Semantic Web for Data Interpretation & Integration: Lessons Learned from Scientific Publishing and the Distributed Annotation System

Position Paper for W3C Semantic Web for Life Sciences Workshop
http://www.w3.org/2004/07/swls-cfp.html

Stephen A. Chervitz, Software Development, Affymetrix, Inc.

Contributors: Melissa Cline, Gregg Helt, Andrew Dalke, Allen Day

Last Updated: Tue Oct 26 11:30:03 2004

Abstract

Scientists are now quite adept at using various "omics" technologies to generate large data sets, creating a bottleneck at the level of data interpretation and integration. This paper addresses the use of semantic web technologies in two areas relevant to this bottleneck: Using RDF to enhance scientific publications and the development of the next version of the Distributed Annotation System (DAS). It also contains some general thoughts on the use of semantic web technologies from the perspective of a commercial tool provider whose tools are used to generate genome-wide data sets, and in so doing, amplify the semantic needs of life scientists.

RDF-Enanced Scientific Publications

In a recent publication, a scientist at Affymetrix (Melissa Cline) provided publically available RDF-formatted data supporing the publication (1). This experience was overwhelmingly positive, and provides an excellent use case for how RDF can enhance the value of scientific communications. It is instructive to document the motivations and lessons learned from this project.

The publication is a study of the impact of splice variation on protein structure in 8000 different mouse proteins. As is typical of most genomics papers, it is most convenient to make the underlying data available via a website rather than in the published article, since it is unwieldy to include it in the article itself, and even if it were, it would be difficult for others to use and build on it. Releasing the data related to a publication at the time of publication has additional benefits: It adds credibility to the publication, provides readers with the means to confirm the results or ask follow-up questions, and gives the data a life beyond the scope of the publication.

To make the data more interpretable and accessible to machine-based reasoning tools, it was released in RDF format (2). The data relates certain motifs in proteins to splicing patterns in the genes that encode them. Since this involves the intersection of two complex phenomena, an ontology was used to describe the data. So, the available RDF files describe a preliminary ontology for relating splice variation and protein annotation.

Advantages of RDF for data encoding

RDF allows an ontology to be defined and distributed in the same file as the data, in a computable form. For contrast, you often see ontologies defined informally, via a README file or other text document. At the end of the day, an ontology that can be computed on is just more useful. RDF ontologies are built on building blocks such as OWL, which yields an ontology that's both specific and concise. RDF has an isomorphic representation called N3 (3) in which the Cline et al. data is also available. N3 is much more readable than RDF-XML and much more human-interpretable than a tab-delimited file, yet it still has a clear computational interpretation.

The table below compares RDF and N3 representations of a gene and its splice variants, highlighting the better readability of N3.

RDF

```
<Gene rdf:about="#Nox4">
   <chr>chr7</chr>
   <hasVariant rdf:parseType="Resource">
      <representedBy rdf:resource="#gi14582296"/>
   </hasVariant>
   <hasVariant rdf:parseType="Resource">
      <representedBy rdf:resource="#gi18204713"/>
   </hasVariant>
   <strand>+</strand>
</Gene>
```

N3

```
:Nox4 a affx:Gene ;
    affx:strand "+" ;
    affx:chr "chr7" ;
    affx:hasVariant [affx:representedBy :gi14582296] ;
    affx:hasVariant [affx:representedBy :gi18204713] .
```

Ontologies in RDF are easy to maintain, especially when compared to XML. In RDF, if you choose to apply a property to a different element, it's a very lightweight change: take a link that connects two nodes, and move it to two different nodes. In XML, it involves rebuilding the whole tree.

RDF makes it easy to extend the original data, without moving it into a different framework. For instance, in RDF one can write

rules or functions for operating on the data. For example, there could be RDF rules for identifying transcripts that are candidates for nonsense-mediated decay (NMD), and thus might not be translated to protein. The biological rules for identifying an NMD candidate are simple and easy to encode. RDF provides an automated means for tagging such data as "chaff", and removing it from further consideration.

