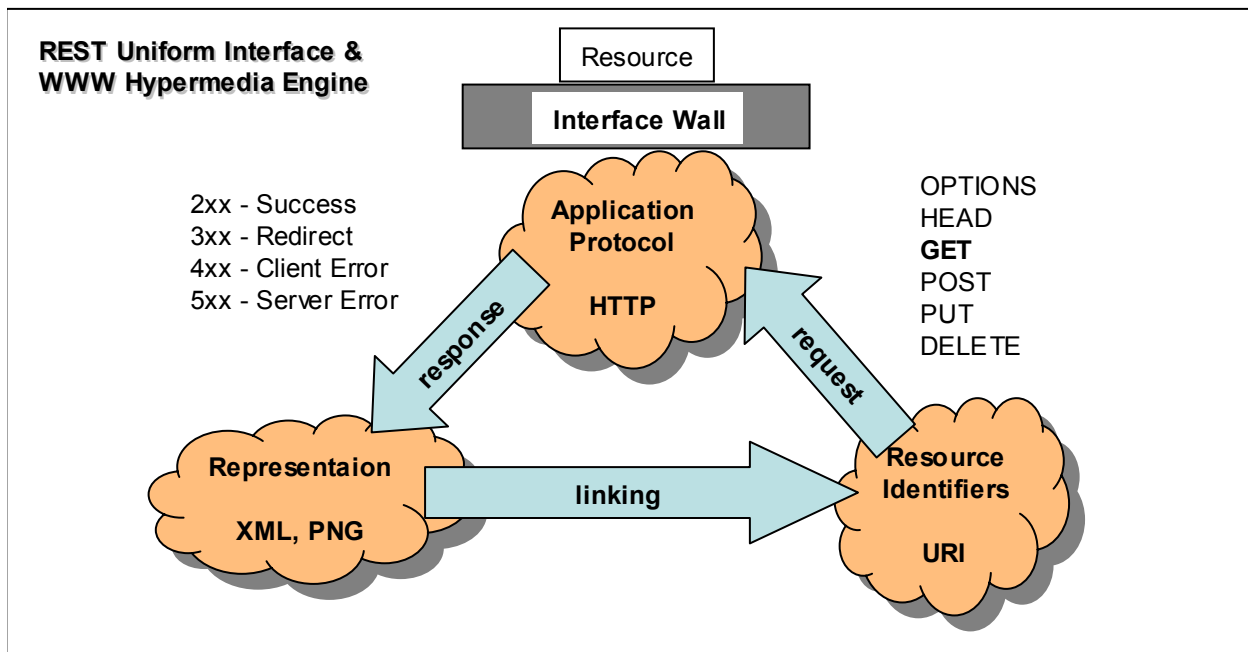


Figure 3-2. An Overview of the REST Uniform Interface and the "Hypermedia Engine"

The SAIC/ASC team believes that the REST architectural style and the semantic Web are synergistic concepts. Taking a resource-centric worldview, where resources are described by well-understood ontologies, is the foundation of semantic interoperability. The most important contrast with the REST architectural model is the Remote Procedure Call (or RPC) model. The RPC model attempts to take the local programming model of a function call and make it work across a network. The success of REST suggests that it has superior characteristics of scalability and mass adoption. It is critical to note that such a resource-centric worldview represents a wholly different perspective than protocols such as SOAP or WSDL could provide.

3.1 Semantic Data Model: RDF and OWL

The next decision in designing the solution architecture was how to model the semantics of the data. The EDB's flexible ontology lends itself well to a graph-centric model. As it is generally the most well understood and widely used language within the Semantic Web community, RDF was chosen as the technology for the semantic model.

