International
Virtual
Observatory
Alliance

On the Use of Capabilities in the VO

Version 1.0

IVOA Note 2019-03-15

Working group
Registry

This version
http://www.ivoa.net/documents/caproles/20190315

Latest version
http://www.ivoa.net/documents/caproles

Previous versions
This is the first public release

Author(s)
Demleitner, M.

Editor(s)
Demleitner, M.

Version Control
Revision 5416, 2019-04-29 15:47:04 +0200 (Mon, 29 Apr 2019)
https://volute.g-vo.org/svn/trunk/projects/registry/caproles/caproles.tex

Abstract

The data model VOResource and its extensions imply for services in the Virtual Observatory (VO) is a four-layered hierarchy of VO resources having capabilities having interfaces having access or mirror URLs. This model has to be aligned to the real (deeply graph-like) structure of services accessible through various machine- and human-readable endpoints that often have to be combined in some way by clients in order to operate a service. Since in the early days of the VO, most services simply had a 1:1:1:1 relationship all the way between services and access URLs, the details of mapping endpoints to the VOResource model did not matter too much. With the advent of complex, multi-endpoint services, authenticated interfaces, and failover endpoints, this has changed, and some of the early design decisions prove to be problematic. This note investigates some problems identified and suggests a roadmap to remedy them.

Status of this document

This is an IVOA Note expressing suggestions from and opinions of the authors. It is intended to share best practices, possible approaches, or other perspectives on interoperability with the Virtual Observatory. It should not be referenced or otherwise interpreted as a standard specification.

A list of current IVOA Recommendations and other technical documents can be found at http://www.ivoa.net/documents/.

Contents

Acknowledgments

The core of this note was conceived during a workshop on authentication and authorisation in the VO held 2019-01-29 through 2019-01-30 at the Osservatorio Astronomico di Trieste in the context of the Horizon 2020 project ASTERICS (grant agreement number 653477).

Conformance-related definitions

The words "MUST", "SHALL", "SHOULD", "MAY", "RECOMMENDED", and "OPTIONAL" (in upper or lower case) used in this document are to be interpreted as described in IETF standard RFC2119 (Bradner, 1997).

The *Virtual Observatory (VO)* is a general term for a collection of federated resources that can be used to conduct astronomical research, education, and outreach. The International Virtual Observatory Alliance (IVOA) is a global collaboration of separately funded projects to develop standards and infrastructure that enable VO applications.

1 Introduction

1.1 Terminology

Before we can state the problem to be discussed here, we need to define the meanings we will assume for some terms that have not always been consistently used by different standards and communities. Readers finding this section somewhat abstract or even hair-splitting can skip it, referring back to it as necessary. The specific interpretation of some of these terms in the VO Registry is given in 2.1.

resource The URI standard RFC 3986 (Berners-Lee and Fielding et al., 2005) defined a resource as essentially anything that can be referenced. This sense of resource we will call "URI resource" here. In contrast, VOResource (Plante and Demleitner et al., 2018) defined a resource to be something registered in the VO Registry. In this note, we are interested in the special (but overwhelmingly typical) case of registered services, which today may be fairly complex entities comprising many different endpoints. We will call this a "VO resource" in the following. While the terms are not contradictory – an entry in the VO registry can be referenced by a URI, and RFC 3986 explicitly states that a collection of other resources can again be a resource –, VO resource is a much narrower term, and certainly not all statements about VO resources are true for URI resources.

endpoint As used here, an endpoint is understood to be the smallest service element the Registry talks about, i.e., as a rule something that has a single URL immediately ready for use by clients. It is hence a URI resource belonging to a service, and very typically this resource's representation will vary by time or by parameters passed in. In VOResource, the URLs of these will appear usually in *accessURL* or *mirrorURL* elements, although certain types of endpoints have been defined elsewhere (e.g., the URL of a footprint endpoint).

service Here, a service is understood to be a special type of VO resource, specifically a VO resource registered with *capability* children. From a client perspective, it is a collection of endpoints, each of which has a defined role in relation to some subset of the other endpoints, the grouping being implied by interfaces and capabilities, all cooperating in order to provide a given functionality.

interface Without qualification, "interface" is taken to mean an interface in the VOResource sense (i.e., a particular instance of a capabilty with a well-defined access mode). In the rare cases when we mean "abstract set of modalities for service use", we speak of "client interfaces" (for machine use) or "user interfaces" (for human use).

1.2 A Brief History of the VOSI Problem

In 2011, the Virtual Observatory standard "IVOA Support Interfaces" (VOSI in the following) appeared in version 1.0 (Graham and Rixon et al., 2011). It defined three endpoints useful in VO operations. One ("VOSI capabilities" in the following) returns an enumeration of way to access a service, one ("VOSI tables" in the following) to obtain the structure of underlying or delivered tables, and one ("VOSI availability") to interrogate whether the service is

down and if so, when it is expected to come back. In section four of VOSI, it said

> An availability endpoint shall be represented by an element named *capability*, of type *[vr:]Capability* (defined by the standard VOResource XML schema [(Plante and Benson et al., 2008)]). The value of the standardID attribute of the capability shall be *ivo://ivoa.net/std/VOSI#availability*.

Similar language follows for the two other endpoints. This seemed reasonable at the time, and it set the tone on how to use the concept of capability in VOResource for a long time.

