ACADEMIC, LINKED DATA

WHAT EXACTLY HAPPENED TO LSID?

2016-02-26 | STIAN SOILAND-REYES | LEAVE A COMMENT

What exactly happened to LSID? It was a technically sound approach it would seem and one whose failure we would do well to learn more from.

Authors:

- Stian Soiland-Reyes, eScience Lab, University of Manchester
  http://orcid.org/0000-0001-9842-9718
- Alan R Williams, eScience Lab, University of Manchester
  http://orcid.org/0000-0003-3156-2105

doi:10.5281/zenodo.46804

Life Science Identifiers (LSID) was an identification scheme for identifying Life Science information, e.g. describing genes, proteins, species. It was created by the bioinformatics community, and standardized in 2004 as an LSID specification through the Object Management Group. As of 2016 it is no longer used, except within the biodiversity community for identifying species.

LSID overview

The LSID specification specifies 4 aspects of LSIDs:

- *LSID Syntax* specifying a URN scheme, e.g. `URN:LSID:rcsb.org:PDB:1D4X:22`
- *LSID Resolution Service*, an API for retrieving data and metadata for a given LSID
- *LSID Resolution Discovery Service* – an API for finding LSID Resolution Services for a given LSID namespace
- *LSID Assigning Service* – an API for minting new LSIDs

The APIs were specified as Java interfaces, WSDL Web Services (SOAP and basic WSDL bindings for HTTP `GET` and FTP retrieval). The specification suggests a method for registering LSID Resolution Services through SRV records in DNS.

The aims of LSIDs in 2004 were certainly promising:

LSIDs are expressed as a URN namespace and share the following functional capabilities of URNs:

- ***Global scope**: A LSID is a name with global scope that does not imply a location. It has the same meaning everywhere.*
- ***Global uniqueness**: The same LSID will never be assigned to two different objects*
- ***Persistence**: It is intended that the lifetime of an LSID be permanent. That is, the LSID will be globally unique forever, and may be used as a reference to an object well beyond the lifetime of the object it identifies or of any naming authority involved in the assignment of its name.*
- ***Scalability**: LSIDs can be assigned to any data element that might conceivably be available on the network, for hundreds of years*
- ***Legacy Support**: The LSID naming scheme must permit the support of existing legacy naming systems, insofar as they meet the requirements specified below.*
- ***Extensibility**: Any scheme for LSIDs must permit future extensions to the scheme.*
- ***Independence**: It is solely the responsibility of a name issuing authority to determine conditions under which it will issue a name.*
- ***Resolution**: A URN will not impede resolution (translation to a URL).*

Source: Life Sciences Identifiers Specification, page 15

So what went wrong?

The short answer is that LSID did not receive enough uptake. But we think the real reason for that is more complicated.

Lack of support from main actors

Lack of uptake might partially be because the importance of global identifiers in Life Sciences, although well known at the time (and obviously causing LSID to be created), did not receive enough focus from upstream bodies like funders, institutions and even PIs, and remained a niche for the Semantic Web branch of Life Science data management.

Many large data providers in life sciences did not adapt LSIDs, probably because it meant too many changes to their architecture. Thus the exposure, knowledge and skills around LSIDs did not propagate to the masses.

Data Management Policies (which would require repositories with identifiers for deposits) were in their infancy at the time, now they are pretty much mandated both by funders and institutions.

Technically there are of course many other potential reasons why LSID failed, which would have influenced the social-political attitudes.

New URI scheme

LSIDs use a its own URN scheme `urn:lsid`, which was not supported by browsers or operating systems.

Adding support for resolving a sub-scheme of `urn:` to browsers seemed difficult as it meant handling the whole `urn:` scheme, some even used another URI scheme `lsidres:` as a workaround.

If LSIDs were linked to at all, then most common would be application-specific `http://` links which embeds the LSID somewhere in its path or parameters, e.g. http://ipni.org/urn:lsid:ipni.org:names:986604-1:1.1.2.1.1.2 which uses HTTP 303 See Other redirects to a HTML representation on http://www.ipni.org/ipni/plantNameByVersion.do?id=986604-1&version=1.1.2.1.1.2&output_format=lsid-metadata&show_history=true (and thus is a Cool URI)

`urn:lsid` was never registered with IANA as an official namespace and so remained an non-standard scheme.

