INTERNATIONAL VIRTUAL OBSERVATORY ALLIANCE
US National Virtual Observatory

IVOA Data Access Layer
Table Access Protocol (TAP Version P [P+Q])

Doug Tody (NRAO/NVO)

TAP Tiger Team Discussions

- Held
  - November 19–20 2008 at JHU
- Purpose
  - Understand partner project requirements
  - Discuss and agree upon main issues concerning TAP
  - Provide basis for further work, broader discussions
- Attending (alphabetical order)
  - J. Good, Bob Hanisch, K. Noddle, F. Ochsenbein, P. Osuna, Alex Szalay (organizer), D. Tody (editor); also R. Plante (registry), M. Graham (GWS)

TAP Tiger Team Discussions

- Meeting Notes
  - http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/TapJhu
- Key Topics and Agreements
  - Agreed need full up ADQL query with VOSpace integration, async, etc.
    - simple synchronous GET version should also be provided
  - Agreed need ParamQuery (formerly SimpleQuery)
    - uniform access to both table data and metadata
    - primary mechanism used to query table metadata
    - provide position-based search capability to replace cone search
    - include region capability for more general regions (STC based)

TAP Tiger Team Discussions

- Key Topics and Agreements (cont'd)
  - Metadata Queries
    - minimal core TAP schema based upon registry model
    - also provide "tableset" metadata in dataless VOTable and XML
  - Interface Consistency
    - TAP interface consistent with DAL service profile and semantics
  - Minimal TAP Service
    - implements ParamQuery for both data and metadata, core TAP schema
    - VOSI support (getCapabilities etc.)
  - VOSpace integration
    - strategy for how to do this was discussed

Required or Advanced Capabilities

- ADQL query capability
- Support for multiple query languages (e.g., SQL pass-through)
- Simple parameter query (90/10, robustness)
- Minimal TAP service (small data providers)
- Uniform query interface for data and metadata
- TAP schema (core, extensible)
- Tableset metadata (registry XML, VOTable)
- Uniformity of DAL service family interfaces
- Multiple table output formats (VOTable, CSV/TSV, text, FITS, etc.)
- Support for both "narrow" and "wide" table output
- Application-specific error customization
- Inline table uploads (via POST)
- VOSpace integration for both query input and output
- Support for large streaming queries
- Cone search capability to replace legacy cone search
- Multi-position queries
- Region-based queries (STC regions)
- UTYPE (data model) and UCD-based queries
- Support for Google-like ranking of queries (advanced)
- Support for propagation of table update
- Both synchronous and asynchronous (UWS) execution
- Authentication (anonymous, SSO)
- VOSI support (capabilities, availability, tableset metadata)
- Distributed job tracking (RUNID)

Service Interface

- Operations
  - *AdqlQuery* ADQL (or other QL) queries
  - *ParamQuery* Parameter queries (fully defined, no parser)
  - *GetCapabilities* VOSI interface for service metadata
  - *GetAvailability* VOSI interface for service monitoring
  - [UWS interface] Used to monitor asynchronous jobs (TBD)
- Commonality
  - AdqlQuery, ParamQuery share much of the same implementation
  - Primary difference is in the form of the query (and complexity)
  - Back-end (query execution, output formatting) is the same for both

Common Elements

- Table name syntax
  - [[<*catalog*>"."[<*schema*>"."]]<*table*>
- Field name resolution
  - UTYPE or UCD references resolve to a physical table field name
    - Uses namespace, e.g., “ssa:target.name”, “ucd:instr.bandpass”
  - All query evaluation is done on physical table fields
  - UCD is a special case of a UTYPE (data model)
- Inline table uploads
  - Tables (or regions etc.) can be uploaded inline in a query
  - Tables can be directly queried: “$UPLOAD.*tableName*”

