Fig. 1

Fig. 2

Fig. 3

1. VOSI, A&A, and the real world

(cf. Fig. 1)

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

(cf. Fig. 2)

- The world according to Registry
- VOSI in that world
- Where capability fits
- Where table fits
- Where availability fits
- . . . another use case
- DALIInterface as an exit strategy

(cf. Fig. 3)

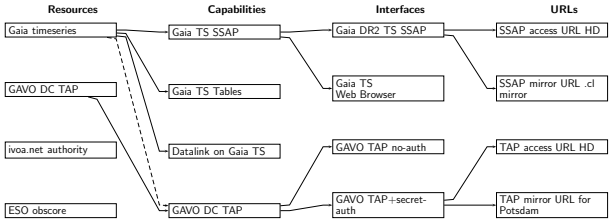

Fig. 4

2. How Registry Sees the World

(cf. Fig. 4)

The Registry has modelled the VO as a set of resources, each of which have zero or more capabilites, each of which have one or more interfaces, each of which is exposed through one or more access URLs. An additional complication is that capabilities can expose more than one resource (e.g., a TAP service with lots of different catalogs in it).

It used to be that most resources in the VO were essentially 1:1:1:1 for all these levels. That meant that if some technology chose the wrong level of modelling (e.g., it used a capability when it should have been using an interface) it didn't matter – there was a 1:1 between the two anyway.

We're now in a world where we have n:m:p:q relations between the four levels. The modelling errors that were made in the past now come back to bite us.

The main example I'm using here is Gaia timeseries; whether these are actually a resource by itself (rather than "all of Gaia DR2", say) is a difficult decision that should not concern us here.

Resources can be actual data collections or other things. Shown here is a TAP service supposedly handing out many data collections and catalogs (the GAVO TAP service) and thus acting as a facade for many other resources, and an authority, which essentially is a claim for a name (and currently doesn't have capabilities).

Resources can have zero or more capabilities. I contend they were originally thought to correspond to certain access modes to data. In the example, you can see that access is provided through SSAP and TAP. We also have two further somewhat more questionable capabilities: A cutout service (say) on top of datalink; that one's questionable because it doesn't really provide "access to data collection". Rather, it's just a way to retrieve already discovered datasets in processed forms. It's never been quite clear to me what that capability should do, but it's mentioned in the standard, and it probably doesn't hurt much, either.

The most questionable "capability" in the diagram is the "Gaia TS tables". That's one of the VOSI endpoints I'm worried about for this talk.

Also note that the GAVO DC TAP *capability* is conceptually shared between the GAVO DC TAP *resource* and the Gaia timeseries resource (in practice, it's an auxiliary capability in the second case, which is symbolised by the dashed line here). With TAP and Obscore, we have more and more capabilities delivering data from multiple other resources (though our old S*AP protocols could of course do the same thing).

Then there's interfaces. These were originally thought as representing specific versions of endpoints for protocols (or so I believe). In practice, people have added WebBrowser interfaces where (say) an SSAP service can reasonably be driven through a web browser, too. Retrospectively, I'd have much preferred if that had been done via capabilities, but it's not so damaging as to justify breaking things now.

Interfaces haven't been used to provide multiple versions simultaneously (as far as I know). They were and still are intended to keep apart different authentication schemes. Which probably is about the correct level.

Interfaces can be accessible through different URLs; these can be mirrors or redundant interfaces. Originally, all different access URLs were declared equally, but people really always just gave one access URL. However, declaring mirrors was becoming a use case, so now you *should* only give one access URL ("main service") and possibly give mirror URLs if you have mirrors.

3. Not Everything is a Capability

2011: VOSI: "An availability endpoint shall be represented by an element named capability. . ."

That's what happened, although it immediately didn't fit; what if TAP is broken but SSAP keeps working? What about mulitple access URLs?

We still limped along further in this direction.

2017: DALI 1.1 sync/async become separate capabilities, and there's examples.

Of course, at least examples again didn't fit: TAP, SSAP, and Datalink need different examples.

4. Capability's Place

/capabilities returns a list of all capabilities of a resource, so there's really a 1:1 with service.

Modelling it as a capability might still make sense: You could have multiple "sorts" of capabilities endpoints.

But: In TAP, clients parse /capabilities to figure out lots of stuff. Wouldn't it be nice if they saw a capabilities version that has the capability they're currently using at the top?

This would become a real issue if whatever we built now ended up having multiple capabilities with the same standardID; clients would then have to pick the right one out of capabilities, which certainly at this point none does. I actually doubt there is a safe way to do that right now. Let's try and avoid that: There should be a 1:1 between resource+standardID and capability.

