International

Virtual

Observatory

Alliance

Lessons Learned   Implementing UWS Version 0.1

IVOA Note, May 2008

This version:

0.1

Latest version:

not issued outside GWS-WG

Author(s):

Paul Harrison

Abstract

The Universal Worker Service pattern (UWS) [1] defines how to manage asynchronous execution of jobs on a service. The experience of implementing the REST[2] version of the pattern on the CEA[3] server component is documented here. The pattern is found to be easily applicable, and deemed suitable for adoption in other IVOA standard protocols. Areas where further clarification of the UWS pattern is necessary are identified as well as suggestions for additions and enhancements.

Status of This Document

This is an internal working draft of the GWS-WG. The first release of this document was on 2008-05-05 within the working group; it has not yet been issued outside the working group.

This is an IVOA Working Draft for review by IVOA members and other interested parties. It is a draft document and may be updated, replaced, or made obsolete by other documents at any time. It is inappropriate to use IVOA Working Drafts as reference materials or to cite them as other than "work in progress".

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

Contents

1 Introduction

The Universal Worker Service (UWS)[1] pattern defines how to build *asynchronous*, *stateful*, *job-oriented* services. It does so in a way that allows for wide-scale reuse of software and support from software toolkits. The pattern is a result of continuing research by the IVOA Grid and Web Services Working Group[9] to find such a pattern that can used by the DAL v2 services which need the asynchrony for scalability.

This document describes the experience of applying this pattern to the existing CEA[2] server implementation that Astrogrid have written.

1.1 Relationship between CEA and UWS

The semantics of the UWS pattern are a good match to the semantics of CEA (which is not surprising since the UWS pattern was inspired by CEA), and consequently adapting existing CEA services to implement the UWS pattern is a natural fit. It is relatively easy to use the same core code to manage the objects and state associated with both of the interface patterns simultaneously.

As is stated in the UWS 0.3[1] document the CEA services represent a class of services that might be classed as “Parameterised Applications”. This essentially can encompass the class of services that are not explicitly defined by the standard DAL services (SIAP[4], SSAP[5], TAP[6]). However, if the DAL v2 services adopt the UWS pattern then the CEA server can also be seen as a “DAL toolkit” that can also be used to implement these standard services as they can be configured using the same mechanisms as the “parameterized applications”. Additionally, the CEA container has the advantage that it is already compliant with other ancillary IVOA standards such as VOSI[7] and the various security related standards[8].

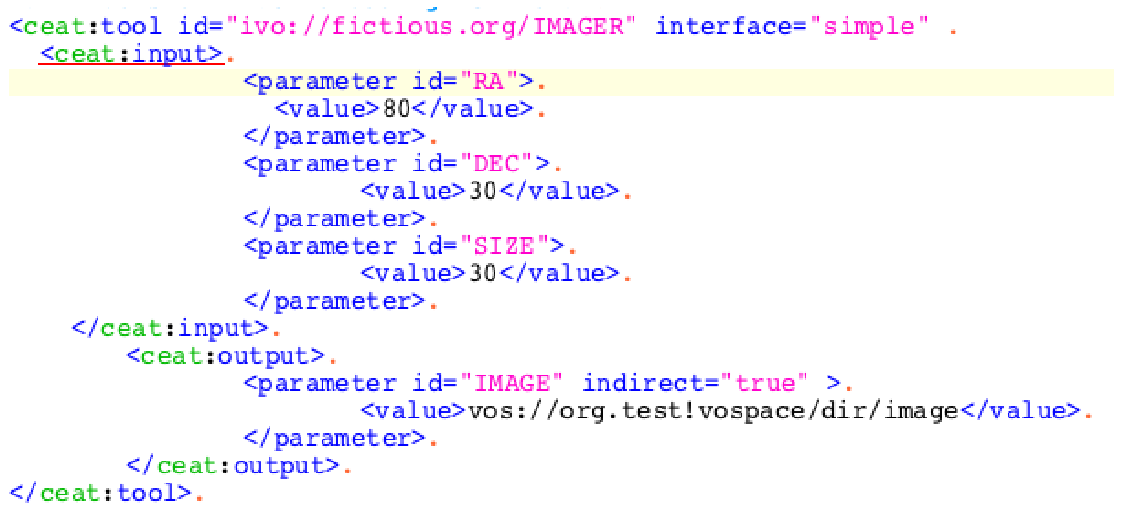
2 UWS REST Interface for CEA

2.1 Implementation Details

As has been stated above, the UWS pattern has its roots in the original CEA pattern, so the existing CEA software already has components that deal with the same concerns as UWS (such as asynchrony) making it relatively easy to “bolt on” the UWS interface to the existing software.

