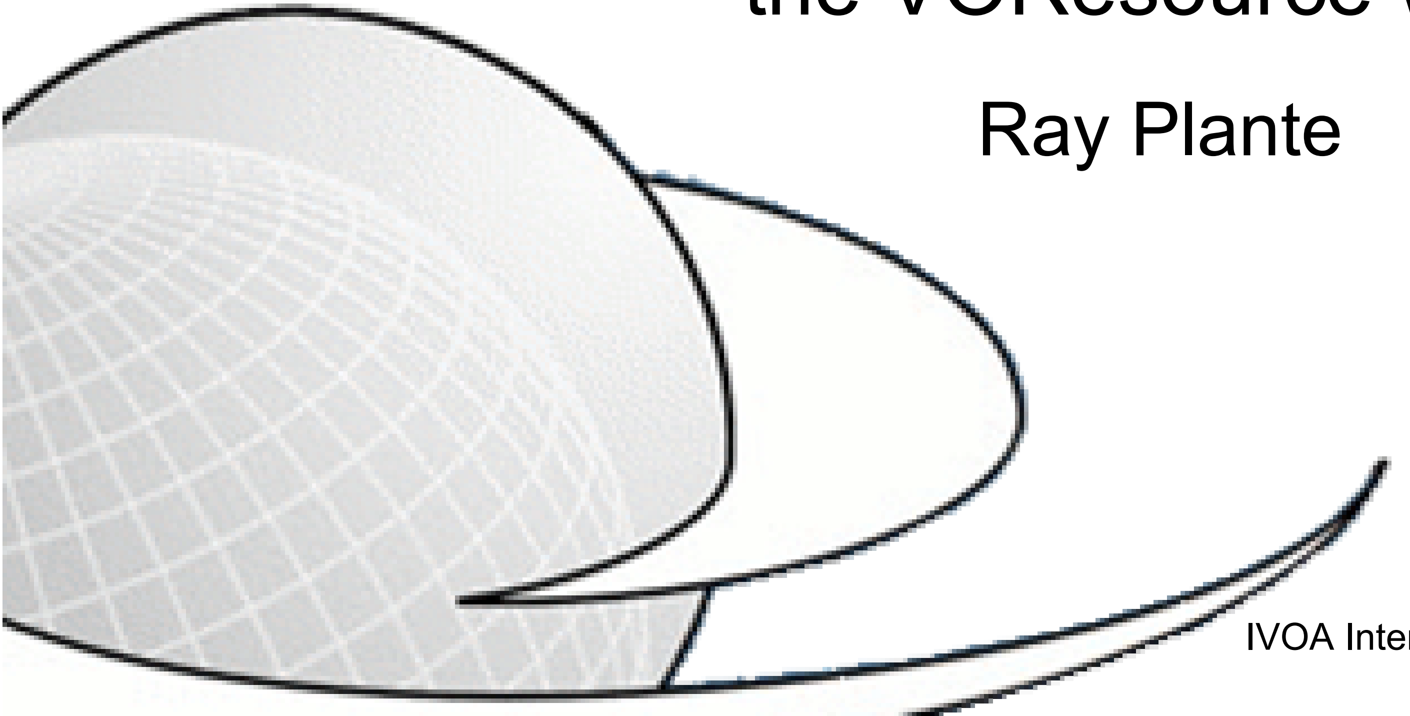
THE INTERNATIONAL VIRTUAL OBSERVATORY ALLIANCE

# Making a Service Standard Registry-Ready

(Defining Capability Metadata the VOResource way)

Ray Plante



14 May 2007
IVOA Interoperability Meeting – Beijing

# VOResource Extension Metadata

- Purpose
  - Provide a means for registry clients to discover/recognize a resource as a standard service.

    *Find me all Cone Search services*
  - Allow selection of service instances based on its instance-specific *capabilities*

    *Find me all TAP services that support table upload*
  - Provide clients with a description of the services capabilities so that it can be used effectively.

    *maxRecords,*
- Defining capability metadata should be part of the service specification
- Process
  - Defining & naming the concepts
  - Creating a *VOResource Extension Schema*

We recommend the following process

## 0. What kinds of Resources?

- Possible VOEvent resources
  - Repositories
  - Subscription/Feed services
  - …
- Types of metadata
  - Metadata about the resource that is relevant independent of any service interface
    - DataCollection, Organisation
    - (repository)
  - Capability metadata: specific to a service protocol
    - capabilities: SimpleImageAccess, ConeSearch, …
    - (feed service)
  - Resource that needs both specializations
    - Registry resource type, registry-related capabilities: Harvest, Search

# 1. Define the concepts

- Name the concepts and provide a definition
  - Try to be precise, avoid ambiguity
  - If value is numeric, specify the units!
  - Don't worry if the value is not single-valued
  - Indicate if whether a value is optional or required, if multiple values are allowed.

