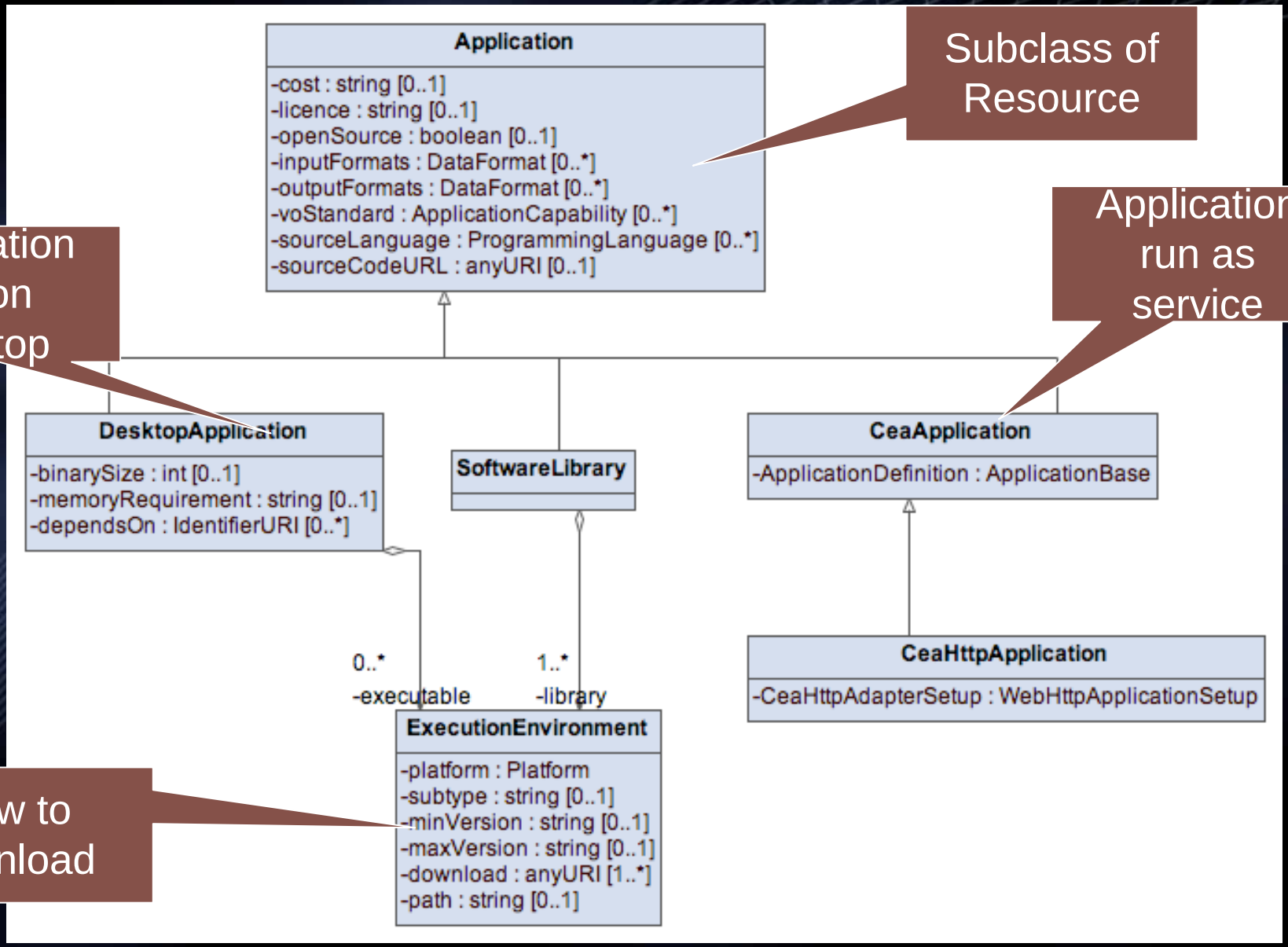
Applications and VOSpace Schema Progress

Paul Harrison ESO

Applications Schema

- Tracked progress of the Main VOResource 1.0 Schema
- Began dialog with the theory WG on their parameter model for theory code with a view to incorporating their needs in the CEA model.