It should be noted that the CEA server implementation retains its existing “Common Execution Connector” SOAP web service interface, rather than implementing a SOAP version of the UWS interface. This was done because there are already a reasonably large set of clients that understand this old SOAP interface, and little apparent interest from clients to implement new SOAP interfaces.

2.2 Job Control Language



```
<ceat:tool id="ivo://fictious.org/IMAGER" interface="simple" .
  <ceat:input>.
               <parameter id="RA">.
                 <value>80</value>.
               </parameter>.
               <parameter id="DEC">.
                       <value>30</value>.
               </parameter>.
               <parameter id="SIZE">.
                       <value>30</value>.
               </parameter>.
    </ceat:input>.
       <ceat:output>.
               <parameter id="IMAGE" indirect="true" >.
                       <value>vos://org.test!vospace/dir/image</value>.
               </parameter>.
       </ceat:output>.
</ceat:tool>.
```

Illustration 1: The CEA Job Control Language

The principal feature that distinguishes different services using the UWS pattern is the Job Control Language (JCL) – i.e. the instructions that are used to create a new job as explained in section 1.4 of [1]. In the case of a CEA service this is a so-called “Tool” XML document, which (in its simplest form) has a structure as indicated in Illustration 1. The interaction of this tool document with the CEA server, as well as the method of discovering more metadata about the parameters from the CEAApplication are discussed elsewhere[10,11]

The details of this JCL are not important for the current discussion apart from noting that there is not a simple “key=value” structure to the parameters so the document needs to be POSTed in its entirety as the body of the request to the /(jobs) endpoint, rather than being able to be directly driven from a HTML form and submitted with an encoding of “application/x-www-form-urlencoded”. Although standard HTML form controls cannot be used to create a CEA JCL document, XForms[12] technology is perfectly suited to creating and manipulating such documents, and the CEA server implementation does contain a prototype XForms based page to allow local testing of the configured applications.

3 Proposed Changes to UWS 0.3.

The following sections list some suggestions for changes and clarifications of the UWS pattern as defined in [1]. These changes are prompted by the experience gained in implementing the pattern both from the server and client side.

3.1 The Quote and Job control.

In the UWS document the Quote object is used to predict when a job is likely to complete, and is used to start the job by the client indicating that it “accepts” the quote by POSTing to the Quote object. This use of the quote object is a little idiosyncratic as often the service will be unable to give an accurate prediction and is even allowed to indicate a “do not know” response.

An alternate (and possibly more natural) method that can be used to affect the job control is via sending messages to the Phase object. The Phase object does of course indicate the current state of the job and so POSTing new values to this object could be seen as a natural “RESTful” way of affecting the overall execution state of the job. In the CEA implementation of UWS the following were implemented;

1. POST phase=run is used to indicate that the job should be moved from the PENDING phase to the QUEUED phase.
2. POST phase=abort is used to indicate that a job should be aborted.

In this way it is a permissible for a client to initiate a job into a running state with

1. a POST of the JDL to the main /(jobs) URL
2. a POST of phase=run to the /(jobs)/(jobid)/phase end point

i.e. there is no need to interact with the Quote nor Termination time objects if the client is prepared to accept whatever defaults are used for these.

3.2 Extra Job State

In addition to the states that are in UWS[1] section 2.1.3 there is an additional state of “ABORTED” which would be useful. The job would be in this state when either;

- The job was aborted by the client.
- The job was aborted because it has exceeded the Termination time.

Note that this state is necessary in the case where Termination time does not equal Destruction time.

3.3 Termination Time and Job Destruction

In UWS[1] the termination time actually fulfills two roles.

- The time when the job should stop executing
- The time when all records of the job should be destroyed.

