UWS as a data-staging mechanism for SIAP

Guy Rixon

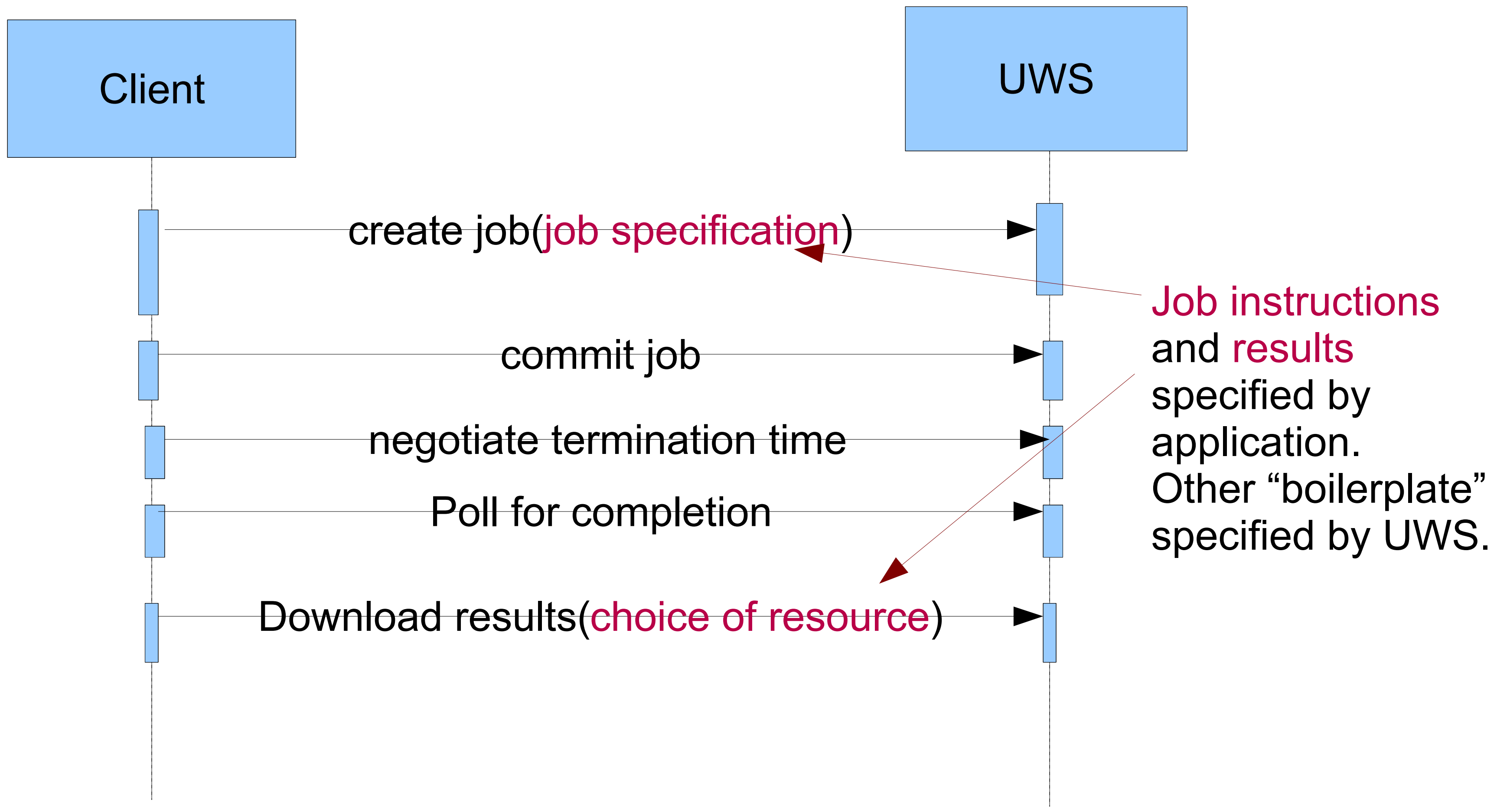
GWS session 2,
IVOA Beijing Interop
May 2007

What is UWS?

- A pattern for controlling asynchronous jobs
  - Post instructions to UWS to create a **job**
  - Review **quoted completion time**; commit job to execution
  - Poll **phase** of job until “COMPLETED”
  - **Results** cached on server; client downloads later
  - **Termination time** for results negotiable
  - Delete job when results fetched...
  - ...or just abandon it and let it time out.
- http://www.ivoa.net/internal/IVOA/IvoaGridAndWebServices/UWS-0.3.pdf

Applying the UWS pattern

- UWS pattern + application = service protocol


Client
UWS
create job(job specification)
commit job
negotiate termination time
Poll for completion
Download results(choice of resource)
Job instructions and results specified by application. Other “boilerplate” specified by UWS.