Some implementers still had what seemed like corner cases that would not quite work with these capabilities. For instance, when the data in a service was available through both Obscore and SSAP, what should *the* tables endpoint (i.e., what `accessURL` in the interface in the tables capability points to) return? The Obscore or the SSAP schema? Or perhaps some internal representation that both of these are generated from? Or should there be two tables capabilities? But if so, how would clients find out which one to use?

Later, when services started to have mirrors, it was noticed that the availability endpoint as specified was rather useless, as it could only indicate global up or global down. This will clearly not enable failover scenarios, where clients could fall back to mirrors or even other protocols if only parts or a single deployment of a service are failing.

The expectation was that such corner cases could be dealt with more throrougly later, and so what we might call the "VOSI model" (i.e., using capabilities for all kinds of aspects of service operation) was picked up by the Data Access Layer Interface specification (DALI) in its 1.0 version that became an IVOA recommendation in 2013 (Dowler and Demleitner et al., 2013). It went further in at least insinuating that synchronous and asynchronous operation of "a" service ought to be two different capabilities, too.[1]

Meanwhile, IVOA's very successful Table Access Protocol (TAP) in its version 1.0 went a different way (Dowler and Rixon et al., 2010): It just declared a common "root" access URL in a single capability (and interface) and defined a fixed hierarchy of child endpoints. For instance, clients of a TAP 1.0 service can always retrieve the service's current capabilities at `<root url>/capabilities`. While this was not in direct conflict with VOSI,

[1]Section 2 of DALI actually speaks of "resources", which there is clearly meant in a URI sense rather than a VOResource sense. DALI's Section 2.5 maps at least some of these "resources" to VOResource capabilities, which further complicates matters: the "resource" (in the DALI sense) in, say, sync, is can have multiple endpoint URLs in multiple interfaces. Thus, the are not really URI resources either, since those are required to be uniquely identifiable.

it made declaring the VOSI capabilities a rather redundant exercise, as no client would use the registry or some capability endpoint to discover the endpoints it needed for a task – why should they, when some simple string manipulation was sufficient?

By the mid-2010s, the awkward dichotomy of an orthodoxy consisting of VOSI and DALI that was ignored by the clients and a practice consisting of TAP that ignored the orthodoxy was exarcerbated when SIAP 2.0 (Dowler, Bonnarel and Tody, 2015) and Datalink (Dowler, Bonnarel, Michel and Demleitner, 2015) followed the DALI recommendation and required different capabilities for sync and async operation, as well as showing VOSI capabilities in example registry records. Since SIAP 2.0 uptake was relatively slow and datalink use usually bypasses both the Registry and the capability endpoint, the contradictions still could be ignored.

The problem really came to light as the consequence of two other developments. For one, more and more services came online that published a whole set of other resources. To cope with them, the concept of an auxiliary capability was introduced in DDC, the note on discovering data collections (Demleitner and Taylor, 2016), which changed the relationship between resources and capabilities from (practically) 1:1 to n:1. This was still relatively manageable while auxiliary capabilities were only used for TAP and other DAL protocols predating DALI. Declaring auxiliary capabilities for a standard like SIAP 2.0, where the example registry record lists a handful of different capabilities will at least be a lot more complicated than the simple examples shown in the DDC note.

Finally, in 2017, TAP 1.1 tackled the long-dormant issue of services with access restrictions. By VOResource, access restrictions are indicated by adding *securityMethod* elements to *interface* declarations. If the access URLs vary by authentication technology, this introduces a 1:n relationship at least between capability and interface. This, in turn, meant that puzzling together the endpoints required for a full DALI-described services (sync, async, VOSI capabilities, VOSI tables, possibly examples) became a non-trivial exercise. It was also found that certain ways of declaring the ensuing resource trees had a significant impact on existing registry discovery patterns (Demleitner, 2017).

This note proposes a way how to, in a nutshell, combine the simplicity of the TAP 1.0 approach with the flexibility of the VOSI-DALI approach in a world of n:m:s:p resources, that is, resources that are served by multiple capabilities (and capabilities that may serve multiple resources), with client interfaces composed of multiple endpoints and capabilties that may be exposed through multiple interfaces (e.g., because of authentication).

In the next section, we will more precisely define the problem we are trying to solve, with a brief treatment of what different conceptions of capability would mean in 2.3 and a method for obtaining well-fit service models in

2.4. Section 3 then discusses what standards are affected and what measures should be taken to correct them.

2 The Problem Setting

2.1 The Service Model in VORegistry

VOResource (Plante and Demleitner et al., 2018) defines services on four levels:

1. The resource. Conceptually, a VOResource resource is not necessarily a service. There are, for instance, authorities, which are used in identifier management, or organisations. In this note, however, we are only concerned with service-like resources, i.e., those that do have capabilities.

2. Capabilities. Since a clear definition of this term is not easy (but fundamental for adequate resource descriptions) we defer it to sect. 2.3.

3. Interfaces. Interfaces collect a set of equivalent URLs. Rather than attempt a precise definition, we enumerate the use cases for having different interfaces on a capability:

   - Different authentication methods; in VOResource, *securityMethod* is a child of *interface*.
   - Different transport layers; for instance, it was originally envisioned that VO services might have plain HTTP and SOAP interfaces. In practice, this was only tried in Registry Interfaces 1.0 (Benson and Plante et al., 2009) with its OAISOAP and OAIHTTP interface types. Still, analogous cases are conceivable, for instance in VOEvent, where multiple transport layers (VTP, XMPP) have been in use in parallel (but never registered).
   - Different versions of protocols. While VOResource 1.1 encourages differentiating protocol versions using standard identifiers on capability, individual protocols might opt to still use the version attribute on interface.