In the CEA implementation there is a distinction made between these two states in that the termination time is taken to mean only the time that the job should finish executing. An additional concept of “Destruction Time” is introduced to indicate the time at which the server will delete all record of the job having executed. This is done for two reasons.

1. If a job does exceed its termination time, then it is possible that some useful results (although presumably not the complete set of final results) could have been produced. In this case, having the destruction time later than the termination time allows the client possibly to pick up the results that are available, which might indicate to the client how to alter the job parameters in such a way to allow successful completion within a given termination time.
2. Even when a job completes successfully within its termination time then it is potentially useful to a client to be able to effectively store the intermediate results at the UWS server for a length of time that is longer than the original termination time. Indeed the UWS server might be prepared to offer storage for a longer time that it is prepared to offer CPU resources. Using the UWS server as temporary storage of results in this fashion is an effective way of managing the flow of data during a workflow without the need for a

VOSpace server, and can potentially reduce network traffic if the intermediate results of the workflow are not required at the end of the workflow.

To access and possibly modify the destruction time of the job a new URI was added to the CEA implementation

/(jobs)/(jobid)/destruction

A GET of this URI returns the ISO8601 format time when the server will delete the record of this job. POSTing a *application/x-wwww-form-urlencoded* and contains the parameter named TIME whose value is the new destruction time in ISO8601 format attempts to change the value of the destruction time. The client should not assume that its request will always be granted, however, and should always GET the URI again to see what the server has accepted. It is expected that a UWS implementation will have a policy for how long it will keep job metadata and in most cases will not allow unbounded extension of this time

3.4 Error Object

The UWS pattern presented in [1] does not have a explicit way in which to report error messages from the underlying application that is being executed by the job. In order to provide some uniformity for clients once they have detected an error status from the phase object, the CEA implementation exposes an error object at /(jobs)/(jobid)/error that contains a message containing more detail about the errors that caused the job to fail. The intention of these error messages is that they are suitable for presenting to the human initiator of the job.

3.5 Job Deletion

The CEA server narrows the case where a POST to the /(jobs)/(jobid) is used to delete a job, by only deleting the job if there is a POST parameter of the name ACTION with a value “DELETE”. This is done to make it less likely that programming errors will result in accidental job deletion.

3.6 UWS Interface type

When registering a UWS service, it will naturally be registered as a vr:Capability of either vr:Service or one of its subtypes according to VOResouce IVOA standard[17]. The vr:Capability has a number of possible vr:Interfaces that are used to register the physical endpoint of the service that provides the capability. Currently there are specialized subtypes of vr:Interface called vr:WebBrowser and vr:WebService, which are intended to indicating a general graphical interface that can be driven from a web browser and a SOAP web service respectively. It would be useful if a specific subtype were created “vr:UWS” that indicated an interface that followed the UWS pattern.

4 Composite Objects Schema

The UWS pattern document [1] does not explicitly specify any of the schema for the end points which might be viewed as returning “composite” metadata about the job i.e.

- /(jobs)/(jobid) – metadata about the job itself.
- /(jobs)/(jobid)/results – metadata about the results.

Although it is generally considered good REST style to allow the client to “drill down” by following hyperlinks to new URIs to obtain more detailed information about an entity there is obviously a balance to be struck of presenting enough useful metadata at each level, possibly repeating some that could be obtained by drilling down. This section discusses the cases where the exact form of the metadata is debatable.

4.1 Job Metadata

Although there are specialized endpoints that are designed to return specific pieces of the job metadata, e.g. termination time and phase, it seems reasonable to include these small metadata in the response to the main job URI itself /(jobs)/(jobid). A proposed schema for what is returned is described in the appendix to this document.

The CEA server actually returns an extension of this schema object to include some extra metadata that is specific to CEA.

4.2 Results Metadata

A simple schema element for the results object is presented in the schema attached in the appendix.

In the case of a pure CEA service this results metadata is actually redundant as the parameter names and the endpoints are all fully described by the CEAApplication entry within the registry. However, a general UWS client would always read the results metadata to determine how to obtain the results.

5 Good REST style

Although there is obviously a great deal of flexibility in the REST style of webservice design, and this section list some specific recommendations to help to make UWS services more interoperable.