Registry Application DM



Application model

- Contains just enough information to identify an application and launch automatically if possible
- Simple searches on a few properties possible
- “Find me an application that does X” style queries are best left to ontologies

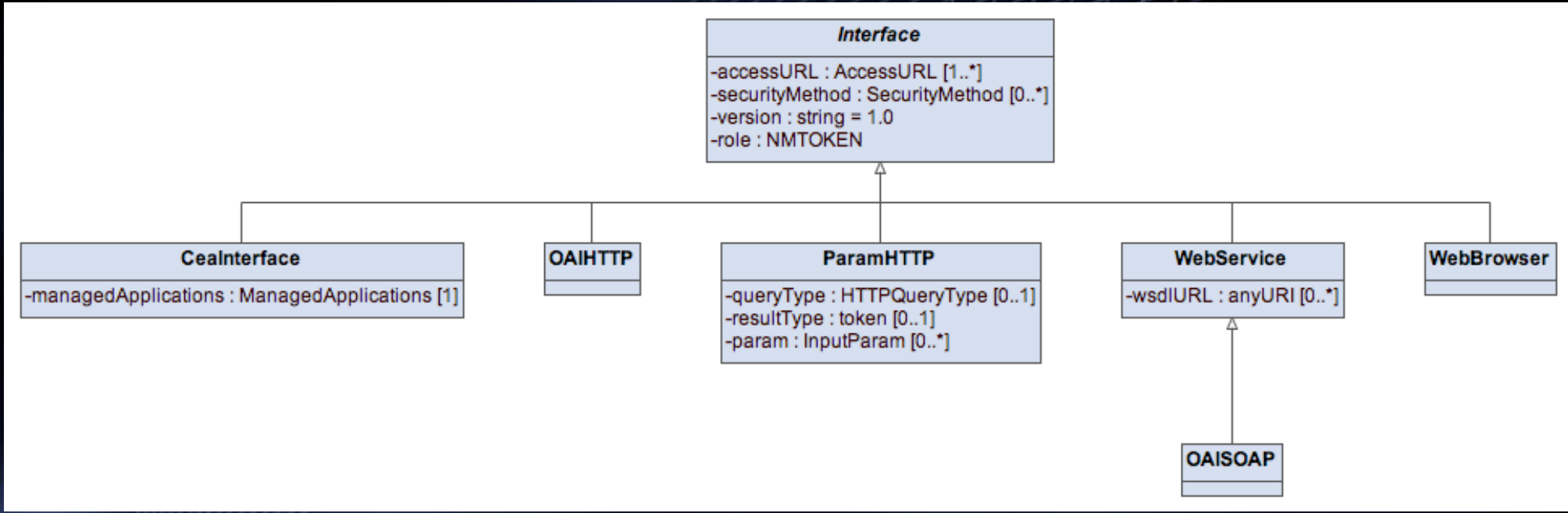
Application Model Questions

- Schema enumerations of items that will change
  - Languages
  - OS description
- OS environment - only Java has automatic run semantics - others possible?