   We believe this enumeration is conclusive for now and may thus stand in for a rigorous definition of interface (that otherwise appears to be hard).

4. Access URLs. VOResource 1.0 suggested representing full mirrors (or perhaps services on machines with multiple network interfaces) by giving multiple *accessURL* elements within an *interface*. Even data providers who operated mirrors did not use this, perhaps because no

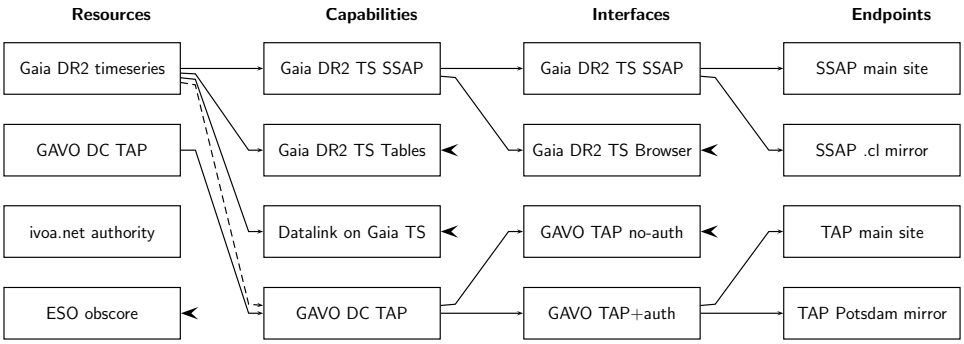

Figure 1: A sketch of a few resources and their various ancestors in VOResource's service model. To keep the figure reasonable, we have elided many branches. Elided branches are marked with an empty arrowhead.

procedure through which clients would choose an access URL was defined. VOResource 1.1 therefore discourages the practice. Instead, it prescribes a single access URL per interface for a "primary" site and has mirror endpoint URLs in `mirrorURL` rlements. These latter have `title` attributes intended to support users in making informed choices of mirrors suitable to their location or usage profile.

2.2 A Sketch of the Current Practice

Figure 1 gives a sketch of how a tiny part of the VO Registry might look like. As resources, there are time series from Gaia Data Release 2 (Gaia TS in the following), a TAP service from GAVO, an authority (as an example for a capability-less resource), and ESO's obscore service (which would have another tree of capabilties).

The Gaia TS service has an SSAP capability for querying the data collection, a VOSI tables capability for retrieving the table schema, and a Datalink capability to retrieve ancillary data and perhaps different serialisation formats of the time series.

The dashed line to the GAVO DC TAP *capability* – which is also a capability of the GAVO DC TAP *resource* – represents an auxiliary capability (Demleitner and Taylor, 2016). These come into play when many different data collections are exposed through a single service. In a sense, many "data-like" VO resources may effectively share one capability with this scheme. While this is not the place to go into more detail, for the purpose of this treatment it is important to keep in mind that in today's VO the relationship between resources and capabilities is $n : m$.

Fig. 1 goes on to show that capabilities can have multiple interfaces. The SSAP capability in the example has one "standard" interface for use by SSAP clients, and one interface claiming to be targeted at browsers (a

common practice which we would like to discourage; see below). On the TAP capability, there are two interfaces, one with and one without authentication.

At the finest level of description (right end of the figure), both the SSAP and the TAP interfaces are available through a main site and a mirror each; these correspond to the actual endpoints that clients communicate with, although in the case of TAP the URLs given in the *accessURL* element need to be modified in protocol-specific ways to obtain usable endpoint URLs.

2.3 What is a Capability?

We believe that some of the difficulties initially sketched have their roots in a lack of consensus what a "capability" of a service should be. VOResource 1.0, which has introduced the concept in the VO, attempts something like a definition in several places:

- "behavioral characteristics and limitations" of a service (in the schema annotation for capability children of service)
- "describes the service's interface as well as information regarding its behavior" (sect. 2.2.1)
- "each [capability] describing a different major functionality"; "describes the behavior of service capability and how to access it" (sect. 2.2.2)

VOResource 1.1, accepted as recommendation in 2018, does little to sharpen the meaning of the term.

In consequence, capabilities are currently used for at least three different purposes:

1. expose different ways to access roughly the same functionality (in the example above, TAP and SSAP as two modes to query the Gaia TS collection). Examples for such functionalities include "publish spectra from instrument X", "publish images of objects of class Y", or "provide access to tables published in the journals, A, B and C" – where we would admit both TAP-style querying and, perhaps, full-table download in the last case.
2. provide the building blocks in a complex, multi-faceted client interface like VOSpace (and some drafts of TAP 1.1) for assembly by clients when building a full client interface.
3. attach functionally different endpoints to one service. The clearest example for this in the above example is the datalink capability on the Gaia DR2 TS service; this has nothing to do with the "main function" of the service, which is data discovery and retrieval as offered by SSAP

and the TAP auxiliary capability. However, the datalink capability is not, as in (2), an integral part of the client interface but offers an entirely distinct user or client interface serving a different purpose.