Dependency on DNS records

While an LSID promised to be *location independent* (allowing multiple sources to describe the same object), and had provisioning for multiple alternative LSID resolvers to be discovered via http://lsidauthority.org/ – this never manifested, and in practice LSID was bound to a DNS domain name.

For such an LSID to be resolvable without additional configuration, this required technical changes to the DNS records. At the time, most Life Science web services were not even using DNS CNAMEs to provide service names (e.g. `repository.example.com`), and Web Service URLs like `http://underthedesk18762.institute.example.edu:8081/~phdstudent5/service2.cgi` were unfortunately very common.

Asking such service providers to modify (and maintain!) their DNS SRV records was probably asking for too much. (As a side-note, the required SRV service type was not registered with IANA either).

In practice the reference implementation LSID Resolver Service also needed to run on its own port, which required it to work through firewall changes.

The end result was that most LSIDs you could find during the 2000s were not resolvable through the LSID Resolution mechanism unless you knew by hard-coding where the LSID Resolution Service was hosted.

Difficult to resolve

While tools like BioJava included support for resolving LSIDs (downloading the data), practically LSIDs were not used for resolution programmatically, at least not through the LSID Resolution Service.

This could be because these services were difficult to find (see DNS section above), or poorly maintained (running on a separate port, often down for weeks until someone notice), or difficult to call (required SOAP libraries, which in the early 2000s suffered from many incompatibility issues).

In practice LSIDs were resolved as in the ipni.org example, with simple HTTP redirects from simple HTTP URIs, but from services which only supported "their own" LSIDs. Such HTTP redirections are simple to support in pretty much any server frameworks and client libraries.

And so this begs the question – what makes an LSID different from a plain old HTTP URI?

So the LSID design suffered with a requirements for resolution that made it trickier (or at least gave the impression of being trickier) to use and mint, but in the end only the identifier bit of LSID was used.

Lack of metadata requirements

LSID had provisioning to ask for metadata in addition to the data, having two methods `getData()` and `getMetadata()`

While keeping a clear distinction between data and metadata seems like a good idea, in practice it can be quite hard as one researcher's metadata is another researchers data. There's also the question if this is the metadata about the *identifier*, (e.g. allocated in April 2004), the metadata about the *record* (e.g. last updated in 2014), or the metadata about the *thing* (e.g. discovered in 1823).

Often the distinction between a thing and the description about the thing can be blurry, which is a well known problem for the Semantic Web (httpRange-14). Additionally many data formats include their own metadata mechanism, e.g. FASTA headers, which could be hard to keep in sync with the metadata in the LSID record.

The reference implementation for the LSID Allocation Service did not require any particular metadata, which meant that in practice you could get away with no metadata at all.

LSID metadata was provided in RDF, which at the time had poor application library support, forcing many to hand-write metadata in the awkward RDF/XML format (since replaced by JSON-LD and Turtle).

The landscape of RDF vocabularies and ontologies for describing biological information was quite rough in the 2000s, with many incompatible or confusing approaches, often require a full "*buy in*" to a particular model. Practically the only commonly used vocabularies at the time were Dublin Core Terms and FOAF – but the LSID specification did not provide any minimal metadata requirements or guidance on how to form the metadata.

Today the simplicity of DC Terms has evolved to schema.org , useful for all kinds of lightweight metadata, detailed provenance can be described with PROV, and collective efforts like the OBO Foundry create domain-specific ontologies.

Rather than distinguishing between data and metadata, today we do content-negotiation between different formats/representations (e.g. HTML, JSON, XML, JSON-LD, RDF Turtle), or embed metadata in-line of HTML using microformats like RDFa.

Non-distributed allocation

Allocating LSIDs centrally through an *LSID Allocation Service* was tricky for distributed architectures, which were popular at the time.

For instance, Taverna Workbench version 1, running on desktop computers, used LSIDs to identify every data item that was created during a workflow run. In theory LSIDs were a good fit – as such data don't have a "proper home" yet – it's still good to give them identifiers early on and carry it along in the provenance trace.

But in practice Taverna had to "call home" to mygrid.org.uk's LSID Allocation Service to get new identifiers. This meant that this LSID service became a single point of failure for any Taverna installation (unless overriden in local configuration), and any downtime would stop every workflow running. Yet these LSIDs were allocated blindly and sequentially, we *(deliberately for privacy reasons)* did not record any metadata beyond "*producer=Taverna*" and had no ability to resolve the data from the LSID server.

