1. Global Discovery in pyVO

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

- A proposed API
- Problems of global dataset discovery
- Registry aspects
- What Registry could fix
- Query aspects

Funded by e-inf-astro, BMBF FKZ 05A20VH5

2. The Image Discovery API

What I want to offer:

```
images, log = discover.images_globally(
        space=(132, 14, 0.1),
        time=time.Time(58794.9, format="mjd"),
        spectrum=600*u.eV,
        inclusive=False)
```

images would be a list of obscore-like metadata (or more: What about datalink-enabled cutouts?), and *log* would be information on which services yielded how much (or how they failed).

If True, *inclusive* would make the function ask services without coverage information and return SIAP1 images that have no usable information on time and spectrum.

3. Global Dataset Discovery

In the VO, global dataset discovery has two steps:

1. Locate services that *could* have relevant datasets
2. Send an appropriate query to each service discovered.

How many candidate services are there for images? Try:

```
import pyvo
print("#sia", len(pyvo.registry.search(servicetype="sia")))
print("#sia2", len(pyvo.registry.search(servicetype="sia2")))
print("#obscore", len(pyvo.registry.search(datamodel="obscore")))
```

That's:

```
#sia 267
#sia2 101
#obscore 49
```


4. A Registry Challenge

Querying 400 services all the time is not viable. And the VO will be growing further.

First reduction step: Use your constraints ("Covers M1 in X-Ray"). But:

```
select count(*) from
rr.stc_spatial
natural join rr.capability
where standard_id like 'ivo://ivoa.net/std/sia%'
```

At the moment, only 83 SIAP1/2 services declare their spatial coverage (30 for spectral, 43 for temporal; SSAP has 35 spatial coverages). Please improve this!

This is one reason for the *inclusive* parameter: This will only exclude services that have a non-overlapping coverage (and hence include ones that declare none at all).

5. The Obscore Problem

Obscore services are currently found as TAP services with the obscore data model.

Hence, their coverage is generally not useful: there can be a lot else in the TAP service.

Second request to Registry: Please change the Obscore registration pattern to how it's done for EPN-TAP (i.e., as a table with its own metadata).

6. Dupes, Dupes, Dupes

How many resources have both SIAP1 and SIAP2 interfaces?

```
select count(*) from
rr.capability as a
join rr.capability as b
using (ivoid)
where
  a.standard_id='ivo://ivoa.net/std/sia'
  and b.standard_id='ivo://ivoa.net/std/sia#query-2.0'
```

That's 24 at the moment; in the service selection, you can filter these out by preferring SIAP2. But...

7. Dupes from Obscore

At the GAVO data centre, all of its 20 SIA services are also reflected in its Obscore (and sitewide SIAP2 service, too). It would be a bad waste of resources to fire off the 20 extra requests.

Proposal (re-using auxiliary capabilities): SIAP(2) and SSAP records should include isServedBy relationships to Obscore, TAP, and sitewide SIAP2 services. Yes, it *is* a bit silly if a service record has an isServedBy relationship. But until convinced otherwise, I'll claim it's not grossly inadequate and does the job with minimal extra effort.

Third request to registry: Recommend that.

8. Running the Queries

SIAP2 and Obscore are easy: Just translate the constraints to queries and collect the rows you get back (slightly normalised).

SIAP1 is more difficult: no (generally usable) constraints on time and spectrum, so you need to filter locally if possible. Also: result rows need to be mapped to the Obscore DM.

Main problem, though: Dealing with hanging or disappearing services, timeouts, hanging reverse proxies...

9. How to log

Reminder: `images, log = images_globally(...)`

How machine-readable should *log* be?

It's an artefact of provenance at the very least: you need to know which services happened to be down with your queries.

It's also potentially a debugging aid.

But if you log everything, it'll be extremely painful to see anything in the log. Are there good models for this kind of thing?

10. OMG Testing

Writing unit tests for code of this kind is a nightmare: Do we really want to mock the Registry and lots of services?

Perhaps. But for now I'm trying to get by LearnableRequestMocker that pulls actual data and builds files from it.

Is this a good idea? Hm.

11. Status

For PoC-level code with rudiments of tests see pyVO PR #470.

It'd be great if a few other people participated in the effort; I'd already be grateful for folks interested in it from a user perspective.

12. Future Directions

Once the basics are there, here's some extensions I could see:

- Allow RoI geometries, intervals for scalars
- Enable object lists for upload (but: that only works for Obscore)
- Optionally, automatic cutouts to the RoI using SODA
- Registry work to make this faster and more reliable

...please join me!