1 Purpose

Data identifiers are intended to function as unique identifiers for persistent “molecules” of data, defined as information contained in one instance of a data object, derived from a data store managed by a grid node, with a UML model registered in the caDSR. Identifiers are not meant to refer to the individual fields of a data object.

We do not intend that data identifiers solve the problem of a grid-wide mechanism for providing identifiers for specific physical or conceptual entities (patients, samples, genes, etc.) that may have multiple data representations across services in the grid. We believe that it is appropriate to ask the individual workspaces themselves to determine the best way to cross-identify a given conceptual or physical entity. In the case of conceptual entities, such as genes, the VCDE workspace might recommend an appropriate controlled vocabulary approach to the representation of gene identity. On the other hand, the CTMS and ICR workspaces might handle grid-wide patient identification as part of the SOP for deidentifying patient information.

caBIG identifiers must be:

1) **Globally unique:** each identifier will refer to one and only one data molecule on the grid.

2) **Locally defined:** for simplicity and low barrier to use, it must be possible for data providers to assign their own data identifiers

3) **Permanent:** a given data identifier, once assigned to a data object, can never be reassigned to another object, though it may fail to resolve

4) **Resolvable:** given an identifier, it must be possible to find the service (or services) that provides the data referred to by the identifier and associated metadata.

5) **Usable outside of caBIG?:** if desired, LSIDs would accommodate

6) Others?

2 Format

This section is still somewhat up in the air. People seem to be comfortable with a URN format for the id. We may just use the LSID format if we decide to adopt the other parts of their specification as well. It would probably be unadvisable to use the LSID format without adopting the entire specification.

LSIDs take the following form:

urn:lsid:<Authority>:<Namespace>:<Object >[:<Revision >]

where the square brackets indicate that the Revision is optional. This is a special form of the URN format, which is:

urn:<Namespace Identifier>:<Namespace Specific String>

So, in an LSID, “lsid” is the URN Namespace Identiifer (NID), and the rest is part of the Namespace Specific String (NSS). NSS syntax is less restricted than NID syntax; the NSS is unrestricted in length and allows, in addition to alphanumeric characters, any of “(“, “)”, “+”, “,”, “-“, “.”, “:”, “=”, “@”, “;”, “$”, “_”, “!”, “*”, and “'”. Other characters must be hex-encoded using the familiar URI syntax, e.g. “%2F” for “/”.

URNs are meant to be location-independent; that is, unlike URLs, they do not specify a location, so they may be available from multiple locations on the network. Location independence does not mean that they cannot be resolved in a URN namespace-specific manner.

Other suggestions: George Komatsoulis suggested that the version identifier could be a timestamp, or perhaps he meant for the timestamp to be included In the metadata, so that the version would not have to be included in the identifier itself, for services that did not want to support retrieval by version, but wanted to allow clients to determine whether the data had been updated. I’m probably misquoting him.

3 Data provider requirements

Contrast with LSIID spec: in all identifier SIG discussions so far, we have spoken as if a request for data corresponding to a particular identifier will return XML. The LSID spec says that a request for data returns bytes. Also, in the LSID spec, one issues separate requests for data and metadata, whereas so far it has seemed to me that we were expecting to return both in one XML document.

3.1 When to provide identifiers

Relationships to other data objects should be expressed using the identifiers of the related objects.

The XML response returned for a particular identifier should include the identifier for the object requested. This allows clients to determine the latest revision identifier for an object.

Analysis service providers are not required to provide data identifiers for ephemeral results that are not persisted, even thought these computed values will have models registered with the caDSR. The logic behind this argument is that the analytical service provider has no means (short of onerous logging requirements) to identify identical output results. However, references to persistent objects within the computed results must be expressed using identifiers. As an example, a BLAST result would not require an identifier, but each of the sequence records within the result would be required to contain their identifiers.

3.2 Storing identifiers

While we recognize these issues should ultimately be left up to the data provider, we recommend that database primary keys and grid data identifiers not depend on one another. The requirements of maintaining an underlying database and the requirements of providing proper grid ids are different and may conflict. Allowing the two to vary independently avoids tendencies to satisfy one set of requirements at the expense of the other. We recommend that grid data services store grid identifiers together with their data objects, perhaps as an additional column in the object's primary database table, alongside the primary key. Alternatively, the service could generate the identifier on the fly, if there were some valid way of doing so that did not invalidate the data identifier requirements (e.g., mapping an accession number, which shares the permanency requirement).