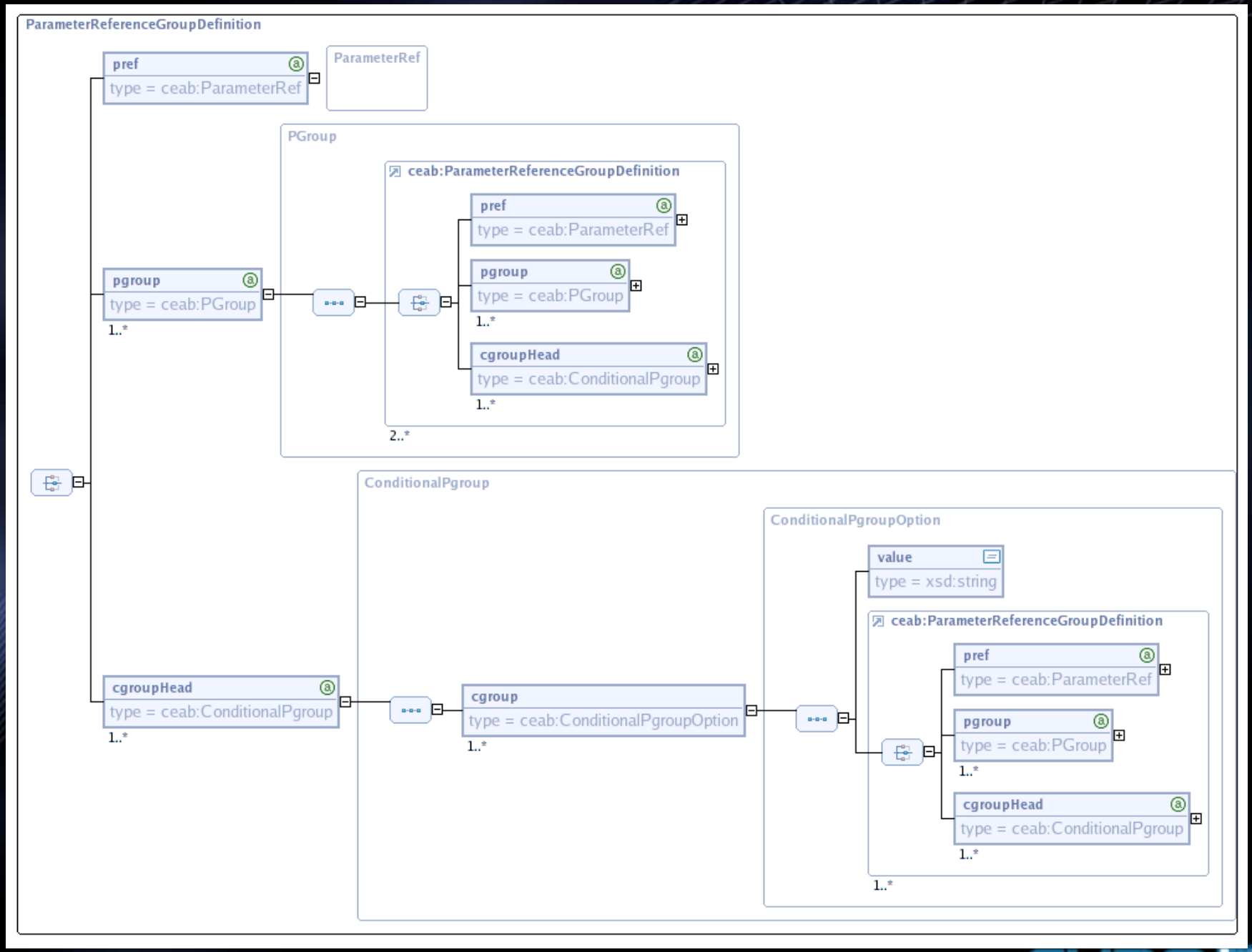
CEA Recap

- Uniform model to describe parameters for an “application”
  - Drew inspiration form WSDL, large astronomical packages (AIPS, IRAF, ADAM)
  - Divided into two parts
    - Application model - subject of this talk
    - Mechanics of calling application - evolving into UWS
- Description of parameters suitable for creating dynamic gui for user to set parameter values and subsequent invocation of application
  - See AstroGrid Workbench/AR
- CEA is not concerned with storing and managing parameter values - cf AIPS, IRAF, ADAM