5.1 Returning xml or html versions of objects

A UWS service should always be prepared to return a the pure xml version of the various objects as that makes it unambiguous (together with the attached XML schema) how to parse the responses to obtain the specific metadata that the client might need.

This need should also be balanced in implementations by the desire to be able to interactively drive the UWS job execution from within a standard web browser as described in section 6.In this case it is desirable to return HTML to the client, along with addition controls to allow the client to move to the next stage in the UWS pattern. Modern browsers allow xml to be transformed to html at the client using XSLT[16], so an implementation can possibly save effort by always returning an xml representation with the “`xml-stylesheet`” processing instruction pointing to xml to convert to html.

5.2 *Identifying links.*

As has been stated above following links is good REST style, and to identify links within the xml, the XLINK schema has been used.

5.3 *Redirects and Reloads*

The correct functioning of a UWS server depends upon issuing redirection and reloading commands to the client. These are done using mechanisms from within the HTTP 1.1 standard, but are repeated here for clarity as there have been historically various interpretations of these mechanisms by both HTTP clients and servers.

5.3.1 Redirect after a POST

Status 303 should returned and “Location” header set to the URL of the desired redirection address. This is exactly as specified by section 10.4.4 of RFC2616, but is in contrast to some historical uses of Status 302 for the same purpose that arose from the HTTP 1.0 specification.

This sort of redirection is used in the response to the initial POST that creates a new job in UWS.

Many browsers and servers still use the 302 status for this purpose, which strictly speaking cannot be used because the http method should not be changed for a 302 redirection.

5.3.2 Reload later

Status 503 should be returned with a “Retry-After” header set to the time after which the client should try to reload the resource. This is exactly as specified in section 10.5.4 of RFC2616.

This sort of “reload later” action can be used as the response to a request for a result before the result is yet available.

Although this is a standards compliant way of asking for a reload, it does not appear to be widely implemented in browsers, which still seem to implement the deprecated HTML `<meta http-equiv="refresh"/>` as a way of effecting a reload later.

6 Evolution of current DAL protocols

One of the attractions of the “Simple” DAL access protocols as they have been designed is that they are synchronous and can easily be driven by a standard HTML browser. Although at first reading it would seem that the UWS pattern involves a more complex interaction that would not be suitable for simple driving via a web browser, this need not necessarily be the case, a typical browser session could consist of

1. A form (or in the specific case of CEA an Xform) is presented to the user with the action endpoint equal to the /(jobs) URI on the UWS service
2. The UWS service responds by sending a redirect to the /(jobs)/(jobid), where the job is presented with a simple button to start it.

3. The user starts the job by pressing the button, which causes the UWS server to respond with redirect back to the /(jobs)/(jobid) page with suitable headers to cause automatic polling of the job page.
4. When the job indicates that it has finished the user then follows the links to the results that are displayed on the job summary page.

6.1 Miscellaneous issues

This section lists some other more speculative techiques that could be used to make interaction with UWS services even more seamless.

6.1.1 AutoRun

The UWS pattern deliberately separates the steps of creating the job and starting the job so that other aspects of job control may be set before starting the job. If a client it willing to accept the default values for these options then the interaction could be simplified by allowing the job to be automatically submitted to the run state on job creation. This could perhaps be achieved by a special parameter attached to the URL of the original job creation URL.

6.1.2 “Pseudo-Synchronous” Execution

Although one of the primary goals of the UWS patterns is to provide asynchronous execution, it would not be difficult to provide an appearance of “synchronous” operation for a client that would follow redirects and obey reload requests as discussed in section 5.3 Redirects and Reloads. As in the example above all that is needed is a way of signaling at job creation time that the pseudo-synchronous mode was desired.

7 Conclusions

The UWS pattern is a relatively “light weight” protocol to implement, whilst at the same time as bringing the benefits of asynchrony to allow more complex interactions with the calling services. It would appear to be a good candidate pattern to use for DAL v2 services to as thy evolve to include this asynchrony requirement.

There are a number of additions to UWS described in [3] that the author believes would enhance the generality and usability of the pattern.