We suggest that it would be desirable to reduce the capabilty concept to meaning (1) in order to have a more orthogonal Registry ecosystem with fewer ways to model a given technical system. One clear advantage of this scheme is that when a client inspects a capabilities element, it can assume that the alternatives offered are conecptually equivalent and it can choose the one most convenient for the task at hand.

The main reason for avoiding use (2) is that when distributing building blocks over mutiple capabilities, it becomes hard to assemble these building blocks when there are multiple interfaces or mirror URLs. It has been proposed to introduce certain uniqueness requirements (e.g., only one url per authentication method and mirror title) in order to allow deterministic endpoint assembly, but these seemed rather brittle. An alternative would be to furnish endpoint URL elements and perhaps interfaces with some sort of key that would let clients assemble parts by grouping them by this key. This still poses serious challenges for service operators and client authors that we would like to avoid unless strong reasons are found to maintain the building blocks meaning of capabilities.

As to purpose (3), our main concern is that significantly different functionalities as a rule should not have common resource-level metadata (say, titles) in the first place. Continuing the example from above, "images of observatory X" and "storage at observatory X" should hardly have the same resource description or title. One could escape this argument by claiming that capabilities generate titles of their own by prepending to the resource title; so, resources could have capabilities "Obscore at..." and "Storage at..."; a resource with the title "Observatory X" would yield our hypothetical example capbilities.

No Registry client works like this, though, and the implications such a wide-ranging emancipation of capabilities would have are profound. It seems to us very much preferable to model such situations with two resources, perhaps with bespoke relationships between them.

Still, whether definition (1) is too narrow to cover all use cases for capabilities we have of course remains to be seen.

To this end, let us investigate the consequences of the narrow definition "capabilities declare functionally equivalent, technically different access modes to a resource" for some of the endpoints that are or were proposed to be declared in separate capabilties. Without significantly limiting generality, we will generally employ the Gaia TS SSAP service mentioned above as an example. The "main function" hence would be time series discovery.

- sync/async. Current DALI represents endpoints for direct, ParamHTTP

(sync) and UWS-based (async) access to the same service in different capabilities. Our narrow definition would allow this, as, ignoring mode-dependent usage limits and the like, sync and async are requried to be functionally equivalent where both are present (but we will see in sect. 2.4 why they should still not be separate capabilities).

- VOSI capabilities. Since no time series can be discovered through a VOSI capabilities endpoint, it does not perform the "main function" of the service and hence should not be a capability itself.

- An auxiliary TAP capability containing the table underlying the SSAP service. Given the right parameters, essentially all data that can be discovered through SSAP can be located through TAP as well. This would suggest that this can stay a capability under the definition proposed here.

  However, in general many more pieces of data will be findable through the TAP service, as it probably contains more data collections. We suspect that some statement to the effect that "usual" capabilities may be specialisations rather than equivalents of auxiliary ones should be made close to where we specialise to meaning (1).

- A web page enabling data discovery using a plain web browser. Again, time series that can be found through the SSAP service would generally be available through such a form, too, so the browser-oriented endpoint could reside in a capability of its own. We would not consider additional functionality of a web form-based service (e.g., some visualisation, perhaps extra discovery modes) a major change in the client interface's "main function"; to us, this would be not very much different to using different clients to operating the SSAP service.

- A passive description of the data ("reference URL"). While the page at the reference URL will in general point users to where the time series can be found, it will typically not itself offer facilities for locating or downloading data. Hence, it is not functionally equivalent to the SSAP service and should not reside in a capability of its own (which indeed it does not in current VOResource; there is a separate element for reference URLs in VOResource).

- A datalink endpoint (Dowler, Bonnarel, Michel and Demleitner, 2015). Datalink offers no data discovery at all. So, while an SSAP service might *use* a Datalink service, SSAP and Datalink capabilities should not be siblings. Whether an independent registration of the Datalink endpoint in a separate service is useful remains to be seen.

- A footprint endpoint. This would be an endpoint that returns some representation of the spatial coverage of a service, as in VODataSer-

vice's *coverage/footprint* element. This does not let users do any discovery or retrieval or actual data and is hence not functionally equivalent to the SSAP service. It should hence not be modelled through a capability.

- A HiPS (**?**) of the spectral collection. This would, again, offer search and retrieval on the full set of the time series. Modelling it as a capability would therefore be in line with meaning (1).

Nothing in these considerations appears to be a major arguments for keeping meanings (1) and (2). Still, several practices that are mentioned – partly as normative behaviour – in VO standards would be frowned upon when limiting capabilties to meaning (1). Sect. 3 assesses what changes and adjustments in the VO ecosystem would become necessary.

2.4 A Structuralist Approach to the Registry

We further suggest that there should be a clear method to decide at which level in the hierarchy between VO resource and access URL a description of some network endpoint should reside. Indeed, we claim that simple substituion testing as in structuralist morphology provides such a method, as in:

> To determine where a network endpoint's description should reside in the hierarchy VO resource (top) – capability – interface – endpoint URL (bottom), starting from capability going down determine at which level one can substitute one instance with another with no changes to the endpoint's result. The endpoint should then be described one hierarchy level higher. If it changes at every level, it must be described at the access URL level.

This is a fairly abstract formulation for a rather straightforward procedure. Let us try it on some example endpoints.

2.4.1 VOSI capabilities

The VOSI capabilities endpoint returns a sequence of all capabilities that are in a VO resource's Registry record.