The RDF rules are encoded in the same framework as the data and ontology, so these components are much less likely to become separated over time. For contrast, with a Perl script-based pipeline, one may lose track of a script, a parameter, or the order of execution, thereby breaking the pipeline.

With RDF one can merge in data from an external data source, and write simple rules for describing the joint interpretation. For instance, in the case of splice variation, suppose there was one dataset identifying splice variants, another describing which tissues each variant was expressed in, and a third describing what tissue some disease is manifested in. With a handful of lines of code, one can derive some data saying which splice variants occur in that tissue, and thus might be involved in the disease. If some genes related to the disease have already been identified, one could move right on to looking at the protein features that characterize those variants. That's a huge reduction in one's search space!

Here's an illustration of what the above this description might look like in N3 format:

```
## Define gene and splice variants

:Nox4 a affx:Gene ;
        affx:strand "+" ;
        affx:chr "chr7" ;
        affx:hasTxVariant [affx:representedBy :gi14582296] ;
        affx:hasTxVariant [affx:representedBy :gi18204713] .

## Add new facts about a disease and tissue specifity

:HepatitisB a bg:Disease ;
        bg:manifestedIn :Liver .

 bg:associatedWithDisease :HepatitisB .

:Liver a affx:Tissue .

{ affx:hasTxVariant ?v . ?v affx:representedBy  .}
=>{?v affx:foundIn :Liver } .

## Rule to link transcript to disease through tissue specificity

{ ?gene affx:hasTxVariant [affx:representedBy ?tx ; affx:foundIn ?tissue] .
  ?gene bg:associatedWithDisease ?disease .
  ?disease bg:manifestedIn ?tissue . }
=>
{ ?tx bg:expressedIn ?tissue  .
 ?tx bg:conveys ?disease . } .
```

Drawbacks of using RDF for data encoding

One disadvantage is that RDF is verbose (a typical complaint of XML formats). However, for this sort of data, it's more important to be specific than concise. With the tools available currently, there's some scalability issues. RDF isn't a tool for everything: to make best use of it currently, the data should be partitioned into modest chunks. Melissa's research RDF data was split across several files, but in retrospect should have been partitioned to a finer grain since some users have reported memory issues while working on this data.

Barriers to widespread adoption of RDF for data encoding

Generating the RDF/N3 ontology and encoding the data for the Cline et al. publictaion was quite easy. Learning to develop an RDF ontology was only marginally more difficult than generating one with XML, where no learning curve would be involved. So complexity is not a major issue.

When data is released in conjunction with a paper, there are issues regarding where the data should be stored and accessed long-term. Ideally, the data would be housed by the journal that published it, so it can live with the publication. For many commercial operations, there is no suitable place for public datasets.

A benefit of RDF-encoded data is that there's less "data bloat" relative to a tab-delimited format. Encoding the data in tab-delimited files could lead to bloat because to make the data interpretable, it would need to be de-normalized substantially. Like other XML formats, RDF is quite compressible. One of the Cline et al. RDF files compressed 8x; an N3 file compressed 5x.

Other RDF Tips

Community response has been extremely positive. A user can easily move the data out of the RDF framework as needed. Since RDF is a subset of XML, many XML readers will work with it as-is. Data in N3 format can be moved into a relational database with minimal parsing. For people staying within the RDF framework, the tools most often used are CWM (4) or Jena (5). The W3C has an RDF editor/viewer called IsaViz, which is built on Graphviz and allows export into a number of graphical formats such as SVG.

Distributed Annotation System and Semantic Web Issues

The Distributed Annotation System (DAS) is a framework for sharing and integrating distributed annotations from various sources via standardized queries and responses across the Web (6). The requirements of distributed genome annotation offers some unique challenges for the semantic web. The semantic web could also enhance DAS. This section gives a DAS-biased perspective on the semantic web.

A new version of the DAS specification, DAS 2.0, is currently being developed to address a number of shortcomings in the original 1.0 DAS spec, including:

- An updating protocol to allow the protocol to be used for annotation and editing rather than just viewing.
- Better support for hierachical structures (e.g. transcript + exons).
- Alternative formats for data payloads, e.g. AGP for representing sequence assemblies.
- An extensible namespacing system that allows non-genomic coordinates to be used for annotations. Non-genomic coordinates include gene space, protein space, and structure space.
- Registry and discovery of DAS servers.