    [examples from SIA, including position]

# About the VOResource Schema

- A *service* can many *capabilities*
  - e.g. a “single” service can support Cone Search and TAP
    - Service: a set of interfaces into a collection of data
  - Each capability can support multiple interfaces
    - Standard interface, a web browser interface, custom interface
    - Each interface has one endpoint URL associated with it
  - How do I recognize support for the Cone Search standard?
    - **`xsi:type`**
    - **`standardID`**

```
<capability xsi:type="cs:ConeSearch"
            standardID="ivo://ivoa.net/std/ConeSearch">
```

- Service Resource types
  - Identified by the **`xsi:type`** attribute on the root Resource element

```
<ri:Resource xsi:type="vr:Service"
```

- Service: a resource that can be invoked to perform some action on the user’s behalf
  - a Resource that permits capability elements
- DataService: A service for accessing astronomical data
  - a Service that permits coverage descriptions
- CatalogService: A service that interacts with one or more specified tables having some coverage of the sky, time, and/or frequency.
  - a DataService that permits table descriptions
- DAL services to date have been considered CatalogServices

# 2. Create a sample instance

- Choose preferred Service Resource Type
  - DAL: Usually CatalogService
- Choose required Interface Type
  - ParamHTTP: HTTP GET with name=value arguments
  - WebService: a service whose interface described by a WSDL (SOAP)
- Add new capability metadata
  - One element per named concept
- Please include a test query, if appropriate
  - Allows a registry to regularly test and validate the service
  - parameters must result in a legal response, preferably not empty
- Keep it simple
  - Prefer flat structures
  - Let semantics provide grouping of data into complex elements.

# 3. Create the Schema Extension

- Use SIA, ConeSearch as examples
  - Mimic use of in-line documentation
- Derive a new type from the base Capability Type
- Often useful to create a sample instance first

3a. Import the VOResource schema

3b. Set the IVOA identifier for the standard

  - Derive an intermediate type by restriction

3c. Derive the standard Capability type by extension

  - Define elements for each capability metadatum
  - Insert semantic definition into `xs:documentation` elements
    - Style: first block is the definition, subsequent are extra notes
  - If needed define types for complex capability metadata

# 4. Describe extension in the protocol specification

4a. Indicate the preferred Resource type

“The resource element SHOULD have its **`xsi:type`** set to **`vs:CatalogService`**; otherwise, it MUST be set to **`vr:Service`** or to a type legally derived from it.”

4b. Require the new capability type

“The resource element MUST include a **`capability`** element with **`xsi:type`** set to [new type]”

4c. Require the proper interface type

“This capacity element MUST include one interface element with **`xsi:type`** set to **`vs:ParamHTTP`** [or **`vr:WebService`**].”

4d. Define each new capability element (and sub-elements), providing

- Semantic definition
- Units, restrictions on values
- If it is required or repeatable

4e. Include full schema document as appendix

- May leave out documentation to save space

- Example: Registries Interfaces, v1.0, section 4.3

# 4. Describe extension in the protocol specification

4a. Indicate the preferred Resource type *

"The resource element SHOULD have its **`xsi:type`** set to **`vs:CatalogService`**; otherwise, it MUST be set to **`vr:Service`** or to a type legally derived from it."

4b. Require the new capability type

"The resource element MUST include a **`capability`** element with **`xsi:type`** set to [new type]"

4c. Require the proper interface type *

"This capacity element MUST include one interface element with **`xsi:type`** set to **`vs:ParamHTTP`** [or **`vr:WebService`**]."

4d. Define each new capability element (and sub-elements), providing

- Semantic definition
- Units, restrictions on values
- If it is required or repeatable

*Not enforced by Schema

4e. Include full schema document as appendix

- May leave out documentation to save space

- Example: Registries Interfaces, v1.0, section 4.3

# Other Considerations

- Validation Issues
  - Requirements not enforced by the Schema
    - the preferred Resource Service sub-type
    - the required interface type
  - Full compliance check requires extra checks by custom validater
- Use `elementDefaultForm="unqualified"`
  - No namespace prefix required on elements
- Service types may be extended, too
  - To add metadata not related specifically to an interface or service capability
  - Example: `vg:Registry` extends `vr:Service` to add a listing of authorized IDs it manages