Because identifiers specify a coarse-grained data object, each table row in a relational database will not generally require its own identifier. Specifically, rows aggregated within a larger object will not need their own identifiers.

3.3 Providing schema identifier

Any data molecule returned should include not only its own grid identifier but also the identifier of its schema, registered in the caDSR. This identifier could also be provided separately in the metadata, as would be required by the LSID specification.

3.4 Identifier management services

Clarifying synonymy, get the latest identifier (including revision) given an identifier not specific to a revision, others? What was really meant here?

3.5 Identifier calculus

Providing ids for projections, joins of data molecules, what else was meant here?

4 Resolvability

caBIG identifiers are resolvable. That is, given an identifier, one can find the service or services that provide the associated data for retrieval.

My [Moses Hohman] original recommendation is below, but it needs some revision in light of recent discussions about my original misconceptions about LSIDs.

4.1 LDAP service directory

In order for an id to be resolvable, some part of the id must enable discovery of the originating (or currently providing) data service. LSIDs provide the domain name of the originating authority, but because the grid id is permanent, this complicates resolution of the service if the service is moved to another host. Instead of a DNS domain name, the caBIG id provides an LDAP distinguished name for the service, which allows the current service host to be located via a (potentially distributed) LDAP service directory. Because this naming scheme is decoupled from standard host domain names, we can allow services to be moved from host to host, and as long as the directory is kept up to date, ids will continue to be resolvable. Because resolvability extends to the individual services themselves, the services offered by a given host could be later split among several hosts, or, alternatively, one could group many services from disparate hosts onto the same server.

The service name portion of the data identifier (see 2) takes the form natural to an LDAP entry reference, the LDAP distinguished name. The LDAP entry referred to by this distinguished name contains attributes that allow a grid client to find the service that provides the data corresponding to the given identifier. At a minimum, these attributes should contain the URL of the service's WSDL document; it will also be convenient to provide the URL of the service endpoint itself, so that the WSDL does not have to be parsed if the client already knows the service's call signature. Providing the URL is simpler than providing the domain, port and path separately, and providing the domain of the host rather than the IP address enables service providers to use DNS round-robin load balancing if desired.

An LDAP service directory provides many of the benefits to the grid that DNS/BIND provides to the Internet. For example, the Directory Information Tree (DIT) can be broken into partitions (in the DNS these are called zones) that can be managed by separate servers and authorities, providing for distributed hosting and governance. A master LDAP server can maintain several replicas, or "shadows", so that the directory service can be load-balanced. Each LDAP server can be configured to cache lookups performed on external servers, reducing the bandwidth necessary to support the lookup load. Also, in LDAP one can enter aliases for other entries, so if a service needs to be renamed (i.e., not just moved to a new host), old identifiers that used the previous service name will still resolve.

Importantly, this could be very simple in the beginning (one centralized server housing the directory) and evolve to become more detailed, redundant, etc., as needed.

4.2 Directory Information Tree (DIT) structure

Use standard components of distinguished names: ou=services,dc=cabig,dc=org for the base (dc = domain component, ou = organizational unit), and use cn (commonname) for the Relative Distinguished Name (RDN) for the service entry (e.g. cn=Genbank, ou=data, ou=services, cn=cabig, cn=org).

Use case: finding services with a given property

Keep tree as flat as possible (reduces need for changing distinguished names). Categorize services using attributes on the caBIG service objectclass; indexing the tree by these attributes will make service searches fast.

I’ll leave out other recommendations related to LDAP here, since it seems like overkill at the moment.

5 Query tool requirements

[Blank for now]

6 Implications for rest of caBIG architecture

6.1 Provenance

Suggestion: Provenance information should include not just the service name of a service (this assumes the LDAP approach to naming services), but also the actual service endpoint URL, since the service name can remain static while the hosting server changes. This would allow a client to track down problems to the actual service provider, when multiple providers are available for a given service name.

6.2 Vocabulary services

CaBIG vocabulary services may be deployed using LDAP (this seemed to be the preference at the Chicago face-to-face), integrating nicely with an LDAP service directory.

7 References

[Blank for now]