Consider the Gaia TS example above. To determine where VOSI capabilities should sit, first substitute the different capabilities for each other and see if the desired result changes. For instance, the VOSI capabilities document for the tables endpoint and the one for the SSAP endpoint will be the same: A document enumerating all capabilities of the Gaia TS resource. The situation for the auxiliary TAP capability is somewhat special, as what it references is a capability of a different service (the GAVO DC TAP service);

a client interested in alternative ways to access Gaia TS data will still want to see the SSAP capability, and thus even for that, it is desirable to return the same answer as on the other capabilities.

Hence, the right place to model a capabilities endpoint is at the VO resource level. This could be, as in the current practice, through a capability element. Considering the discussion in section 2.3, one might, however, prefer to use a dedicated method to express "get this VO resource's capabilites here".

2.4.2 VOSI tables

VOSI 1.1 says that a "service which uses tables in its interface should define a VOSI endpoint from which table metadata can be read". This leaves ample room of interpretation, in particular considering the discussion above on how this applies to TAP and SIAP.

Some clarification can be effected by considering possible use cases. For TAP, that is obviously enabling clients to discover what tables and columns are available for queries. For a SIAP service, however, this certainly does not apply. On the other hand, knowing about extra columns returned might be useful. Also, when VODataService is extended to show the domains values in the various result columns can take, a tables endpoint associated with a SIAP endpoint could greatly help in the construction of the SIAP query. Accepting this, a SIAP service's tables endpoint would return the columns present in its SIAP response, and hence the tables response would vary by capability.

What about interface? Going through the different use cases for interfaces given in sect. 2.1 yields:

- It is conceivable that authenticated users see extra columns, and so different authentication methods (None vs. something) might change a VOSI tables response.
- Different transport layers should not change the structure of the tables queried or returned.
- Response schemas can be expected vary with protocol version and hence by interface.

Finally, since responses from the different endpoints or an interface must be identical, so must their table schemas.

Hence, VOSI tables endpoints should be associated with interfaces.

2.4.3 VOSI availability

Capability: In the Gaia TS example, the auxiliary TAP might be down while SSAP still works (e.g., trouble in the ADQL translation layer), and

the tables endpoint might keep working when the database is offline. Hence, availability responses vary with capability.

Interface: Authenticated endpoints might be inoperative because of failures in the authentication infrastructure while non-authenticated endpoints still work. Hence, availability responses vary with interface.

Endpoint URL: A mirror might be down while the main site still works (or vice versa). Hence, availability responses vary with endpoint URL.

This means that availability endpoints need to be associated with the endpoint URLs, concretely, with the *accessURL* and *mirrorURL* elements in VOResource.

2.4.4 DALI sync and async

DALI forsees that services in general offer synchronous and asynchronous (i.e., UWS) endpoints. They can be treated in parallel, as the considerations are identical in both cases.

Capability: A given search and the response to it will be very much different for a TAP capability than on an SSAP capability. Hence, sync/async vary by capability.

Interface: Both protocol version and authentication can influence service responses. Hence, sync/async vary by interface.

Endpoint URL: Since all access URLs for a given interface are expected to have identical behaviour, a given query should yield the same result on all endpoint URLs of an interface. Note that as regards async, this statement is only true as regards job creation. Once a job is created, its URI is fixed, and clients cannot replace an access URL with a mirror URL. This is, however, irrelevant for the present discussion centered on service discovery: A client that has a job URI obviously already is beyond discovery and has successfully operated the service.

Hence, both sync and async should be described on the interface level. An alternative viewpoint on this finding is that when they are modelled at the capability level as forseen by DALI 1.1, some means of grouping together sync and async endpoints belonging together is necessary, leading to extra modelling and interpretation effort.

2.4.5 DALI examples

Capability: For a machine client, DALI examples returns sets of parameters that operate a service using a specific protocol. Names and values of these parameters depend on this specific protocol. Hence, DALI examples responses vary by capability.

Interface: Names and values of the parameters should be invariant against different transport layers. It is, however, conceivable that authenti-

cated interfaces should give more examples (e.g., for features only available to authenticated clients). Likewise, the availability of new features might make more (or different) examples desirable. Hence, DALI examples responses vary by interface.

Access URL: Since all endpoint URLs for a given interface are expected to have identical behaviour, a given example should have the same effect regardless of which access URL a client chooses.

This means that DALI examples should be associated with interfaces.

3 A Roadmap for Adopting these Findings

Since VOSI 1.0 pioneered the use of capabilities for infrastructure functionalities in 2011, several standards have taken up the pattern. Fortunately, few implementations have actually made use of techniques resulting from the pattern (which is perhaps not very surprising given the above discussion). This means that while a substantial number of standards are affected when one more carefully locates endpoints in the VOResource hierarchy, the effects on software that is actually deployed should be minimal.

This section lists the affected standards roughly in the sequence we believe work on them should commence.

3.1 VODataService

As discussed in the introduction, we consider the pattern set in TAP 1.0 – in effect, a single interface collecting VOSI, DALI and service endpoints – as what future service standards requiring complex interfaces should adopt. However, it is clear that the complex interface defined by TAP 1.0 is not a *vs:ParamHTTP* interface, contrary to what TAPRegExt 1.0 used in the examples provided.