1. Control of the execution status of the job by POSTing to the phase object – i.e. the /(jobs)/(jobid)/phase endpoint
2. The addition of an extra standard job phase of “ABORTED” to indicate when the job has been aborted.
3. An additional UWS object called destruction time that can be accessed at /(jobs)/(jobid)/destruction that determines the time at which the UWS server will destroy all records and results of a particular job. Correspondingly, the “termination time” object only refers to the time at which the job execution is terminated.

4. An extra UWS object called error that can be accessed at /(jobs)/(jobid)/error that can be used to report in detail the any error associated with the job.

Appendix 1 Schema

UWS schema for reponses

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- $Id: UWS.xsd,v 1.1.2.2 2008/05/07 12:15:00 pah Exp $ -->
<!-- proposal for basic UWS schema - Paul Harrison May 2008 -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.ivoa.net/xml/UWS/v0.9"
xmlns:uws="http://www.ivoa.net/xml/UWS/v0.9"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   elementFormDefault="qualified"
   >
   <xs:import namespace="http://www.w3.org/1999/xlink"
schemaLocation="../../../stc/STC/v1.30/XLINK.xsd"/>
<!--
  <xs:import namespace="http://www.w3.org/1999/xlink"
schemaLocation="http://www.ivoa.net/xml/Xlink/xlink.xsd"/>
-->

   <xs:complexType name="ShortJobDescription">
      <xs:sequence>
         <xs:element ref="uws:phase"></xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="uws:job-identifier-type"
use="required"></xs:attribute>
      <xs:attributeGroup ref="uws:reference"></xs:attributeGroup>
   </xs:complexType>
   <xs:attributeGroup name="reference">
    <xs:annotation>
      <xs:documentation>standard xlink references</xs:documentation>
    </xs:annotation>
    <xs:attribute ref="xlink:type" use="optional" default="simple"/>
    <xs:attribute ref="xlink:href" use="optional"/>
  </xs:attributeGroup>

  <xs:simpleType name="ExecutionPhase">
      <!-- need to think a little here about the implication of allowing a "re-
entrant" application that is capable of running mini-jobs...probably this is
indicated with a different state varible entirely -->
      <xs:annotation>
         <xs:documentation>
            Enumeration of possible phases of job execution
         </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string">
         <xs:enumeration value="PENDING">
            <xs:annotation>
               <xs:documentation>
                  The first phase a job is entered into - this is where a job is
being set up but no request to run has occurred.
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="QUEUED">
            <xs:annotation>
               <xs:documentation>
                  An job has been accepted for execution but is waiting
                  in a queue
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="EXECUTING">
            <xs:annotation>
               <xs:documentation>An job is running</xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="COMPLETED">
            <xs:annotation>
               <xs:documentation>
                  An job has completed successfully
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="ERROR">
            <xs:annotation>
               <xs:documentation>
                  Some form of error has occurred
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="UNKNOWN">
            <xs:annotation>
               <xs:documentation>
                  The job is in an unknown state
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="HELD">
            <xs:annotation>
               <xs:documentation>
                  The job is HELD pending execution and will not
                  automatically be executed (cf pending)
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="SUSPENDED">
            <xs:annotation>
               <xs:documentation>
                  The job has been suspended by the system during
                  execution
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="ABORTED">
            <xs:annotation>
               <xs:documentation>
                  The job has been aborted, either by user request or by the
server because of lack or overuse of resources.
```

```
            </xs:documentation>
         </xs:annotation>
      </xs:enumeration>
   </xs:restriction>
</xs:simpleType>

<xs:complexType name="JobSummary">
   <xs:sequence>
      <xs:element name="jobId" type="uws:job-identifier-type" />
      <xs:element ref="uws:phase" />
      <xs:element ref="uws:quote" />
      <xs:element name="startTime" type="xs:dateTime" />
      <xs:element name="endTime" type="xs:dateTime" />
      <xs:element ref="uws:termination" />
      <xs:element ref="uws:destruction"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="job-identifier-type">
   <xs:annotation>
      <xs:documentation>
         The identifier for the job
      </xs:documentation>
   </xs:annotation>
   <xs:restriction base="xs:string" />
</xs:simpleType>