CEA is an interface definition too



Parameter Grouping



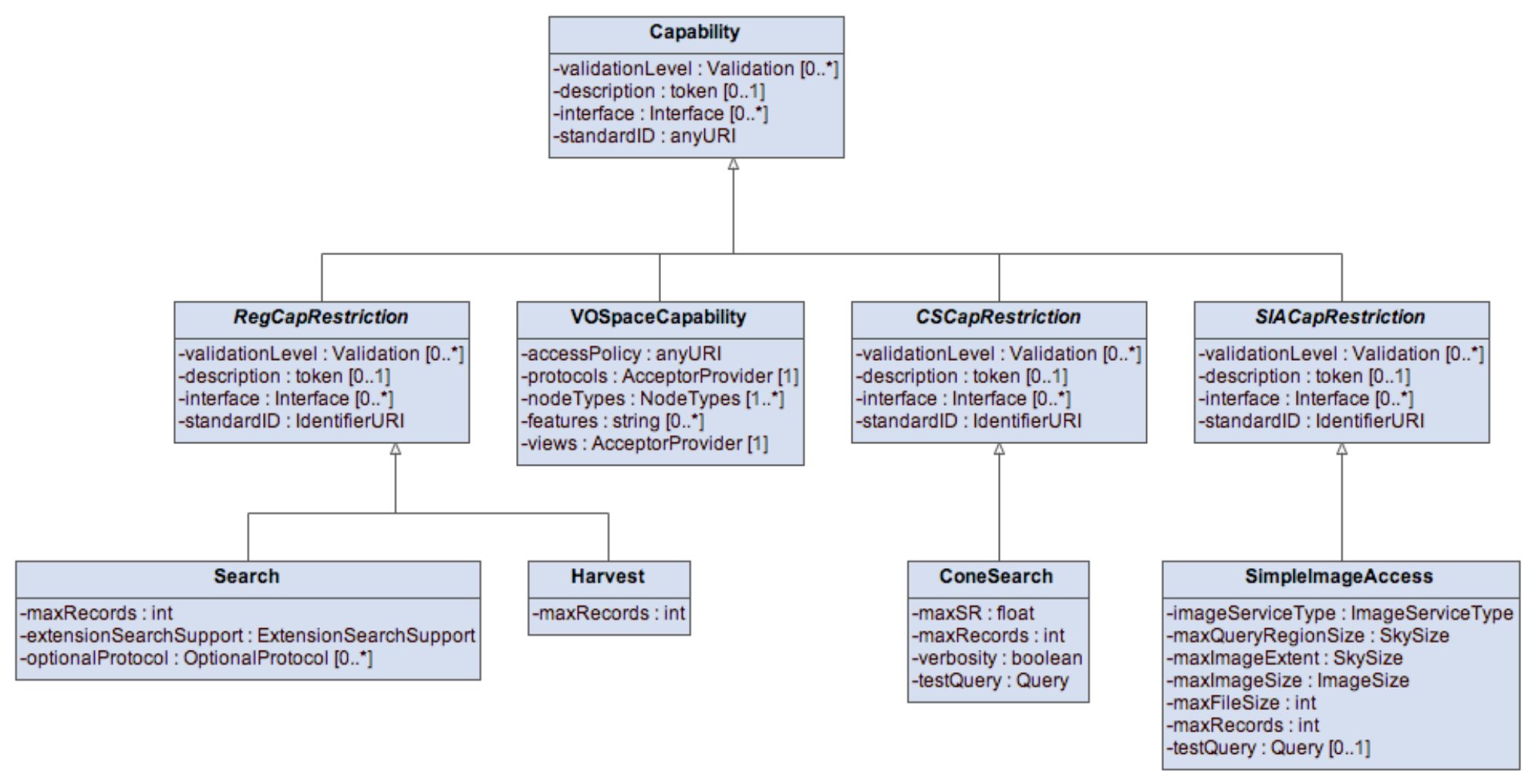
Relation to other WGs

- UWS
  - The parameter model is used to describe the call to a UWS-PA service
- Theory and SNAP - this parameter model could be used in SNAP
- CEA invocation model could be “dumbed down” to include a REST style invocation similar to SxAP.