Hence, a new interface type is called for, and the natural location for this type is VODataService. We propose to add to its schema an interface type *vs:DALIInterface*. In addition to the members of the *vr:Interface*, it should contain one or more *endpoint* elements of type *xs:token*. These tokens would give the names of endpoints below the declared access URLs, which, for TAP, would include sync, async, tables, capabilities, and possibly examples. For easy extensibility, these names should be managed in a vocabulary; the descriptions of the vocabulary terms would then point to the standards defining their functionality, which initially is exclusively DALI.

We propose to reserve names with dashes in them, in the sense that such names will never become part of the IVOA vocabulary. This lets data center operators adopt experimental extensions (e.g., `cadc-manage` or `gavo-getplan`) with minimal risk of collision. In order to avoid a situation

comparable to the inflation of private CSS properties (which used a similar scheme for a while), authors of interoperable clients would be severely discouraged from using such extensions. It is hoped that such a policy will provide sufficient incentive to make the creators of such extensions work towards a standardisation of their extensions.

Under this scheme, a TAP service might declare its interface as follows:

```
<interface xsi:type="vs:DALIInterface">
  <accessURL>http://example.edu/tap</accessURL>
  <endpoint>sync</endpoint>
  <endpoint>async</endpoint>
  <endpoint>tables</endpoint>
  <endpoint>examples</endpoint>
  <endpoint>capabilities</endpoint>
</interface>
```

VODataService is under review at the time of writing. Provided sufficient community interest, VODataService 1.2 could become recommendation by late 2019.

3.2 RegTAP

RegTAP 1.0 recommends constraining the `interface.intf_type` column to `vs:paramhttp` when discovering VO standard services; this had the function to weed out web browser-targeted interfaces that some data providers put into standard capabilities, while some of the standard interfaces were missing `role="std"`.

As of this writing, this is no longer a problem. On the other hand, we may need to be more flexible in declaring new interface types in the future. It is therefore desirable not to abuse the interface type to, essentially, say "this interface is the one that actually implements the standard." Instead, RegTAP's example queries should revert to the *`role`* attribute that was (essentially) created for this purpose.

RegTAP is under review at the time of writing. The changes can be introduced quickly and can probably become recommendation by late 2019.

3.3 TAPRegExt

TAPRegExt 1.0 implied that TAP services should use *`vs:ParamHTTP`*-typed elements to declare their interfaces, which is inconsistent with what the declared endpoints actually do, since TAP leaves undefined the behaviour when dereferencing the access URL given and instead requires to append endpoint names; even with the appended endpoint names, the resulting endpoints are not always *`vs:ParamHTTP`* interfaces (e.g., async).

However, current clients use these *vs:ParamHTTP* interfaces to discover TAP services. Hence, while TAPRegExt 1.1 should prescribe the use of *vs:DALIInterface*, it will probably have to require the parallel declaration of *vs:ParamHTTP* interfaces (but without *role* set to `std`). This will give full and unique listings of services for both clients matching against the interface type (legacy) and the interface role (new).

TAPRegExt is under review at the time of this writing. Work on assessing the consequences of the introduction of DALIInterface should start as soon as a VODataService draft is available. Since careful evaluation of client impact is required, TAPRegExt 1.1 will probably not be ready for recommendation until early 2020.

3.4 VOSI

The standard text of VOSI needs fairly substantial changes, probably too many for a simple erratum. Here is a tentative list of changes to be applied; we reference page numbers from the PDF for version 1.1.

- p. 6, "The endpoints and interface types for the support interface shall be defined in the service's registration using one capability element for each interface. The values of the standardID attribute for these Capabilities are given in section 4." – this and the following paragraph should be replaced by language like: "The support interfaces are generally used within other VO standards. The DALI standard gives a way to associate capabilities and tables endpoints with main service endpoints. Other arrangements are possible. The association of availability interfaces to access URLs is unspecified at the time of writing."

- p. 8, "In the REST binding, the service metadata shall be a single web resource with a registered URL." – here, "with a registered URL" should be removed, as there is no scenario for separate discovery of tables endpoints. In the same paragraph, another instance of "registered URI" should be replaced with "endpoint URI".

- p. 9, "In the REST binding, the availability shall be a single web resource with a registered URL." – here, "with a registered URL" should be removed; while there is an important scenario for discovery of availability endpoints ("Where can I learn why this service is down and when it will be back?"), no such scheme is defined at this point.

- p. 10, "In the REST binding, the TableSet metadata shall be a hierarchical web resource with a registered URL." – here, "with a registered URL" should be removed, as there is no scenario for separate discovery of tables endpoints.

- p. 11f, “Registration of VOSI endpoints” – the entire section should be replaced by language like: “Previous versions of this document have recommended registering VOSI endpoints using separate capabilities. This scheme has been found inadequate as the VO grew more complex (citation to this note). This document no longer recommends defining capabilities for VOSI endpoints. Other standards (like DALI and VOResource) define how the endpoints defined here are associated with the services they support.”

- p. 14, end of the example of a SIAP service’s capability response: remove the two VOSI capabilities.

Incidentally, in today’s VO, speaking of a “REST binding” is confusing without giving a discernable benefit. We suggest to use the opportunity of the review to continue the program of removing references to the “SOAP binding” (as mentioned in the Changelog for VOSI 1.1) and simply remove all instances of “in the REST binding” and “for the REST binding” in the document. This will obviously require some additional the minor work to repair sentences damaged in the process.

