Desktop Integration with VO
VO-Client, DAL-Client

IVOA Victoria, May 2006

Doug Tody (NRAO/US-NVO/IVOA)

DAL-5 Agenda

- Current and Future Implementations
  - John Taylor – SIAP client experiences in Astrogrid
  - F. Bonnarel, F. Pierfederici – Footprints
  - Doug Tody – VO-client/dalclient
  - Kelly McCusker – Spectrum data model Java implementation
  - John Taylor (for Noel Winstanley) – Astro Runtime
- Conclusion of DAL WG Sessions

GROUP PHOTO at 12:30, location TBA
Please remember to upload your presentations!

VO-Client / DAL-Client

- Architecture issues
  - User environments
    - Hands-on user wants high level scripting environment
    - Python, IRAF, IDL, PHP, Ruby, etc. (NVO Summer school)
    - C/C++ for lower level interfaces
    - Java for Web-based desktop applications
  - VO framework
    - Java primarily; also primary language for web/grid software

VO-Client / DAL-Client

- Architecture issues
  - Problem
    - User environments are C/C++ based; VO framework is Java
    - Java does not mix well with other languages
  - What to do?
    - JNI problematic, has limited applicability
    - Best solution is probably to use distributed computing architecture
    - Languages mix ok over process boundaries with runtime binding
  - Conclusion
    - We can't avoid client-side Java for desktop integration with VO
    - Use C/C++ client-server APIs to integrate with user environments

DAL-Client Java Library

- Motivation
  - Provide client-side reference implementation of DAL interfaces
  - Not really for users, but as core of client-side support
- Approach
  - Use Java since this is the language of choice for VO framework
  - Will integrate well with VOTable parsers, VOSpace etc.

DAL-Client Java Library

- Scope
  - Registry queries for data collection, service discovery
  - Client-side of DAL interfaces (SCS, SIA, SSA so far)
- Functionality
  - Multi-level: stream, votable, data model
  - At highest level provides client with data-model view
  - Independence from protocol details
  - versioning, authentication, asynchronous operations, etc.
  - Capability for async distributed queries, join of results

DAL-Client

- Class Hierarchy
  - DatasetQuery (generic dataset)
    - ConeQuery
    - SiapQuery
    - SsapQuery

US-VO Website - Mozilla Firefox Thu May 18, 2006 08:37:09 Victoria

File Edit View Go Bookmarks ScrapBook Tools Help

ConeQuery Classes

The `ConeQuery` classes implement the simple cone search query, used for generic catalog access. Since there is no data model for a generic catalog query, the query response object for a cone search uses the UCD of each returned table field as the lookup key. For each row of the returned table one can use the `getAttribute` method to fetch catalog fields directly using the UCD as the key.

```
cone = new ConeConnection()                # Connection context
cone = new ConeConnection(service-url)
          cone.addService(service-url)
     cone.getServiceCount()
       cone.getServiceURL(index)

query = cone.getConeQuery()                # Query context
query = cone.getConeQuery(ra, dec, sr)
       query.addParameter(name, value)

       qr = query.execute()                # VOTable -> Dataset|Record
         query.executeCSV()                # VOTable -> CSV
         query.executeCSV(fname)
vot = query.executeVOTable()               # query as VOTable
 is = query.executeRAW()                   # raw XML
```

US-VO Website - Mozilla Firefox Thu May 18, 2006 08:39:21 Victoria

File Edit View Go Bookmarks ScrapBook Tools Help

SiapQuery Classes

The `SiapQuery` classes implement the simple image access query, used to discover candidate image datasets which can subsequently be downloaded if desired. In this case the query response implements the SIAP data model, which defines a number of standard attributes used to describe each candidate image dataset. The `getDataset` method can be used to retrieve the image associated with a given row of the query response.

```
siap = new SiapConnection()
siap = new SiapConnection(service-url)
          siap.addService(service-url)
     siap.getServiceCount()
       siap.getServiceURL(index)

query = siap.getSiapQuery()
query = siap.getSiapQuery(ra, dec, size)
query = siap.getSiapQuery(ra, dec, size, format)
query = siap.getSiapQuery(ra, dec, ra_size, dec_size)
query = siap.getSiapQuery(ra, dec, ra_size, dec_size, format)
       query.addParameter(name, value)

       qr = query.execute()              # VOTable -> Dataset
         query.executeCSV()
         query.executeCSV(fname)
vot = query.executeVOTable()
     is = query.executeRAW()
```

QueryResponse Class

The response to a DAL query can be processed directly as a VOTable, the generic table container format used by the wire protocol. This requires knowledge within the client code of the details of the DAL protocol used, but provides a generic way to process the results from any query. The `QueryResponse` class goes one step further and provides an object API based on the data model for a particular object as defined by the DAL interfaces (catalog, image, spectrum, etc.). This permits an application to interact with a remote data object service without detailed knowledge of the underlying DAL protocol used. In addition since the query response table is stored in memory in a hash table, efficient random access by keyword is provided.

The `QueryResponse` class is a sequence of dataset descriptors corresponding to the rows of the query response VOTable. Each dataset is described by a set of standard data model attributes. A data provider may add additional non-standard attributes which will flow through all layers the interface to the client code. These are not visible in the standard data model via the normal query-by-attribute interface, but can be accessed by traversing the response record as a list.

```
Get dataset descriptor:

   qr.getRecordCount()                  # number of descriptors
  rec = qr.getRecord(i)                 # get a descriptor

Access by dataset attribute:

v = rec.getAttribute(attrname)          # may return null

        v.boolValue()
         v.intValue()
       v.floatValue()
      v.doubleValue()
      v.stringValue()

Get dataset file corresponding to descriptor:

      rec.getDataset(path)              # fetch data to given path
path = rec.getDataset()                 # fetch to autogenerated path
```

The `getDataset` method may optionally be used to generate and retrieve the dataset described by a given query response record. A filename will be automatically generated if one is not provided. An attempt is made to determine the type of data returned (FITS, JPEG,

VO-Client

- What is it?
  - Client-server package built on top of dalclient
  - Provides support for multi-language client APIs (Python etc.)
- Functionality
  - Data discovery
  - Virtual directory management (based on queries)
  - Client-side data access
  - VOStore/VOSpace for network, cluster data caching
  - Object APIs for data analysis (e.g., SAO SSA Java library)