VOSpace

- Registry entries are essential for correct functioning of VOSpace
  - Discovery of the correct VOSpace server to invoke from the vos: URL.
  - Discovery of the capabilities of VOSpace services
    - Data transports available
    - Support for optional functionality
      - containers
      - links

VOSpace Capability

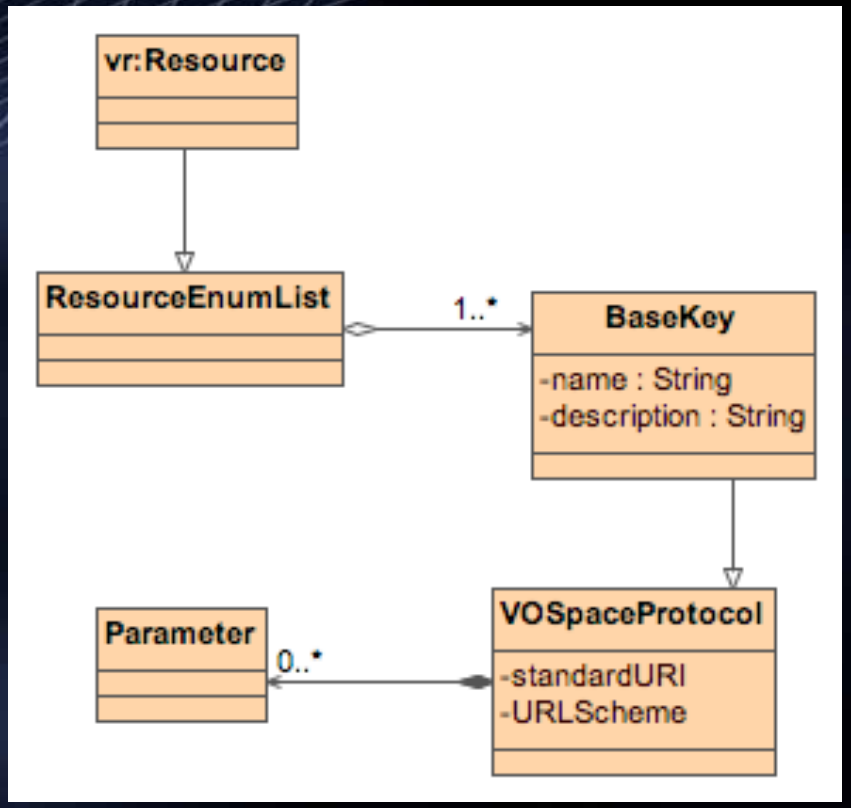


Detail - schema enumerations

- Rather than encode enumerations in the interface schema, these are encoded in the registry -
  - E.g. property names, protocol names, views etc.
- Pros
  - Easier to extend - do not have to issue new version of interface schema when a new enumeration value is required - simply edit the registry entry.
  - Easier for individual implementation to publish details of ‘non-standard’ enumeration values in a way that can be semi-automatically understood - e.g. by GUI tools to display a message to user.
- Cons
  - Allowed values are not enforced directly by the interface - up to the programmer to read registry.

Detail - schema enumerations(2)

- Aim is to produce URI
- Multiple keys per registry entry
  - Only one copy of the Dublin core
  - Standard prefix
  - Use fragment # separator to indicate the enumeration key



ivo://net.ivoa.vospace/protocols#http-get

Detail - schema enumerations(3)

```
<ri:Resource updated="2005-09-09T12:28:16" xsi:type="vsp:ResourceEnumList" created="200
  <title>VOSpace standard protocols</title>
  <shortName>VOSpace Protocol</shortName>
  <identifier>ivo://net.ivoa.vospace/protocols</identifier>
+ <curation></curation>
+ <content></content>
  <!-- now the actual protocol metadata -->
  <!-- needs to be completed -->
- <key id="http-1.1-get" xsi:type="vsp:VOSpaceProtocol">
    <description>http 1.1 get</description>
   - <standardUrl>
      http://www.w3.org/Protocols/rfc2616/rfc2616.html
    </standardUrl>
    <urlScheme>http:</urlScheme>
  </key>
- <key id="http-1.1-put" xsi:type="vsp:VOSpaceProtocol">
    <description>http 1.1 put</description>
   - <standardUrl>
      http://www.w3.org/Protocols/rfc2616/rfc2616.html
    </standardUrl>
    <urlScheme>http:</urlScheme>
  </key>
```

Conclusions

- Schema are very close to 1.0 status
- Needs accompanying Standard documentation