Semantic MOBY

Position Paper for the
W3C Workshop on Semantic Web for Life Sciences
27-28 October 2004, Cambridge, Massachusetts USA

Gary Schiltz <gss@ncgr.org>, Damian Gessler <ddg@ncgr.org>
National Center for Genome Resources, Santa Fe, New Mexico USA

Lincoln Stein <lstein@cshl.edu>
Cold Spring Harbor Laboratory, Cold Spring Harbor, New York USA

Introduction

This paper describes Semantic MOBY (www.semanticmoby.org), a Semantic Web-based architecture and open source software project for integrating web-hosted resources in the field of bioinformatics. During its design and development, a number of questions were addressed regarding standards and technologies related to the Semantic Web. The authors hope that lessons learned (discussed in this paper and in [1]) while addressing these questions will be valuable to the W3C and the broader Semantic Web community as they guide the evolution of the Semantic Web.

The MOBY Project

The name MOBY is derived from the acronym MOBY-DIC: Model Organism Bring Your own - Database Interconnectivity Conference. This series of conferences, first held in 2001, was for scientists and engineers working in bioinformatics – a field concerned with the management and analysis of biological data, particularly in genomics and proteomics. Participants sought to formulate approaches to make their respective databases and analysis services widely available to the community, and to make it easier to integrate these resources. Because a large number of bioinformatics data sources and analysis services are available over the Web, integration efforts including MOBY mainly focus on Web-based data and services. This paper uses the term "integration" to refer to the description, discovery, and engagement of bioinformatics resources.

The integration needs of two groups of users, namely bioinformaticians and biologists, are addressed by Semantic MOBY. Bioinformaticians focus on the development and use of software tools for analysis of biological data. They require tool sets to facilitate development of bioinformatics software, particularly applications that engage multiple distributed resources and merge results. For example, an application could perform a BLAST search to find proteins that contain similar amino acid sequences, followed by a literature search for articles that discuss the function of those proteins in connection with a particular disease. Conversely, biologists are primarily concerned with applying their expertise in biology, and even those with significant computer skills may not have programming experience. What is more important is to discover and engage resources provided by bioinformaticians, and if possible, to do so without the necessity of installing or maintaining software. Software aimed at biologists is usually browser-based.

As a result of the first conference, a set of use cases and scenarios [2] were developed to guide the effort. Two complementary approaches to integration were proposed [3]. The first approach, based on the Web Services paradigm, is referred to as "MOBY Services;" that approach is not described here. The second approach, based on concepts from the Semantic Web and dubbed "Semantic MOBY," is the subject of this position paper. Information about the MOBY effort in general (referred to generically as "BioMOBY"), as well as MOBY Services and Semantic MOBY, may be found at www.biomoby.org. Although Semantic MOBY and MOBY Services are specifically targeted at the field of bioinformatics (based on the expertise of the investigators and the agencies that sponsored the project), both architectures are quite general and are suitable for integration efforts in other fields.

Shortcomings of Traditional Architectures

A number of technologies have been used to provide interoperability in the context of distributed software development. A few examples include CORBA [4], Enterprise JavaBeans [5], and Web Services [6], as well as *ad hoc* CGI. Although these technologies are quite different, the Semantic MOBY team identified three seemingly ubiquitous problems with current technologies that hinder scalable interoperability among bioinformatics resources:

- *Fatal mutability of interfaces* — if a provider changes its interface, client code that depends on the signature fails *en masse*. This exhibits the undesirable property that the more clients use an interface (i.e., the more widely adopted it is), the less flexibility providers have in evolving it;

- *Rigidity and fragility of static classification schemes (hierarchies)* — changing the properties of a class near the root of an inheritance hierarchy simultaneously affects the entire subtree. This has the undesirable property that in an open world, where independent parties are arbitrarily adding "is-a" relationships, the nodes near the root, which were added when the system was least evolved, are the least able to change without causing large scale failure;

- *Confounding structure and content* — access to the content information of the data – the "data" itself – is entangled with the presentation layer and implicit behaviors of the presentation software. This has the undesirable property that the data content of value to the client may be difficult or essentially impossible to parse from the data presentation of an arbitrary provider, thereby crippling machine-automated, semantic determination.

Semantic MOBY Philosophy

In juxtaposition with these problems, the Semantic MOBY team was impressed with the overwhelming success of the web – its stateless, method-sparse, document-based architecture demonstrating many desirable properties of robustness, evolvability, and scalability. Our conclusion was to design an architecture that mimicked much of what works for the web – in particular, a document-based architecture with explicit delineations of data and its contextual relationships. A technology assessment phase led the team to the choice of RDF [7] and OWL DL [8] as the underlying technologies.