3.5 VOResource

As discussed above, the capabilities endpoint can be implemented service-wide, and several use cases exist for services to obtain capabilities directly from a service rather than from the registry. For this, we could continue to employ a capability element with the VOSI standardID. However, given the considerations in sect. 2.3, we would prefer if capability elements were not (regularly) abused to declare support functionality.

Hence, we suggest to declare the location of a copy of a service’s capabilities document separately. Concretely, we propose adding a *capabilitiesURL* child to *vr:Service* in VOResource. This could be done in a small update to VOResource.

Furthermore, as argued above, VOSI availability endpoint URLs would need to be associated with *vr:accessURL* and *vr:mirrorURL*. Technically, this would probably be effected by defining a base *vr:monitoredURL* type that would have an *availabilityURL* attribute (or perhaps even child, if multiple URLs to fetch availability from were found advantageous). This base type could then also be used to define *content*/*referenceURL* or even VODataService’s *coverage*/*footprint*.

However, before actually standardising anything here, we suggest that a clear consumer for this information must come forward. That, to our knowledge, no client consumes availability data even about ten years after the endpoint’s conception might suggest that the specification effort might better be spent elsewhere. Also, we believe that a preferable architecture

might be to enable clients to reliably the status of the services themselves and without relying on out-of-band metadata. This could be effected, for instance, by communicating a sample query and result per interface. The *testQueryString* element introduced in VOResource 1.1 might be the fist step on that path.

Since capabilities with a VOSI capabilities standardId are functionally sound, reworking VOResource is more a matter of orthogonality and hence not terribly urgent as long as no new code starts relying on VOSI capabiliites endpoints. We do not believe that a review of VOResource should be started just to introduce *capabilitiesURL*.

3.6 DALI

Large parts of section 2 of DALI would need to be changed; in particular, what is called "resource" in the standard should be changed to endpoints within DALIInterfaces (except for availability).

The textual differences are too large to list here. Since the regulations in question are mostly just patterns for concrete specifications, we suggest the update should be performed while an actual standard is being updated in order to immediately have implemenation feedback.

3.7 Datalink

In Datalink 1.0, the issues discussed here concern the introductions to sections 2, 2.2, 2.3. In the remaining document, the terms "resource" and "capability" are used more or less synonymously. We would suggest to unify terminology to "endpoint" throughout (unless actual VO resources or services are meant).

In the introduction to section 2, references to VOSI-availability and VOSI-capabilities should be removed. We suggest to revisit the question whether datalink services need a representation in the Registry at all based on implementation experience.

If use cases for that are found, the introduction to section 2 could be reformulated like this:

> A minimal Datalink endpoint conforms to DALI-sync with parameters described below. In most usage scenarios, clients discover the Datalink endpoint while using other services, and hence there is no need to represent them in the Registry.
>
> Where registration of Datalink services is desired, they should contain at least one *capability* element with a *standardId* of `ivo://ivoa.net/std/Datalink#links-1.1`. This must contain at least one interface with `role="std"`. This interface must be of type *vs:DALIInterface*, where the sync endpoint responds

to this specification. The optional async endpoint accepts the same parameters; its successful result corresponds to what is specified in section 3. No more than one interface element with `role="std"` without a *`securityMethod`* child can be given.

If we go into the trouble to discuss Datalink representation in VOResource, we should also say what a client is expected to do with the information discovered in this way.

Section 2.2 should simply be removed; if we express availability as suggested in sect. 3.5, this would automatically solve the association of availability endpoints to registered datalink endpoints .

Section 2.3 should be removed. If we find a VOResource capability declaration is desirable, an example capability declaration could be included in an appendix.

3.8 SIAP2

SIAP 2.0 uses the term "capability" in a generic way ("something you can use"). While this is probably defensible, we would like to discourage such usage and suggest that perhaps something like "facility" would express the intended meaning just as well without colliding with VOResource technical terms.

The specification has only minor dependencies on VOSI capabilities, at least when letting SIAP 2.0 remain the "simple" service that its predecessor was. This means SIAP 2.0 is operated through a *`vs:ParamHTTP`* interface and no VOSI features are required by clients. Judging by the formulated used cases and actual use, this is the case. Hence, we propose a simple erratum would suffice to bring SIAP2 into compliance with the principles formulated here. The erratum content could be:

> The entire introduction to section 2 (PDF p. 11) is replaced by "SIAP 2.0 is a implemented through a single endpoint described below."
>
> In section 2.1, first paragraph, the sentence "the client will find the resource path using the VOSI-capabilities resource" is replaced by "SIAP 2.0 endpoints are located by querying a registry for endpoints in interface elements marked with `role="std"` in capabilities with a standardID of `ivo://ivoa.net/std/SIA#query-2.0`. This specification does not define the types of their interface or capability for SIAP 2.0."
>
> Section 2.2 is removed; SIAP 2.0 services are no longer required to provide a VOSI availability capability
>
> Section 2.3 is removed; SIAP 2.0 services are no longer required to provide a VOSI capabilities capability. To still provide

a sample capability, the following material is added as a non-normative Appendix A:

> To register a SIAP 2.0 service, use a VODataService CatalogService with a minimal capability definition like
>
> ```
> <capability
>   standardID="ivo://ivoa.net/std/SIA#query-2.0">
>   <interface xsi:type="vs:ParamHTTP"
>     role="std" version="2.0">
>     <accessURL>
>     http://example.com/sia2/query
>     </accessURL>
>   </interface>
> </capability>
> ```
>
> As specified above, extension types of capability and interface may be used. In particular, the SIA-Capability from SimpleDALRegExt 1.1 can be used together with SIAP 2.0 as well.