“RDF is a language designed for making semantic assertions about resources in the World Wide Web. However it has evolved into a more general purpose language for making semantic assertions about any URI-addressable resource. RDF statements are composed of triples, with a subject, predicate, and object. For example, the triple <edb:Person rdfs:subClassOf edb:Entity> means that the subject, the “Person” class within the “edb” namespace, is related to the object, the “Entity” class within the same namespace, by the predicate “subClassOf”, defined within the “rdfs” namespace (the RDF Schema vocabulary). A triple is also the foundation for an RDF graph, where subjects and objects are graph nodes and predicates are graph arcs. RDF Schema is one way of imposing constraints on an RDF data model. For example, by making the previous assertion we have imposed a constraint on the EDB vocabulary such that the Person class is a subclass of the Entity class. OWL is another vocabulary that is used in conjunction with RDF Schema to provide finer-grained control over an RDF data model, for example OWL allows for cardinality, strong-typing, class axioms, and versioning constraints. RDF/XML is a common serialization syntax for RDF, although other syntaxes (such as N3) do exist.”ⁱⁱ

In creating the RDF vocabulary for a previous relational model, EDB, RDF Schema was applied to define properties and classes; OWL was used to further constrain the relations, cardinality, and typing of the properties and classes within the ontology. For example, the base class of all persistent EDB objects within the EDB namespace is “PersistentObject.” These objects are first-class resources, and thus are subclasses of the RDF resource class. The class PersistentObject has a property called “uniqueId,” which has cardinality of exactly one. The RDF graph for this class is as follows:

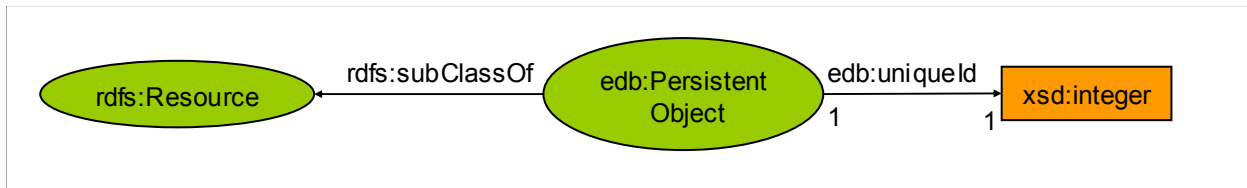


Figure 3-3. RDF Graph for the EDB Persistent Object Class

The RDF/XML definition of the PersistentObject class and its property uniqueId is as follows:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:edb="http://localhost/edb-schema#">
  <owl:Ontology rdf:about="">
    <dc:title>The EDB-RDF Vocabulary (RDF)</dc:title>
    <dc:description>This is the RDF Schema for the EDB vocabulary
      defined in the EDB namespace.</dc:description>
  </owl:Ontology>
  <owl:Class rdf:ID="edb:PersistentObject">
    <rdfs:comment>The class of EDB persistent objects.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="rdfs:Resource"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#uniqueId" />
        <owl:cardinality rdf:datatype="xsd:integer">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="uniqueId">
    <rdfs:comment>The unique id for an EDB object.</rdfs:comment>
    <rdfs:domain rdf:resource="#PersistenObject" />
  </owl:ObjectProperty>
</rdf:RDF>
```

The combination RDF Schema and OWL offers a powerful vocabulary for ontology definition. The class of EDB persistent objects is defined as a subclass of the class `rdf:Resource` and another “anonymous” class of resources that have a property called “`uniqueId`” with a cardinality of exactly 1. The property `uniqueId` is defined as having a domain of `edb:PersistentObject` and a range of `xsd:integer`.

What this means is that whenever an RDF statement is made with `edb:uniqueId` as the predicate, the subject must be of type `edb:PersistentObject` and the object must be of type `xsd:integer`. The RDF vocabulary description language class and property system is similar to the type systems of object-oriented programming languages such as Java. RDF differs from many such systems in that, rather than defining a class in terms of the properties its instances might have, the RDF vocabulary description language describes properties in terms of the classes of resource to which they apply. This is the role of the domain and range mechanisms described in this specification. Using the RDF approach, it is easy for others to define subsequent additional properties with a domain of `edb:PersistentObject` without the need to redefine the original description of the class. One benefit of the RDF property-centric approach is that it allows anyone to extend the description of existing resources—one of the architectural principles of the Web.

3.2 Semantic Query Mechanism: Server-Side XPointer

Once the semantic data model has been defined, the repository can be treated as a semantic store with its own flexible ontology and easily represented by a graph-centric data model. However, queries against the database were previously performed in a fashion that was tightly coupled to the actual relational database model, rather than the semantic layer that the implementation describes. To migrate to a semantic store, the solution architecture needed a query mechanism suitable for the graph-centric model.

In a resource-centric worldview, the entire contents of a repository instance can be thought of as a very large graph, addressable by a Uniform Resource Identifier (URI) and serializable as RDF/XML that conforms to the RDF vocabulary discussed in the previous section. However, this approach to reading EDB data is intractable due to the size of the graph. This is particularly true within the counter-terrorism domain. A method was needed to constrain the request so that only a selected subgraph is transmitted back to the client.

The traditional Web is made up of two kinds of interactions: hypermedia navigation and submitting data to arbitrary processes. These interactions map directly onto the HTTP/GET request and the use of the HTTP/POST request to send form data to a server where it is consumed by an arbitrary process. So, what changes in the semantic Web? First, clients are not acting as direct agents for human users. Such machine clients can be citizens of the Web using the same protocol to interchange documents. However, resources are polymorphic—they can act as both interchanged documents rendered for human consumption and as representations of machine-processable graph models. In considering the counter-terrorism domain, use cases exist where the semantic stores are very large (e.g., the EDB). Furthermore, there are use cases where the semantic stores are derivative views of other resources. In both of these use cases, extensible addressing schemes are required to provide for ad hoc processing and use of resources in analytical processes.