And so in a next version of Taverna we changed the identifier code to generate random UUIDs locally, and allocated LSIDs like:

```
urn:lsid:net.sf.taverna:DataThing:b1b5c94f-d54d-4039-901c-3ad022e5845f
```

Now what makes that LSID URI any different from an URI with the prefix urn:uuid: or http://ns.taverna.org.uk/ ?

HTTP took over – death of WSDL

LSIDs are born in the early WSDL days. WSDL and SOAP was a glorified XML-RPC message-passing protocol that just happened to run over HTTP – in theory you could even transport SOAP messages over email!

WSDL users did not easily agree on common XML schemas, and so each web service would have its own schema for its operations and results.

So an `<id>` XML field (or was that attribute?) that could be sent across multiple WSDL services seemed useful – hence there was a need for LSID.

There was no need for the LSID to be directly downloadable – as WSDL services always ran through a single HTTP endpoint and just modified which XML message requests were POSTed.

But this meant some size challenges, the LSID Resolution protocol is primarily WSDL based, which proved tricky with the `getData()` method returning base64-encoded bytes, which would make the SOAP libraries eat memory – anything over 50 MB became "big data"!

In plain HTTP we have known for a long time now to transfer largish files, while LSID had to resolve to a separate byte-range-variant of `getData()` which, if at all supported, complicated download for clients.

In 2000, Roy Fielding published his PhD thesis Architectural Styles and the Design of Network-based Software Architectures – summarised by its mantra "*hypermedia as the engine of application state*", and effectively establishing the Representational State Transfer (REST) as the software architectural style of the Web.

REST services are now ubiquitous on the web, and with JSON taking over as a much simpler data format than XML, REST now power not just most bioinformatics web services but also today's modern mobile apps and web applications, like Facebook, Twitter and GMail.

So with REST the HTTP Resources and their URIs become first citizens again (rather than WSDLs lonely endpoints). HTTP URIs are resolvable directly to both human (HTML) and computational representations (JSON, XML, RDF) thanks to content-negotiation.

So I can use http://purl.uniprot.org/uniprot/P99999 to refer to a protein, and even if you have no special code to resolve this, you can just paste the link into your browser and read about *Cytochrome c* protein. But if you retrieve that Uniprot URI as Linked Data, then you can programmatically access the data, or retrieve just the FASTA sequence. Even as a uniprot.org assigned identifier, the URI works well with third-party APIs, e.g. with with Open PHACTS to retrieve the related pathways.

As a plain old URI, http://purl.uniprot.org/uniprot/P99999 can be added to web pages, publications and other repositories without any further explanation.

Centralization

Since last decade we have moved back towards centralised architectures. While our architectures consists of distributed REST services (e.g. ElasticSearch and Apache CouchDB) and *vertically scalable* platforms (e.g. Apache Hadoop, Docker microservices), but now it is running on centralised *cloud services* owned by a handful of companies like Amazon AWS, DigitalOcean and Microsoft Azure.

Large data integration efforts like Uniprot and ChEMBL have also effectively centralised ID allocations in bioinformatics. Yet after the advent of next-gen sequencing and robotic synthesis and analysis we are also producing more data than ever before.

Difficult to integrate

This is just speculation, but given the state of Web Services support in server frameworks at the time of

LSID it would be very hard for actors like EBI and Uniprot to modify their existing web-based services to support the LSID protocol.

Thus they would need to set up a separate LSID Resolution Service, but that didn't know anything about their existing ID schemes which understandably they didn't want to change over night.

Done again today (20/20 hindsight) as a plain HTTP redirection based service, LSID could easily be implemented even in modern frameworks like node.js or Ruby on Rails without needing any special libraries.

Conclusion

LSIDs were doing all the "right things" according to its time: defining a location-independent URN scheme, using DNS SRV entries, providing WSDL services, separating data and metadata, using RDF, providing discovery mechanisms and alternative resolution services.

Yet it can be argued that LSIDs, in many ways just like SOAP, was a complicated way to replicate something that could already do the job – the Web and plain-old http:// URIs.

Perhaps we needed to go the long way around to figure it all out.

IDENTIFIER LSID