This essentially recast the problem of interoperability. Instead of specifying a syntax and messaging layer to *connect* clients and providers via a registry lookup, we would provide clients and providers a way to *describe* their data and *identify* data relevant to them. Thus the three-fold challenge to Semantic MOBY became:

- *Deploy a common syntax* — enable clients and providers to engage each other under shared syntactic rules of engagement. The decision was made to use RDF graphs serialized as RDF/XML [9];

- *Develop a common semantic* — enable machine-discernible meaning so clients can request the same conceptual object or service from different providers. The decision was made to develop a set of RDF classes and properties to describe a canonical graph structure that providers and clients can agree upon, and add OWL DL constraints on the classes and properties to enforce the canonical structure;

- *Implement a discovery server* — enable clients to find providers based on the semantics of their data and services. The decision was made to store RDF graphs that describe mapping operations performed by providers, and provide an HTTP interface for searching the repository.

The result is a document-based design that uses a W3C-syntax middle layer for a semantically rich encoding of data and service descriptions, operating under the constraints listed earlier.

Canonical Graph Structure

Software agents in Semantic MOBY communicate using RDF graphs whose statements contain instances of MOBY-defined classes and properties [10]. Graphs referred to as *provider description graphs* are used to describe mapping relationships defined by service providers, and have a canonical form defined by a set of OWL DL restrictions. Certain RDF blank nodes in graphs hold information (the *Subject* of the mapping) that the provider requires to carry out its function; other blank nodes represent information that the provider agrees to supply as results (the *Object* of the mapping). Requiring that providers be described by canonical graphs facilitates the writing of parsers that can unambiguously identify mapping relationships for software agents to manipulate. The general form of a Semantic MOBY provider description graph is illustrated by the following N3 [11] code that describes a hypothetical publication search provider aimed at mapping a gene symbol string to a literature guide URI string (note that some namespace prefixes are fictitious):

```
@prefix acmepubs:   <http://www.acmepubs.com/> .
@prefix acmeterms:  <http://www.acmepubs.com/terms/> .
@prefix pubs:       <http://www.pubs.org/terms/> .
@prefix go:         <http://www.go.org/terms/> .
@prefix sgd:        <http://www.sgd.org/terms/> .
@prefix moby:       <http://www.semanticmoby.org/ontologies/core/> .
@prefix rdf:        <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

acmepubs:citation-search
      a moby:Provider, pubs:CitationSearch ;
      moby:name "AcmePubs Citation Search" ;
      moby:oneLineDescription "Industry standard citation search" ;
      moby:operatesOn [
          a rdf:Bag ;
          rdf:_1 [
              a moby:Graph ;
              moby:hasMapping [
                  a moby:Subject, go:GeneSymbol ;
                  acmeterms:geneSymbol _:bnode1 ;
                  moby:mapsTo [
                      a moby:Object, sgd:LiteratureGuide ;
                      acmeterms:literatureGuideURI _:bnode2
                  ]
              ]
          ]
      ] .
```

This graph asserts that the URI `http://www.acmepubs.com/citation-search` represents a `Provider`, which is also an instance of the class defined at `http://www.pubs.org/terms/CitationSearch`. This provider operates on an RDF `Bag`, the first element of which is a `Graph`. This graph node has a mapping headed by a `Subject` node, which is also a `GeneSymbol`; this node has a `geneSymbol` property whose value is the blank node `_:bnode1`. The gene symbol node maps (via a `mapsTo` property) to a node that is an `Object` as well as a `LiteratureGuide`. Finally, this literature guide node has a `literatureGuideURI` property, whose value is the blank node `_:bnode2`.

Semantic MOBY Architecture

Semantic MOBY defines four roles for software agents that use its architecture: *(1) Service Providers* use the architectural conventions to make their services available over the web; *(2) Ontology Providers* supply shared, web-accessible definitions of OWL classes and properties that are the building blocks from which graphs are formed; *(3) Discovery Servers* maintain searchable repositories of descriptions of service providers; and *(4) Service Consumers* query discovery servers to find service providers that meet their requirements, and engage service providers to access the required services. Our implementation of Semantic MOBY additionally provides an *Invocation Broker* that collaborates with the discovery server to provide browser-based user interfaces and presentation interfaces for service providers on behalf of end users.