Once a resource-centric view of the repository is determined, an extensible means of providing arbitrary addressing is required. The W3C XPointerⁱ framework provides an extensible processing model for URI fragment identifiers (`#foo`, etc.). One of the more common misconceptions is that XPointer is a variation on XPath.^{viii} XPath is a specific language for identifying parts of an XML data model (or info set). XPointer is an extensible framework for

addressing XML subresources. The closest area of overlap is the xpointer() addressing scheme, which has not yet matured to a W3C Recommendation. The W3C has defined several XPointer addressing schemes: for example, element(), xpointer(), and svgView(). In addition, the xmlns() scheme allows other organizations to define their own addressing schemes in their own namespaces. For example, one could define an XPointer scheme that tunneled the XPath Recommendation.

This use of namespaces is part of what makes XPointer so extensible. The other extensible dimension is that multiple pointer parts can be specified in a single fragment identifier. This makes it possible for a client to ask for the same information in several different ways and have the XPointer processor select the first pointer part that actually identifies a subresource in the addressed resource representation.

XPointer provides an extensible processing model for URI fragment identifiers:

- #foo
- #element(foo/2)
- #svgView(0,0,100,100)
- #xmlns(x=http://www.myorg.org)x:xpath(//title)
- #xmlns(x=http://www.myorg.org)x:rdf-query(...)
- #xmlns(q=http://www.myorg.org)q:tm-query(...)

The most significant problem facing XPointer currently is that the URI fragment identifier is not passed with a normal HTTP request. Therefore, the client must GET the entire graph representation and then apply an XPointer processor to interpret the fragment identifier. Thus, client-side XPointer processing, while certainly extensible, is not scalable. The SAIC/ASC team's prototype under development implements server-side XPointer processing. HTTP defines a "Range" header that can be used to request a range of the negotiated resource representation. Our solution is to copy the fragment identifier into this Range header and indicate to the server that the range is being expressed using XPointer. At this point, the semantic service layer has all of the pieces and it applies the XPointer expression (our query) and sends back only the triples actually identified by that query (rather than the entire triple store). This server-side XPointer solution to querying the semantic store is both extensible *and* scalable.

3.3 Semantic Query Language: RDQL and DIG

Finally, the solution architecture includes a means for semantic queries using XPointer. It provides a highly extensible query mechanism that will allow for complex queries against a graph-centric model. However, XPointer alone is not the entire solution. Query semantics must be defined and used within XPointer to define meaningful views of the data model. Once again, one should not confuse XPointer with XPath. Although the serialization of the EDB semantic store is RDF/XML, writing an XPointer scheme that uses XPath to query against the actual XML syntax would be a mistake. Using the XML syntax for addressing schemes means the query mechanism is bound to a specific XML schema. For example, tunneling XPath will break if a version of an XML schema is changed or a non-XML representation is used. Thus, the XPointer schemas should be based on the data model's *semantics* rather than its *syntax*. Two candidates for the semantic query language are RDF Data Query Language (RDQL)^{ix} and the DL Implementation Group's Description Logics Interface (DIG).^x

RDQL is a W3C RDF data query language developed to extract information from RDF graphs. An RDQL query consists of a graph pattern, expressed as a list of triple patterns. Each triple pattern is comprised of named variables and RDF values (URIs and literals). An RDQL query can additionally have a set of constraints on the values of those variables, and a list of the

variables required in the answer set. Several implementations of RDQL exist, the most notable being JENA.^{xi} A sample of the HTTP request-response for an RDQL-XPointer query might look as follows:

```
GET /2004/xptr/foaf.rdf HTTP/1.1
Host: www.mindswap.org
Accept: application/rdf+xml
Range: xpointer = xmlns(ms=http://mindswap.org)
      ms:rdql( SELECT ?entity WHERE (?entity edb:uniqueId "18746") )

-----
HTTP/1.1 206 Partial Content
Content-Type: application/rdf+xml
<!-- Only the selected sub-graph is transmitted to the client. -->
<rdf:RDF ... />
```

The above query selects the subject from all RDF triples that match the patterns described in the where clause (where the subject has an edb:uniqueId of 18746). The data transmitted back to the client would be the serialized RDF/XML that represents entity #18746 in the EDB database.

The other candidate technology, DIG, is based on work done by the DL Implementation Group, a self-selecting assembly of researchers and developers associated with implementations of Description Logic systems. The “mid-weight” version of OWL, OWL-DL, is a description logic system that provides the grammar necessary to define an ontology that can be mapped onto DIG queries. By using OWL to define the EDB ontology, DIG becomes another candidate for the query language through XPointer.

4 Current Status and Reflections

The outcome of the efforts described in this paper will be applied on an experimental network with real users using Semantic Web technologies with real data to analyze and better understand terrorist organizations and their relationships. Migration to the solution architecture described in the previous sections is an ongoing process expected to be demonstrated in December 2004. To date, an initial draft of an RDF vocabulary for the EDB data model has been defined. Once the semantic graph model is fully defined, we will then develop the XPointer query language that will be ultimately implemented in the semantic service layer. The server-side XPointer evaluation mechanism has been prototyped in an initial implementation.

The SAIC/ASC team believes the application of REST Web services, RDF/OWL descriptions of entity and link information, combined with a semantic query mechanism will result in significant progress in being able demonstrate enhanced information sharing capabilities on the experimental network mentioned in the initial section. Our confidence is based on three areas:

1. The semantic data model and the query mechanism used to access it will remain fixed for the duration of the project, allowing for a complete separation of concerns of the application layer and database.
2. This separation of concerns will allow the database team to create denormalized views of the database, which will significantly enhance query performance.
3. By implementing the semantic service layer according to the REST architectural style, very loose coupling, maximum flexibility, and anarchic scalability can be achieved.

Another important question to ask is whether these goals could have been achieved without the use of semantic technologies—the answer to which is a resounding “no.” As long as the

interface between the business layer and the data persistence layer remains below the top level of abstraction (the semantics), tight coupling will constrain flexibility, scalability, and performance.

5 Future Work: Federated Semantic Stores

The prototyping efforts described in this paper will enable loose coupling between applications as well as well-described information within repositories. Within the SAIC/ASC team's work in the counter-terrorism domain, one key area of future investigation is federated semantic stores. This capability would enable not only the mediation of information between repositories and analytical applications, but also federated queries of distributed repositories based on the semantic descriptions. This capability would be enabled by federated semantic stores.

A federated (semantic) store is different in several ways from a traditional resource. In many ways, the closest deployed examples of federated stores are Google™ and RSS/Atom feeds. The federated store is not the *authority* for the source data; rather, it is only a fused view. A federated semantic store is graph-centric, rather than document + metadata centric (as with Atom or Google™), which makes it more challenging to use. Examples of federated semantic stores that emphasize the graph over the documents and their metadata include federated evidentiary databases, federated information on biomedical pathways, and the like.

Each of these systems is concerned both with documents and with the meanings that different agents have ascribed to the assertions made by those documents. Federation is introduced in order to cross trust boundaries and provide the shared data model required for coherent analysis.

In a federated store, the authoritative resources are those that have been federated and those that describe the federation process (i.e., mergeMaps), not the result of that federation. In a semantic store, inferred subgraphs can be encountered that cannot be retracted or updated without modifying the authoritative data or the rules that produce them. In addition, clients might need the federated store to update the authoritative resources from which it is derived, even if those changes will be lost the next time the authoritative data is re-federated.

6 Conclusions

The prototyping the SAIC/ASC team initiated using semantic technologies has specific promise for facilitating the analysis of terrorist groups and organizations. We believe the unique combination of using Semantic Web technologies on an experimental network will result in a tangible demonstration of the impact and benefits the semantic Web can have for real users and data within the Intelligence Community.

In addition, it can serve as an outline for a general approach to enabling and facilitating information sharing in any domain. SAIC/ASC believes that modeling information as a semantic store, along with taking a resource-centric approach to sharing the information (per the REST architectural style), and allowing extensible server-side logical queries against the semantic resources (using XPointer) without coupling to the serialization syntax (by using an "RDQL-like" technology) provides a tangible and usable framework for general semantic interoperability. Combining this framework with the concept of federated semantic stores provides a general architecture for the entire next-generation semantic Web.

Notes

ⁱ XPointer Framework. W3C Recommendation: <http://www.w3.org/TR/xptr-framework/>

- ⁱⁱ Resource Description Framework. “RDF Primer”: <http://www.w3.org/TR/rdf-primer/>
- ⁱⁱⁱ Ontology Web Language. “OWL Guide”: <http://www.w3.org/TR/owl-guide/>
- ^{iv} Silk, B., Bergert, B. “Evidence Database Description v0.1,” 10 May 2003.
- ^v Hibernate Object Relation Mapping for Java™: <http://www.hibernate.org/>
- ^{vi} Fielding, R. “Architectural Styles and the Design of Network-based Software Architectures”: <http://www1.ics.uci.edu/~fielding/pubs/dissertation/top>
- ^{vii} Hypertext Transfer Protocol—HTTP/1.1, The Internet Society, 1999, Fielding, et al. (Obsoletes RFC 2068): <http://www.ietf.org/rfc/rfc2616>
- ^{viii} XML Path Language. W3C Recommendation. <http://www.w3.org/TR/xpath>
- ^{ix} “RDQL - A Query Language for RDF”: <http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/>
- ^x “DIG Description Logics Interface 1.0”: <http://dl-web.man.ac.uk/dig/2002/10/interface.pdf>
- ^{xi} JENA: A Semantic Web Framework for Java: <http://jena.sourceforge.net/>