Simplicity is important in the design of the DAS2 spec, as it was for DAS 1.0 which uses only HTTP GET, URLs, and XML. This has been a factor in the ease of adoption of DAS 1.0 by various genome annotation centers. In keeping DAS2 as simple as possible while adding the desired enhancements, the DAS2 developers are striving to ease the transition to DAS2 for DAS1 users.

Development of DAS2 is being funded by a multi-site grant (7) involving Affymetrix, Cold Spring Harbor Lab (CSHL), Ensembl, and Dalke Scientific, and commenced in the summer of 2004. A driving force of DAS evolution is community feedback (8), so the new DAS specification reflects input from the spectrum of the DAS user base. The most current version of the DAS2 spec can be obtained using CVS (9).

The Semantic Web for Life Sciences workshop presents an opportunity to express some of the rationale behind the design decisions of the DAS2 spec and to solicit community feedback. We invite more in-depth discussions of these issues on the DAS mailing list (link at http://biodas.org).

While the semantic web is beyond the current scope of DAS, new features introduced with DAS2 should put DAS in good position to integrate with such systems as the become available. The DAS2 developers encourage community experimentation with the semantic web and related technologies. These may be incorporated into the next version of DAS.

LSID within DAS

Within a DAS system there are identifiers for objects internal to DAS, such as biological sequences and features on those sequences, and identifiers for external data, such as a RefSeq record or an entry in an ontology. Data for internal DAS objects can be retrieved using protocols specified by the DAS specification. Retrieval of external data would be handled outside of DAS.

The DAS2 developers have decided that all IDs, internal and external, should be URIs, which allows either URLs or LSIDs. The primary reason for allowing URLs for external data links is so that small labs are not required to have resolvers installed in order to set up DAS2 networks. The use of URIs within DAS for objects like DAS features and sequences should help enable the integration of DAS with the semantic web.

For internal DAS data identifiers, there have been ongoing discussions about using LSIDs, URLs, or MOBY Triples (10). The quest for simplicity in DAS has lead the develoers to favor a RESTful architecture (11) in which URLs are used as unique pointers to objects as well as allowing data retrieval using HTTP GET in a format specified by HTTP content negotiation. Further, the HTTP header offers a facility for metadata encoding, and authentication via normal HTTP authentication, something which the LSID spec doesn't discuss. LSID involves novel forms of content negotiation and metadata encoding, so adding this requirement to the DAS spec is viewed as an unnecessary complexity. By using URIs, The DAS2 spec leaves the door open for LSID-enabled implementations.

One of the benefits of DAS is that it gives users ready access to the latest annotation and sequence data from a variety of sources. Genome annotation is a never-ending process, where annotations are continually updated based on new knowledge. The underlying sequences being annotated are also updated as sequencing projects improve the quality of their assemblies. This fact poses difficulties with the LSID spec's requirement that an LSID must always resolve to the same set of bytes.

A problem with the required constancy of the LSID-resolved data is that it negates the ability to refer to an entity which is the "history tracker" for an object or the "most recent version" of an object. These are things that are useful referents within a genome annotation project. It might be useful if the LSID spec designated predicates for these aspects of an LSID.

According to the LSID spec, the only thing that can change for an LSID is its metadata. In DAS/2 there may be URLs like

```
http://www.wormbase.org/das-genome/wormbase/1/feature/cTel54X.1
```

The contents of that URL may change, as when someone edits one of its properties. This is fine in the DAS world. But in an LSID based scheme, DAS would have to use an abstract name for the version history of the entity, which is always resolved to the most recent version of the entity, which is resolved for the actual description. That is complicated. Every data fetch from the server now requires three different transactions, or two, if the metadata for a given version included the metadata for the most recent version. As long as the most recent version points to itself, there's no need to go back to the version history. On top of that, DAS would need to implement a resolver.