These agents communicate via HTTP, sending and receiving canonical RDF graphs as the payload of the messages. Importantly, these software agents do not send a different graph, with different semantics, for each communication. Instead, they operate on what could be described as a *single, mutable graph*. A service consumer sends a graph to a discovery server that describes the essential topology of providers of interest. The discovery server returns a list of copies of the query graph, where each copy is merged with the statements in a matching provider description graph. At that point, the consumer selects a matching graph, augments it by adding inputs, and sends the graph to the provider. Lastly, the provider augments the graph with the results of its actions and returns the graph to the consumer. The remainder of this section describes each of these agents in more detail.

Service Providers use the architecture and its conventions to make their functionality available over the Web. A service provider exists at a given HTTP URI, and is expected to handle requests as follows:

- A GET is a request for the "terms of engagement" for a provider, and should be handled by streaming back, as its response, its provider description graph serialized as RDF/XML. In the reference implementation, a set of Java classes based on the Jena2 [12] library is provided to enable service providers and consumers (see below) to manipulate graphs;

- A POST is a request to invoke a service. The message body should contain a serialized RDF/XML graph, with the same topology as the provider description graph, but augmented by replacing any required blank `Subject` properties with concrete resources or literals. The provider should in turn modify the graph by adding statements to represent the results of the service invocation and stream it back in RDF/XML as the response to the request.

Service Consumers use query graphs to ask discovery servers for lists of providers whose descriptions are subsumed by the query graphs submitted. They then invoke the service by modifying its provider description graph to add information that the provider requires, POST the serialized graph to the provider's URI, and extract the results from the graph contained in the response to the POST request.

Ontology Providers provide definitions of classes and properties that are the building blocks of graphs. Like service providers, they exist at a URI, and are expected to respond to GET and POST requests by streaming back serialized RDF/XML graphs that describe the class or property that they represent. These graphs should only contain statements whose subjects are either the URI of the graph or blank nodes.

Discovery Servers maintain searchable repositories of provider description graphs. The reference implementation provides two discovery servers, described below – an RDF-based server for access by software applications (for the bioinformatics community), and a keyword-based server, accessible with a web browser (for the biology community). Sophisticated discovery servers could populate their repositories by crawling the web, searching for service providers by issuing GET requests and attempting to parse canonical graphs. The reference implementation, with its modest resources, merely provides a *passive registration interface* for suggesting a provider URI (there is a form based interface at www.semanticmoby.org/developer/suggest.html). If the URI returns a valid provider description graph, then all the statements from the graph are added to the repository (if it isn't already there) or updated (if the repository already contains a graph for the URI, with an earlier modification date). If the URI is inaccessible or doesn't parse to a valid provider description graph, the statements from the existing graph (if one exists) for the URI are removed from the repository. This passive registration approach has an advantage over an explicit registration scheme since it prevents malicious agents from registering or deregistering a graph for a provider – the request is simply a suggestion, and whoever controls the URI containing a graph is responsible for the outcome of the suggestion.

The repository, as currently implemented, contains the set of statements from all the graphs registered. Early in the design of the reference implementation, provider description graphs were allowed to contain statements asserting properties about any resource. This presented a problem – since the repository is a *set* of statements, if the same statement were added on behalf of multiple providers, it would be stored only once. But what if one of the providers goes away and its statements need to be removed from the repository? It would be necessary to know when all the providers that made a particular statement went away before removing the "shared" statement. Several solutions were considered, including reference counting of statements and reification. In the end, however, it seemed philosophically appealing to restrict the statements contained in a provider description graph to having either blank nodes or the URI of the provider itself as their subjects. This removes the need to ever store a statement twice, while maintaining the spirit of the controller of a URI being the only one able to make statements that describe the provider at that URI.

The two discovery servers are now described:

> The *RDF-based Discovery Server* operates on the repository of graphs found by the passive registration interface, and accepts POST requests from service consumer applications. The message body of each request should contain an RDF graph that serves as a query; the query graph is converted to an RDQL [13] query, with blank nodes representing RDQL variables. This query is then executed against the repository, and for each set of bindings for the variables, a graph is generated and added to the response. The consumer application can then use the Semantic MOBY class libraries to parse the response into a set of graphs, and finally select one to be used to engage its provider. A forms-based interface to this discovery server is at www.semanticmoby.org/developer/search.html.
>
> The *Keyword-based Discovery Server* searches for providers based on user-supplied keywords. When the passive registration interface finds nodes, in the graph being added, that have certain *rdf*:type properties (namely *moby*:Provider, *moby*:Subject, and *moby*:Object), the nodes are examined for other *rdf*:type properties. The classes (and their superclasses) that are the values of these properties are retrieved (by doing an HTTP GET on their URIs), and are examined to see if they have any *moby*:keyword properties. If so, the keywords are associated with the provider being registered, as well as whether it was associated with the provider, subject, or object node. Although intended primarily to be accessed programmatically, a forms-based interface to this discovery server is at the home page of Semantic MOBY, www.semanticmoby.org.

