*Fig. 1*

# 1. Availability after Caproles

Markus Demleitner
*msdemlei@ari.uni-heidelberg.de*

- Availability (a10y) is no capability
- Three possible solutions
- How do we get on?

(cf. Fig. 1)

# 2. A10y response

VOSI a10y is intended to let clients figure out whether a service might be responsive. Example response:

```
<avl:availability>
  <avl:available>true</avl:available>
  <avl:upSince>2021-10-27T09:52:35Z</avl:upSince>
  <avl:downAt>2021-10-27T12:00:00Z</avl:upSince>
  <avl:note>We will take the service out for lunch today</avl:note>
</avl:availability>
```

Exactly one endpoint returning data like this is required per service by VOSI. Only 355 services (of $\sim$ 25000 in the current VO) currently fulfill that requirement. Really: It's time we dropped that requirement if we're not prepared to actually enforce it.

My personal take is: The main thing I'd consider useful in this is the note: Knowing whether the operators are aware there's a problem and what they think about it would sometimes be useful. Whether it's useful enough to warrant the effort we and the operators would have to pour into a10y that actually works I'm not sure.

# 3. A10y is not a Capability

VOSI (2011) says to register a10y endpoints using a capability with the standardID `ivo://ivoa.net/std/VOSI#availability`.

As shown in *On the Use of Capabilities in the VO* ("Caproles"), this doesn't work: Different capabilities on the same resource may have varying a10y.

In particular, a10y with mirrors *only* makes sense if they can declare their own a10y each.

So: We'll have to fix things. I'll show three types of solutions, S1, S2, S3.



# 4. S1: Forget A10y

For all I can see, there is no *interoperable* use of a10y endpoints.

Clients just try using services and simply give up when things don't work out.

Could this be enough?

Signals from the Ops IG rather point in that direction. On the other hand, CADC internally uses a10y quite a bit. Of course, they can do that without an IVOA standard as long as that's just internal use. But we at least should first think hard if these internal uses couldn't help external users or couldn't be extended to other data centers. If that enables shared tooling, it might again be worth the standardisation effort.

# 5. S2: Per-Interface Endpoints

Moving the availability URI into vr:AccessURL and vr:MirrorURL would let clients fetch info per URI:

```
<interface role="std" xsi:type="vs:ParamHTTP">
  <accessURL use="base"
    availabilityURL="https://watcher.example.org/svc1-available"
  >http://example.org/svc1</accessURL>
  <mirrorURL
    availabilityURL="https://watcher.example.org/encrypted-svc1-available"
  >https://example.org/svc1</mirrorURL>
```

This would let us keep VOSI 1.1 responses while fixing the underlying problem.

# 6. S3: Multi-endpoint Response

We could keep per-resource a10y endpoints if the responses were allowed to give multiple endpoints:

```
<availability>
  <endpoint url="http://example.org/svc1">
    <available>True</available>
  </endpoint>
  <endpoint url="https://example.org/svc1">
    <available>False</available>
    <note>Certificate renewal in progress</note>
  </endpoint>
  ...
</availability>
```

I would expect that medium-sized data centers would have just one availability endpoint that lists all their resources' status in one convenient place.

# 7. How do we go on?

What proposed solution should we try first? (Or should we try something different still?)

Then: S1 and S3 would be VOSI work.

S2 would be VOResource (and later a bit of VOSI) work.

How wants to have a say? How do we figure out where to go? Who will do the legwork?