5. Tables's Place

While the tableset in Registry records probably should be "representative", it's clear to me that /tables. . .

- . . . for a datalink endpoint should return the datalink columns (plus perhaps local extensions)
- . . . for an SSAP endpoint should return the SSA columns
- . . . for a TAP service might return different things depending on whether you authenticate or not.

So: /tables varies by interface (where auth sits).

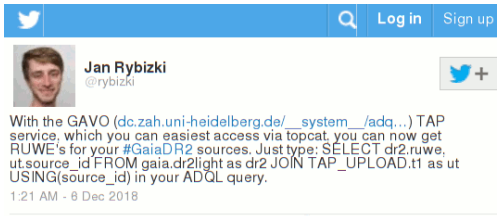

Fig. 5

6. Availability's Place

I'm not aware of anyone using availability right now, but a reasonable use case would be mirror selection in case of failures.

So: /availability varies by accessURL/mirrorURL.

7. Example's Place

Examples essentially give commented sets of service parameters. These are different depending on protocol spoken.

If we adopt the conventional stance that a capability corresponds to a standard interface, /examples thus varies by capability.

8. Another Use Case

Tangentially related, there's another reason why keeping a "single TAP access URL" is something our users want:

(cf. Fig. 5)

9. Realisation

It took me a while, too, but last September I realised:

- Using capabilities in VOSI was wrong – these things hang off various levels in the VOResource element hierarchy.
- Worse: It set us off into the wrong direction of modelling all kinds of things as capabilities.
- The one standard that got it right is TAP.
- We should not break the only standard actually using a complex interface in the VO (i.e., TAP).

10. More Insights

Once I had realised how we've got us deeper and deeper into a quagmire and the only reason we didn't sink like a stone was that most everyone has 1:1:1:1 from resource to interface, I also saw:

- My original sin was to have abused ParamHTTP as the interface type in TAP. That's just not what it is.
- We've been far too afraid to extend VOResource to match our needs.

The reluctance to change VOResource, or course, was due to changing any schema would break all clients because policy seemed to require changing the target namespace. That is fixed with the schema versioning note. That doesn't mean we can mess around with the schemas with impunity. It *does* mean that when we need new types, it's reasonable to put them in without taking everything down.

11. Proposition

Let's properly define the "sort of interface TAP uses". DALI defines such complex interfaces; so, what about DALIInterface? The new thing over ParamHTTP would be that it would enumerate the endpoints present below accessURL; and it will properly state what's always been the case in TAP: you have to concatenate your accessURL to get to the URLs you want.

```
<capability standardID="ivo://ivoa.net/TAP">
  <interface xsi:type="vs:DALIInterface">
    <accessURL>http://example.org/tap</accessURL>
    <mirrorURL>http://example.com/tap</mirrorURL>
    <endpoint>async</endpoint>
    <endpoint>sync</endpoint>
    <endpoint>tables</endpoint>
    <endpoint>capabilities</endpoint>
    <endpoint>examples</endpoint>
  </interface>
  <interface xsi:type="vs:DALIInterface">
    <accessURL>https://example.org/tap</accessURL>
    <securityMethod standardID="ivo://bla"/>
    <endpoint>async</endpoint>...
  </interface>
</capability>
```

Adding new endpoint terms should be made easy – I'd say they should be kept in a vocabulary, with names with dashes reserved for internal used.

This may incur a little bit of denormalisation (capabilities *might* be shared between the various interfaces) – but avoiding this will put the load of forming joins on the clients, and even if we believe they can properly pull that off, burdening the clients certainly won't help takeup.

All this doesn't talk about availability; there's no way around having them at the level of accessURL and mirrorURL. That needs changes in VOResource. We should have a credible consumer of that information before we touch VOResource in that respect.

12. Alternative

We could try to save the current breakage into the world of n:m:p:q services by attaching `tag` or misusing `title` attributes to the URLs or interfaces and grouping them using these tags.

It's going to put the load of our messing up on our clients.

It's going to lead to a lot of code handling that mess.

Virtually none of this messy stuff exists right now. If it were written, it would break once we properly fix things. In particular, no interoperable software I know discovers VOSI or Datalink capabilities.

13. Proposed Guidelines

Don't break TAP as it currently works. That is, neither discovery nor operation.

Work *with* the model, not against it.

14. Exit Strategy

First: For 1:1:1:1 resources, the current mess can go on as long as we need it. *There is no flag day.*

Changes required in standards:

- VODataService: define DALIInterface
- VOSI: deprecate separate VOSI capabilities
- DALI: mention legacy ParamHTTP, mention DALIInterface
- Datalink+SIAv2: deprecate split-capability registration, move to DALIInterface
- TAPRegExt: probably phase in DALIInterface