An *Invocation Broker* service is used in conjunction with the keyword-based discovery service described above to provide a browser-based provider engagement architecture. When the keyword-based discovery server receives a query, it searches for graphs whose `Provider`, `Subject`, and `Object` nodes match keywords in the keyword query and returns an HTML page showing the results. For each matching provider URI, it retrieves the provider description graph from the RDF-based discovery server and uses the graph to display a summary of the provider – its name, description, where to find more information, etc. It also provides a link, with an appropriate query string, to the invocation broker on the Semantic MOBY site. Briefly stated, when the invocation broker is asked to invoke a URI, it retrieves the graph at that URI. If the graph requires no inputs, it is POSTed immediately to the URI and the results are streamed back to the client. If the graph does require inputs, then if the graph contains a *moby*:`inputURI` property, the client is redirected to that URI; if the graph has no such property, then a generic HTML form is presented for the user to edit and submit a serialized RDF/XML graph. The invocation broker is still a work in progress.

Conclusions

The authors believe that research still needs to be done in several areas; we offer the following as starting points: 1) the value of more human-readable graph notation; 2) getting users in the loop through building brokering schemes to generate or identify interfaces for gathering service provider inputs and displaying service provider outputs; 3) the importance of the ability to retrieve and process resource descriptions (classes and properties). This last point is particularly important, since much of Semantic MOBY relies on being able to retrieve and analyze a resource at a URI. This brought to light a question that has been discussed extensively on the W3C lists, namely the merits of "hash URI" notation (a URI containing a fragment identifier, e.g. `http://foo.bar#baz`) vs. "slash URI" notation (a URI that adds an additional level with a slash rather than a fragment identifier, e.g. `http://foo.bar/baz`). Several pages [14] on the W3C Semantic Web Wiki are devoted to this subject. Because Semantic MOBY associates a resource with a machine readable definition at its URI, and needs to frequently retrieve such resources, it stipulates that ontology providers and service providers expose their graphs using slash URI notation.

Although still evolving, the authors believe the open standards of RDF and OWL are well suited to bioinformatics integration. Looking forward, we see great potential in Semantic MOBY in brokering and making useful bioinformatics resources, maintained and populated by expert communities.

Acknowledgments

We thank Mark Wilkinson, Phil Lord, Andrew Farmer, and members of the `moby-dev` mailing list for helpful discussions. This work was funded by NSF grant 0213512.

References

1. Philip Lord, S. Bechhofer, M.D. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C. Goble, L. Stein, *Applying Semantic Web Services to bioinformatics: Experiences gained, lessons learnt*, accepted for publication in proceedings of the International Semantic Web Conference, 2004.
2. BioMOBY use cases and scenarios are at www.biomoby.org/twiki/bin/view/General/WebHome
3. Mark Wilkinson, D. Gessler, A. Farmer, L. Stein, *The BioMOBY Project Explores Open-Source, Simple, Extensible Protocols for Enabling Biological Database Interoperability*, Proceedings of the Virtual Conference on Genomics and Bioinformatics, 2003. The paper is available at www.virtualgenomics.org/proceedings2003/VCGB_130.pdf
4. A FAQ for CORBA is at www.omg.org/gettingstarted/corbafaq.htm
5. Enterprise JavaBeans information is at java.sun.com/products/ejb
6. Web Services information is at www.w3.org/2002/ws
7. RDF information is at www.w3.org/RDF
8. The OWL reference is at www.w3.org/TR/owl-ref
9. The standard XML syntax for RDF is at www.w3.org/TR/rdf-syntax-grammar
10. Semantic MOBY class and property definitions are located at www.semanticmoby.org/ontologies/core
11. An introduction to N3 notation is at www.w3.org/2000/10/swap/Primer
12. Jena information is at www.hpl.hp.com/semweb/jena2.htm
13. Information about RDQL is at www.w3.org/Submission/2004/SUBM-RDQL-20040109
14. Discussion of the "hash versus slash" URI notation is at esw.w3.org/topic/HashVsSlash