Fig. 1

Fig. 2

1. News on RegTAP

(vgl. Fig. 1)

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

(vgl. Fig. 2)

- A brief history
- Utypes reform
- Open questions

2. The Story so Far

2006-2009: Development of Registry Interfaces 1:

1. Publishing/searchable registries
2. OAI-PMH interface for both with some extra rules
3. SOAP/ADQL1-based interface for searchable registries
4. Registries in the Registry

2008: ADQL 2.0; 2010: TAP 1.0 – so, we now have fairly good IVOA standards to cover (3)

2011: Paul Harrison started to work on a relational model for the Registry

2012: Proposal for RI2 in Urbana, agreement to postpone RI reform and do RegTAP separately in Sao Paulo

3. RegTAP development

RegTAP WDs have been out for almost a year now, rougly 20 major changes in 50 SVN revisions since then.

Implementations:

- two DaCHS/Postgres-based instances harvesting OAI-PMH in Heidelberg and Potsdam,
- a VODance/MySQL-based implementation in Trieste harvesting another relational registry
- ESAVO has an ingestor for the RegTAP schema, but the database has no public TAP interface yet
- STScI reported on good RegTAP progress at the last interop.

4. Stable contents

The following features have been stable for a long while:

- a schema with 13 tables,
- rules to fill them from VOResource,
- a data model id to locate the services,
- three UDFs to match ADQL with registry needs,
- and use cases and their respective queries

So: this is the perfect moment to start using RegTAP and to port over your clients.

(we already have a usable browser-based client at http://dc.g-vo.org/WIRR)

5. The Next WD

Changes since the interop include:

- The final utypes reform: I'm no longer pretending to do utypes, I'm officially doing xpaths.
- Seven columns renamed to avoid name clashes with res_role.base_role
- Now ignoring "formal" interfaces from StandardsRegExt, interfaces must always be in capabilities again
- Now boldly deprecating multiple access URLs per interface and claiming they're going to go away in VOResource
- Quite a bit of redaction

6. Uneasy Relationships I

Background: Ray is seriously considering having the following as a standard model:

Service record ivoid_1:

- capabilities with interfaces
- shallow metadata

Data collection record ivoid_2:

- rich metadata on the data itself

Declared relationships:

- ivoid_1 service-for ivoid_2
- ivoid_2 served-by ivoid_1.

There are some good reasons why one could want this; in particular, it's much more elegant when there's a bunch of services all serving the same data (think, e.g., 2MASS).

With TAP in general and ObsTAP in particular, but also federated typed services ("lens images from various campaigns"), the principle of plucking apart data and service metadata is inevitable.

If it's going to the norm, though, usual queries should still be simple. And here the uneasiness starts.

7. Uneasy Relationships II

In RegTAP, this requires joins through the relationship table; e.g., get the ivoids of services for things talking about masers:

```
SELECT related_id
FROM rr.resource
  NATURAL JOIN rr.relationship
WHERE
  1=ivo_hasword(res_description, 'maser')
  AND relationship_type='service-for'
```

What are the accessURLs? The way of RegTAP suggests

- add `access_url` in the select clause
- add `NATURAL JOIN rr.interface` in the from clause

Dead wrong here.

8. Uneasy Relationships III

The right way to get the accessURLs of services for data on masers in a world of split data and services would along the lines of:

```
SELECT access_url
FROM rr.resource
  NATURAL JOIN rr.relationship as r
  JOIN rr.interface AS i ON (i.ivoid=r.related_id)
WHERE
  1=ivo_hasword(res_description, 'maser')
  AND relationship_type='service-for'
```

This may still look manageable – but now combine this with capabilities within "old-style" records. And UNION is out, since ADQL doesn't have it. Yikes.

Now, quick, figure out the join conditions for

- "...and having a column with flux around 1 GHz"
- "...and letting me query by flux around 1 GHz"

The upshot is: As soon as you routinely join through the relationship table, the rr schema is split into two parts – those that will usually be joined with resource (res_date, res_role, res_schema, res_subject, res_table, table_column), those that will usually be joined with capability (interface, intf_param), and those that end up messy (validation, res_detail).

Writing joins under these circumstances becomes an intellectual strain. I've even started considering having ivoid for the resource-affine tables and sivoid ("service ivoid") for the capability-affine tables (validation and res_detail would have both, which may actually be an improvement).

It'd still be ugly.

My take take: restrict the use of relationship to when it's absolutely necessary.

9. More Open Questions

- Do we want an interface to retrieve multiple RRs? (OAI-PMH is fine for single RRs)
- STC: can someone devise a smart way to marry MOCs and RDBMSes?
- Can we have a validation suite?

10. Hack Away

The registry is cool. Let's use it some.