NRC·CNRC

Managing Web Service Evolution with VOSI-capabilities

Patrick Dowler
Canadian Astronomy Data Centre

What is VOSI-capabilities?

- a web service endpoint for a self-describing service
  - e.g. http://example.net/service/capabilities
- capability: a single feature
  - standardID attribute: what feature is this?
  - contains 1+ interface(s)
- interface: a single callable endpoint
  - contains 1 accessURL
  - contains 1 securityMethod* (optional)
  - exists mostly to create the accessURL,securityMethod pair
- accessURL: the usable (possibly base) URL
- securityMethod: a constant specifying what kind of authentication the sibling accessURL supports or requires
- with a registry extension to define a capability sub-type, other information about the service can be included
- do not need a registry extension to use VOSI-capabilities!!

How do I use VOSI-capabilities?

- chose an appropriate IVOA standardID or…
  ... define a custom standardID for each feature of the service

- add a VOSI-capabilities resource to the service
- write clients to read the capabilities document instead of embedding service URLs in code or config

- clients read the capabilities document and look for the combination of {standardID,securityMethod} that match:
  - the feature they want to invoke
  - the credentials they want to use to authenticate

Service Evolution Use Cases

- change in REST API
  - behaviour (redirects, side effects)
  - error messages/code clients do something with
  - bug fix where existing client actually depends on the bug
  - shift of some responsibility from service to client

- change in input and/or output format
  - clients can read and write xsd version N
  - server can read and write version N
  - data model change ~ xml schema change → version N+1
  - want to switch to version N+1

Service Evolution : multi-stage release

- increment version in standardID of the feature
- add new endpoint and capability to service with the new standardID
  - different accessURL in the same service
- deploy service (supports current and new clients)

….

- modify client to look for new version (standardID) and work correctly
- release client to users

….

- wait for users to upgrade…
- wait for use of old client to stop…
- track down users and help them upgrade…

….

- eventually remove old endpoint from service

Service Evolution : multi-stage release

- probably hard to support more than two adjacent versions
  - could be weeks or months, probably not a year
- pros:
  - eliminates “downtime” of a monolithic client-server release
  - allows some users to straggle behind for awhile
  - allows for limited releases to detect and/or minimise the impact of subtle side effects
- cons:
  - version number alone makes it hard to write client tools to detect that they are out of date (parsing/ordering versioned standardID)
  - deprecated flag and message inside the capability?
- con?
  - this usage doesn’t really match the cadence of IVOA standard minor versions… this usage allows for much higher cadence of minor versions