SOAP or REST

- UWS spec has both SOAP and REST bindings
- Affects how you download results:
  - SOAP: all results packed in one XML doc
  - REST: one web resource per result; MIME types vary
- Clients:
  - SOAP binding needs custom, rich client
  - REST binding can be driven by web browser
- Otherwise, no semantic difference
- Choose the binding that best fits the application
- Expect most IVOA standards would use REST.

What is data staging?

- SIAP as an example:
- Find virtual images using query on catalogue: quick
- “Stage” selected images into service cache: slow
  - Images may have to be computed or got from off-line storage
  - Staging runs asynchronously
- Download images from service as each staging job completes
  - (Or have them pushed to a VOSpace.)

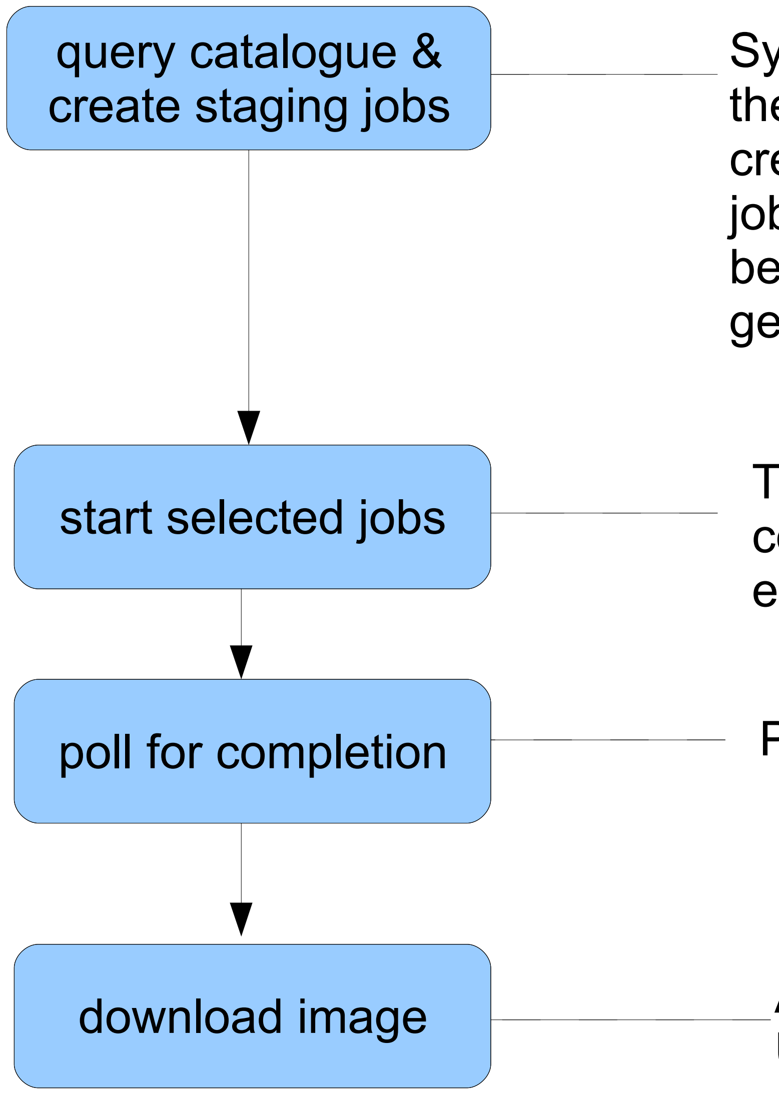
UWS for data staging

- Caveats:
  - My interpretation, not DAL-WG policy
  - This differs from data staging in UWS v0.3 spec
- *queryData* is synchronous; *stageData* is asynchronous => use UWS on *stageData* only
- DAL is RESTful, so use UWS REST binding
- One UWS job per staged image
  - Because it's simpler
  - Because it gives more control to the user

Demo

- Stages INT-WFS images from Cambridge
- This is prototype code demonstrating UWS
- It's not a released, supported science service
- Runs on my laptop :)

What it's doing



Synchronous: blocks until query is complete then returns VOTable. Using VOTable, creates staging jobs in PENDING state; adds job URIs to the table. Query is POSTed because it creates web resources. Query gets delegated to SIA v1 in Cambridge.

Triggering controls are UWS. URIs for those controls are in the table. Unselected jobs eventually time out and go away.

Pure UWS.

Application specifies that an image results. UWS reveals its URI.

How it works

1) Query image catalogue
   - Delegated to existing SIA in Cambridge
   - Request is POSTed (because it creates resources on server)
   - Returns SIA VOTable, synchronously
2) Create UWS job for each image returned
   - Jobs are created PENDING
   - User selects which jobs to activate; others time out
3) Each job → one image → one download URI
   - Images cached on server
   - Deleted when job deleted/timed out

Parts list

- One Java web-app
- Recycled, original SIA v1 in Perl
- 3 servlets
  - query, job-list, job
- 8 JSPs
  - One for each UWS resource
  - Some UWS resources have two JSPs: XML and HTML
- 4 non-servlet Java classes
- 2 HTML pages