By comparison, URLs are easy to use, easy to resolve, and can point to data that is allowed to change. DAS doesn't support metadata requests, but could do so in the future by either asking for "?format=x-application/rdf" or by putting that in the content negotiation. The LSID spec does not discuss mechanisms for modifying data on the server, while HTTP gives us POST.

If there were no requirement for constancy, then supporting LSIDs in DAS would be easy, using a mapping from URLs to LSIDs such as:

```
URN:LSID:server.name:DAS:<urlencoded-path>
```

RDF could then be used to relate the URL and LSID as aliases for the same entity.

Here's an illustration of the complications in dealing with "the most recent version of this sequence" using LSIDs. The strategy is to get the metadata of the sequence and look for some appropriate entry (don't know which one) for the abstract object representing the sequence through time. That in turn is metadata inspected to get the abstract object representing the current version of the sequence, which is then metadata inspected to get a concrete representation. Here are the steps one might follow to do this:

1. Get metadata for `URN:LSID:ebi.ac.uk:SWISS-PROT.accession:P34355:3`. Find the entry for the abstract object of that accession through time, let's call it `URN:LSID:ebi.ac.uk:SWISS-PROT.accession:P34355`.
2. Get metadata for `URN:LSID:ebi.ac.uk:SWISS-PROT.accession:P34355`. Find the entry for the abstract object for the current version, let's call it `URN:LSID:ebi.ac.uk:SWISS-PROT.accession:P34355:6`.
3. Get metadata for `URN:LSID:ebi.ac.uk:SWISS-PROT.accession:P34355:6`. Find the entry for the fasta version of the record, let's call it `URN:LSID:ebi.ac.uk:SWISS-PROT.accession:P34355:6?format=fasta`
4. Fetch the concrete data `URN:LSID:ebi.ac.uk:SWISS-PROT.accession:P34355:6?format=fasta`

A complication is that it's possible that the sequence could get updated before these steps are completed. This could be a problem when, for example, someone wants to retrieve FASTA and SwissProt formatted versions of the most recent version of the LSID from step #1. The first request gets the FASTA but before the second starts, the record gets updated and the sequence changed. Now the second request goes through and gets the SwissProt version, which has a different sequence than the FASTA version.

In other words, the client needs to do step #3 to prevent errors that might occur if the server updates during operation. (It could cache the metadata from #2 but then you're working with stale data, and end up with the same problem after the cache is invalidated.) Nothing in the LSID spec describes how to provide this information.

Genome annotation thus poses some challenges for an LSID-based semantic web. How is a semantic web system supposed to be able to make any inferences about

```
URN:LSID:ebi.ac.uk:SWISS-PROT.accession:P34355:6
```

when it need to follow a chain of two links to find that someone annotated a different version

```
URN:LSID:ebi.ac.uk:SWISS-PROT.accession:P34355:3
```

There would need to be some mechanism to indicate which, if any, annotations on LSID version X apply to LSID version Y. A locatable annotation could be propagated across sequence versions if sequence underlying the feature hasn't changed.

To ease the use of LSIDs within genome annotation, increasing the granularity where LSIDs are assigned could help. For example, rather than assign an LSID to an object that represents a sequence plus its annotations (such as a complete GenBank entry), assign an LSID to the sequence data itself and separate LSIDs for each annotation, which would exist as metadata for the sequence. Annotations can then be added or modified without necessitating a new LSID for the sequence. The sequence's metadata thus changes over time, and that is permitted by the LSID spec.

Increased granularity could apply to individual features themselves. For example, one could assign an LSID to the genetic concept of a gene, consisting of a genetic location and a ontology term for feature type=gene. All of the more specific properties of the gene, including the detailed physical mapping data to sequence coordinates, would all be part of the gene's metadata.

A problem with this approach is that there is no established way to refer to an entity that represents the data + metadata collection at some specific point in time. Providing the LSID for the data plus the LSIDs for all metadata would provide the snapshot. Perhaps this would be a role of RDF: associating the set of data + metadata LSIDs together. Alternatively, one could assign an LSID to the collection. The lack of consistency between the approaches taken by different annotators in this regard might be problematic.