Common Elements

- VOSpace integration
  - Use VOSpace tables for both input and output
    - The user's VOSpace appears as a DBMS schema in queries
    - Tables can be used directly in queries: "$VOSPACE.*tableName*"
    - Output tables saved at server, can be used in a subsequent query
  - Provides per-user storage, persistence
- Asychronous execution
  - Required for large queries; UWS used to monitor execution
- Multiple Output formats
  - Allow client to get data/metadata in desired format
  - VOTable (default), CSV/TSV, FITS, text, html, etc.
- TAP schema
  - Standard data model for Table/DBMS metadata; extensible

AdqlQuery

- Input parameters
  - Query, QueryType — URL-encoded ADQL (or other) expression
  - Format — Output data format specification (any)
  - Maxrec — Allows streaming of large queries
  - Mtime — Allows propagation of table update/add/delete
  - RunId — Monitoring of distributed jobs
  - Output — Direct output to VOSpace, initiate async
- Functionality
  - All query specification is done using ADQL
  - Common functionality for output formatting, data staging, etc.

ParamQuery

- Input parameters
  - Pos, Size — Single or multiple position ("cone search")
  - Region — More general regions
  - Select,From,Where — Simplified param-based SQL like query
  - Top — Heuristic-based (Google-like) query
  - Format,Maxrec, — Common with AdqlQuery
  - Mtime,RunId,Output
- Functionality
  - All query specification is parameter-based, constrained, robust
  - Common functionality for output formatting, data staging, etc.

ParamQuery

- Table metadata queries
  - Metadata is represented like data tables (TAP Schema)
  - Entire query interface can be used for metadata queries
  - VOSI/Registry compatibility is easily provided as well
- Basic Examples
  - FROM=TAP_SCHEMA.tables
  - FROM=TAP_SCHEMA.columns &WHERE=tableName,fp_psc
  - FROM=TAP_SCHEMA.tableset &FORMAT={xml|votable}
- More Advanced Examples
  - FROM=TAP_SCHEMA.tables &WHERE=tableName,$VOSPACE.*
  - FROM=TAP_SCHEMA.tables &POS=xx&SIZE=yy

(ADQL could also be used for metadata queries of course)

TAP Schema

- Concept
  - Same concept as SQL92 INFORMATION_SCHEMA
  - That is, represent DBMS/Table metadata as data tables
  - Allows power of RDBMS mechanism to be re-used for metadata queries
  - Easily extensible since it is data, not interface

- TAP_SCHEMA.tables
  - *TableName* Table name including catalog and schema
  - *Description* Brief description of table
  - *TableType* Base_table, view, output
  - *Utype* UTYPE if table corresponds to a data model

TAP Schema

- TAP_SCHEMA.columns
  - *Name* Column name
  - *TableName* Table name, e.g., <schema>.<table>
  - *Description* Brief description of column
  - *Unit* Unit in VO standard format
  - *Ucd* UCD of column if any
  - *Utype* UTYPE of column if any
  - *Datatype* Datatype as in VOTable/Registry
  - *Arraysize* Array dimensions as in VOTable/Registry
  - *Primary* Column is visible in default selection
  - *Indexed* Column is indexed on the server
  - *Std* Standard column (as opposed to custom)

ParamQuery

- Multi-Position Queries
  - Concept
    - User uploads or references table containing multiple positions
    - Perform same query for each position
    - Scale up to many objects; first phase of cross-match
  - Approach
    - Generalize POS,SIZE to multi-position
    - POS=@UPLOAD.tableName (VOSpace, DBMS can also be used)
- Output
  - A single table containing data for all positions
  - Rows for a given position are tagged by postion ID
  - Either sync/async is possible

ParamQuery

- "Cone Search" Capability
  - Motivation
    - Simple cone search most successful VO service of all time!
    - Obsolete/replace legacy simple cone search capability
    - Replace with more powerful, but still simple, capability
    - Supports multiple data collections, UTYPE/UCD, etc.
    - Non-positional queries are always possible - not a limitation
- Approach
  - ParamQuery, POS/SIZE, optional SELECT, WHERE constraints
  - REGION can be also be used to generalize search region
  - Multi-position version can be used to scale up
- Example
  - FROM=tableName &POS=x,y&SIZE=z &WHERE=flux,3/