<xs:element name="job" type="uws:JobSummary">
   <xs:annotation>
      <xs:documentation>
         This is the information that is returned when a GET is made
         for a single job resource - i.e. /(jobs)/(jobid)
      </xs:documentation>
   </xs:annotation></xs:element>
<xs:element name="phase" type="uws:ExecutionPhase">
   <xs:annotation>
      <xs:documentation>
         the execution phase - returned at /(jobs)/(jobid)/phase
      </xs:documentation>
   </xs:annotation></xs:element>
<xs:element name="quote" type="xs:dateTime">
   <xs:annotation>
      <xs:documentation>
         A Quote predicts when the job is likely to complete - returned at
/(jobs)/(jobid)/quote
         TODO - how to encode "don't know"
      </xs:documentation>
   </xs:annotation></xs:element>
<xs:element name="termination" type="xs:dateTime">
   <xs:annotation>
      <xs:documentation>
         The time at which the job should be aborted if it is still
         running - returned at /(jobs)/(jobid)/termination
      </xs:documentation>
   </xs:annotation></xs:element>
<xs:element name="destruction" type="xs:dateTime">
   <xs:annotation>
```

```
         <xs:documentation>
            The time at which the whole job + records + results will be
destroyed. returned at /(jobs)/(jobid)/destruction
         </xs:documentation>
      </xs:annotation>
      </xs:element>


   <xs:element name="jobList">
        <xs:annotation>
           <xs:documentation>
              The list of job references returned at /(jobs)
           </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:annotation>
               <xs:documentation>
                  ISSUE - do we want to have any sort of paging or
                  selection mechanism in case the job list gets very
                  large? Or is that an unnecessary complication...
               </xs:documentation>
            </xs:annotation>
            <xs:sequence>
            <xs:element name="jobref" type="uws:ShortJobDescription"
maxOccurs="unbounded" minOccurs="0"></xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>


   <xs:complexType name="ResultReference">
      <xs:annotation>
         <xs:documentation>
            A reference to a UWS result
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element ref="uws:phase"></xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string"></xs:attribute>
      <xs:attributeGroup ref="uws:reference"></xs:attributeGroup>
   </xs:complexType>
   <xs:element name="resultList">
        <xs:annotation>
           <xs:documentation>
              The element returned for /(jobs)/(jobid)/results
           </xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:sequence>
            <xs:element name="result" type="uws:ResultReference"
maxOccurs="unbounded" minOccurs="0"></xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Appendix 2 References

[1] G. Rixon, The Universal Worker Service
http://www.ivoa.net/internal/IVOA/IvoaGridAndWebServices/UWS-0.3.pdf

[2] P. Harrison, *Proposal for a Common Execution Architecture*,
http://www.ivoa.net/Documents/latest/CEA.html

[3] R. Fielding, Architectural Styles andthe Design of Network-based Software
Architectures

[4] D. Tody, R. Plante, *Simple Image Access Specification*,
http://www.ivoa.net/Documents/latest/SIA.html

[5] *D.Tody, M.Dolensky et al., Simple Spectral Access Protocol*,
http://www.ivoa.net/Documents/latest/SSA.html

[6] *Table Access Protocol, work in progress* http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/TableAccess

[7] *VO Support Interfaces*, work in progress http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/VOSIHome

[8] VO *Security, work in progress,* http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/SecurityHome

[9] Grid and Web Services Working Group of IVOA, work in progress,
http://www.ivoa.net/twiki/bin/view/IVOA/IvoaGridAndWebServices

[10] P. Harrison, CEA Application Model,
http://www.jb.man.ac.uk/~pah/ivoa/CEAApplicationDM.html

[11] P. Harrison, CEA Interfaces, http://www.jb.man.ac.uk/~pah/ivoa/CEAInterface.html

[12] *Xforms 1.0*, http://www.w3.org/TR/xforms/

[13] *XLink 1.0*, http://www.w3.org/TR/xlink/

[14] HTTP *1.1*, rfc2616 http://www.w3.org/Protocols/rfc2616/rfc2616.html

[15] VOTable 1.1, F.Ochsenbein et al. ,http://www.ivoa.net/Documents/latest/VOT.html

[16] XSLT 1.0, http://www.w3.org/TR/xslt

[17] VOResource: an XML Encoding Schema for Resource Metadata 1.03, R.Plante et al.
http://www.ivoa.net/Documents/latest/VOResource.html