LSIDs and Content Negotiation

In discussions here at Affymetrix and among DAS developers, a complaint that often comes up regarding the LSID spec concerns content negotiation. The LSID spec relies heavily on metadata facilities for this and there is some uncertainty as to how metadata is supposed to be used.

For example, let's say GenBank wanted to have a DataRetrievalService for resolving LSIDs for sequence data in either Fasta or GenBank format. They could design their system so that getData(lsid) returns the raw, unformatted sequence string while the other data formats are placed directly in the metadata. This would have the advantage of retrieving the data in a single call but would be cumbersome for large sequences or supporting lots of formats. Alternatively, GenBank could use metadata to specify the other formats that are available and then provide URLs for the data in these formats, which would then be retrievable in a separate request. This flexibility in how a data provider may choose to use metadata is cited as a weakness of the LSID spec, since there is no enforced uniformity in what different data providers will do.

Some suggestions that have been offered were to either (1) provide a getData(lsid, format) method on the LSID resolver or (2) encode the format within the LSID. The first suggestion begs the question of how the format argument would be specified and how a server would indicate what formats are available. Also, having a single LSID be resolvable to different data depending on the format argument breaks a requirement of the LSID spec that an LSID always be resolvable to the same set of bytes (see more on this below). The second suggestion, is not acceptible because it mixes the notion of identification and representation. A different ID for each different format is not desirable.

RDF and Ontologies Within DAS

A key goal for the DAS2 spec is to improve the classification of features, which is rather limited with DAS 1.0. The plan to replace the current DAS three-part classification of method, type, and category with an ontology of sequence feature types. The Gene Ontology group is developing a Sequence Ontology (SO, 12) to specify types of sequences and sequence annotations. DAS2 will use a restricted subset of SO called SOFA (13) as a replacement for the type and category elements of DAS features. Both of these can be replaced by a single type element that refers to an SOFA term. One can then determine the relationship of that type to any other type be exploring the sequence ontology. For example, a feature may be specified as type 'LINE', and examining the ontology it is clear that this feature is also of type 'retrotransposon' , because LINE is itself a type of retrotransposon.

Terms in the Sequence Ontology can be referenced within DAS2 XML using URIs published by an RDF version of SO that is under development. DAS2 will likely use SO to specify typing for sequences as well. The exact format of the ontology URIs is undecided. Using LSIDs for ontology term references is also a possibility. An attractive feature of LSIDs is the ability to indicate version of a term. Ontologies are very much still evolving, so it would be valuable to flag a term with the version of the ontology in which it resides.

Using a standard sequence ontology does not preclude extending the ontology. One of the advantages of using RDF and OWL to specify ontologies is that there is a formal mechanism for anyone to extend an existing ontology, which gives rise to distributed ontologies. This is an important feature as ontologies are evolving and as DAS is being used by genome annotators with highly diverse backgrounds. The technical details not yet specified regarding how one would extend SO/SOFA.

In addition to improving feature classification, there have been a number of proposals to add a general mechanism to DAS to attach arbitrary attributes to sequences, features, and quantitative data and also to allow for describing arbitrary relationships between these things. Both of these could be achieved by using an RDF XML syntax that references the URIs of the features. Using RDF, attributes and relationships would essentially be the same thing, since a relationship between two features is just a special case of an attribute that is shared by both features. This would also allow both attributes and relationships to be specified by completely arbitrary text, or by a formal ontology if desired. SO, GO, MGED (which defines terms for describing experimental conditions) and other ontologies may be suitable for this use.

An issue with using RDF by DAS is that arbitrary RDF cannot be embedded within a DAS XML document (e.g.). One approach would be to reference an external document containing the RDF from a DAS XML document. One solution is to use a "link" tag within DAS2 XML to point at other relevant data sources:

```
<link href="blah.html" type="text/html" name="Human readable text" />
<link href="some-url" type="x-application/xml+rdf" name="for the semantic web"/>
```

The DAS developers have experimented with enabling DAS2 servers to supply RDF documents directly, rather than supplying DAS XML, by using "?format=RDF" to a server request. Alternatively, this could be done through content negotiation, whereby a client requests "x-application/xml+rdf" in its headers.