3.9 SimpleDALRegExt

SimpleDALRegExt is only indirectly affected by the considerations presented here. While rectifying some undesirable practices in the Registry, however, we should use the opportunity to discourage the related practice of having *vr:WebBrowser* interfaces within standard capabilities (which, in effect, is an abuse of interface rather than capability). This could be effected in an erratum of roughly the following content:

> In appendix A, the fragment "as well as a web browser interface on the 1.0 endpoint" is replaced by "and also has a separate capability for querying with non-VO-aware web browsers".
>
> The example in appendix A is changed as follows:
>
> (a) remove the `vr:WebBrowser`-typed interface from the SIAP version 1 capabilitiy.
>
> (b) add an extra capability like this:

```
<capability>
  <!-- This is a non-standard interface to the data
  that can be operated by a normal web browser. -->
  <interface type="vr:WebBrowser">
    <accessURL use="full">
      http://example.com/asvc/form.html
    </accessURL>
  </interface>
</capability>
```

3.10 VOSpace

The VOSpace specification (in version 2.1) mentions capabilities quite a bit, but not always very consistently. For instance, the introduction to section 3.3 defines that a "Capability is a third-party interface to a node", while it later speaks of "standard capabilities" (sect. 3.3.5). It would be desirable to streamline the terminology in a future version.

Content-wise, VOSpace is the only current VO standard that operationally uses meaning (2) of "capability" as defined in sect. 2.3, i.e., it builds a complex client interface from several VOResource capabilities, the standard identifiers of which are given on p. 36 of the PDF version of VOSpace 2.1.

We are still researching if and how clients make use of the declaration of the various endpoints and if there are relevant advantages to managing them in capabilities rather than, say, endpoints within an interface.

Hence, we have no definite answer as to how VOSpace would be impacted by the findings presented here.

A Changes from Previous Versions

No previous versions yet.

References

Benson, K., Plante, R., Auden, E., Graham, M., Greene, G., Hill, M., Linde, T., Morris, D., O'Mullane, W., Rixon, G., Stébé, A. and Andrews, K. (2009), 'IVOA Registry Interfaces Version 1.0', IVOA Recommendation 04 November 2009, arXiv:1110.0513.
http://adsabs.harvard.edu/abs/2009ivoa.spec.1104B

Berners-Lee, T., Fielding, R. and Masinter, L. (2005), 'Uniform Resource Identifier (URI): Generic syntax', RFC 3986.
http://www.ietf.org/rfc/rfc3986.txt

Bradner, S. (1997), 'Key words for use in RFCs to indicate requirement levels', RFC 2119.
http://www.ietf.org/rfc/rfc2119.txt

Demleitner, M. (2017), 'TAP service discovery', Talk given at October 2017 IVOA Interoperability Conference, Santiago de Chile.
http://wiki.ivoa.net/internal/IVOA/InterOpOct2017GWS/tapdiscovery.pdf

Demleitner, M. and Taylor, M. (2016), 'Discovering data collections within services', IVOA Note.
http://www.ivoa.net/documents/Notes/DataCollect

Dowler, P., Bonnarel, F., Michel, L. and Demleitner, M. (2015), 'IVOA DataLink Version 1.0', IVOA Recommendation 17 June 2015, arXiv:1509.06152.
http://adsabs.harvard.edu/abs/2015ivoa.spec.0617D

Dowler, P., Bonnarel, F. and Tody, D. (2015), 'IVOA Simple Image Access Version 2.0', IVOA Recommendation 23 December 2015.
http://adsabs.harvard.edu/abs/2015ivoa.spec.1223D

Dowler, P., Demleitner, M., Taylor, M. and Tody, D. (2013), 'Data Access Layer Interface Version 1.0', IVOA Recommendation 29 November 2013, arXiv:1402.4750.
http://adsabs.harvard.edu/abs/2013ivoa.spec.1129D

Dowler, P., Rixon, G. and Tody, D. (2010), 'Table Access Protocol Version 1.0', IVOA Recommendation 27 March 2010, arXiv:1110.0497.
http://adsabs.harvard.edu/abs/2010ivoa.spec.0327D

Graham, M., Rixon, G. and Grid and Web Services Working Group (2011), 'IVOA Support Interfaces Version 1.0', IVOA Recommendation 31 May 2011, arXiv:1110.5825.
http://adsabs.harvard.edu/abs/2011ivoa.spec.0531G

Plante, R., Benson, K., Graham, M., Greene, G., Harrison, P., Lemson, G., Linde, T., Rixon, G., Stébé, A. and IVOA Registry Working Group (2008), 'VOResource: an XML Encoding Schema for Resource Metadata Version 1.03', IVOA Recommendation 22 February 2008, arXiv:1110.0515.
http://adsabs.harvard.edu/abs/2008ivoa.spec.0222P

Plante, R., Demleitner, M., Benson, K., Graham, M., Greene, G., Harrison, P., Lemson, G., Linde, T. and Rixon, G. (2018), 'VOResource: an XML Encoding Schema for Resource Metadata Version 1.1', IVOA Recommendation 25 June 2018.
http://adsabs.harvard.edu/abs/2018ivoa.spec.0625P