We also intend to explore methods to easily query over DAS2 servers to retrieve annotations based on their attributes and relationships, and possibly additions to the DAS2 spec so that DAS2 servers can optionally support these constrained queries. The best mechanism to support such queries is unclear. For example, general query technologies utilizing RDF may prove too complex to meet our criteria of keeping DAS2 simple. Based on further investigation, we may adopt a more specialized but simpler query protocol.

Issues for Tool Vendors & Data Producers

A worthy goal for the Semweb-Lifesci initiative is to provide guidance for vendors of "omics" tools that can help make the primary data generated by these tools be more semantic web-friendly.

The mountains of data being generated by technologies such as microarrays have lead to bottlenecks at the level of interpretation and integration. Difficulty in comprehending these large data sets severely impacts the value of the data themselves. Large data sets may be easy to generate, but doing so worthless if it cannot be easily interpreted or explored in the context of other types of data. The semantic web could most help scientists in this regard by making it easier for them to explore the relationships amonst data from gene expression, protein expression, transcript structure, protein structure, compound toxicity data, pathways, genetic polymorphisms, and other areas.

Semantic web tools can really help scientists "make sense" of all this data. But what are the properties of the tools and raw omics data that are necessary to help scientists or some automated tool do this effectively? Workshops such as this could help to start establish guidelines for tool vendors and data producers to achieve this goal.

To take the case of Affymetrix, one such guideline might be:

> Assign and publish LSIDs for probes, probe sets, and target sequences, provide LSID resolution services for them, and publish RDF-OWL ontologies describing their relationships to each other and to external data.

Some practical questions arise such as:

- What if we decide at a later date to change our LSID assignment scheme?
- What standard ontologies should we reference in describing our data?

Ethical, Legal, Social Implications of the Semantic Web

At this stage the discussion is primarily around how can the semantic web make for better science, tools, and services. However, ELSI issues will become more important to address as web technologies become more powerful and are used in a

wider range of applications, such as within a clinical or diagnostic setting. Perhaps the powers of the semantic web could be brought to bear in these areas as well.

- Privacy issues (e.g., individual genotypes)
- Proprietary vs. public annotations

References

1. Cline, M.S., R. Shigeta, R. L. Wheeler, M. A. Siani-Rose, D. Kulp, A. E. Loraine. The effects of alternative splicing on transmembrane proteins in the mouse genome. PSB Proceedings (2004). http://www-smi.stanford.edu/projects/helix/psb04/cline.pdf

2. Supplemental data for Cline et al. in RDF and N3 format: http://www.affymetrix.com/community/publications/affymetrix/tmsplice

3. N3 representation of RDF: http://www.w3.org/DesignIssues/Notation3.html http://www.w3.org/2000/10/swap/Primer

4. CWM - a python program that reads, writes, translates, merges, and performs some logic on RDF and N3 files. Fairly easy to use, and great for testing. Main limitation is handling relatively small RDF/N3 files: http://www.w3.org/2000/10/swap/doc/cwm.html

5. JENA - Java RDF API developed by HP and used by many projects. Jena 2 is equipped with JDBC based persistence and an inference engine: http://www.hpl.hp.com/semweb/.

6. Dowell, R.D., Jokerst, R.M., Day, A., Eddy, S.R. & Stein, L. The Distributed Annotation System. BMC Bioinformatics 2, 7 (2001).

7. Helt, G. A. DAS2: A Distributed Genome Annotation System. NIH Grant# 1R01HG003040-01 For the abstract of the grant, go to the CRISP query page and enter "DAS2" as a search term: http://crisp.cit.nih.gov/crisp/crisp_query.generate_screen

8. See the RFC link at http://biodas.org

9. DAS CVS repository, password=cvs: :pserver:cvs@pub.open-bio.org:/home/repository/biodata

10. Wilkinson, M.D. & Links, M. BioMOBY: an open source biological web services proposal. Brief Bioinform 3, 331-41 (2002).

11. REST: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

12. Sequence Ontology: http://song.sourceforge.net/

13. SOFA: Sequence Ontology Feature Annotation, described here: http://www.gmod.org/so.shtml