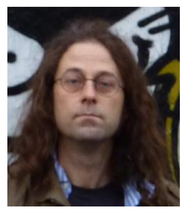
TOPCAT's TAP Client

Mark Taylor, University of Bristol, UK



TAP into Australia!

Introduction

TAP, the **Table Access Protocol**, is a Virtual Observatory (VO) protocol for executing queries in remote relational databases using **ADQL**, an SQL-like query language. It is one of the most powerful VO components, but also one of the most complex to use, with an extensive stack of associated standards.

This poster presents significant improvements to the GUI TAP client in the recent release of **TOPCAT**, a desktop table analysis tool. It attempts to give the user as much help as possible in locating services, understanding service metadata and capabilities, and submitting correct and useful ADQL queries. The implementation is both usable and performant for very large TAP services.

This GUI client is also available for standalone or embedded use.

General Considerations

One window: Because of its position within the, already complex, TOPCAT application, the GUI is constrained to fit within a single screen window.

Scalability: Many TAP services have only a few tables, but some have many (∼900 in HEASARC, ∼30 000 in TAPVizieR). Both user interface and data access have to be designed with this in mind.

Fault tolerance: TAP services are of variable quality. An attempt is made to provide best-efforts behaviour when interacting with non-compliant or partially implemented services.

Standards Evolution: TAP and its associated standards continue to evolve. Some attempt is made to provide forward, as well as backward, compatibility.

Service Discovery

Science users typically know the data sets they want to query (CALIFA, WISE) rather than the names or locations of the services hosting them (GAVO DC, HEASARC). So when selecting a TAP service to query, it's important to be able to locate services by searching against table metadata (table name, table description) not only service metadata (organisation name, service title).

The IVOA Registry does not currently have sufficient detail to support these searches. So we cheat. GAVO at ARI Heidelberg maintains a database called Global TAP Schema (**GloTS**), which contains the table-level information we need. This is queried (using TAP, since it is itself a TAP service) to locate services of interest.

The implementation is pluggable, so if the registry acquires sufficient content for these searches in the future, the client can be switched to use that instead.

Job Submission Mode

TAP allows you to submit queries in **Synchronous** or **Asynchronous** mode; sync is more straightforward but async is suitable for long-running jobs. You can choose which you want, and the result is loaded into TOPCAT for further analysis.

Alternatively you can choose **Quick Look**, which makes a synchronous query but just displays the result in a window, and does not load it into the application. It's useful for experimental queries, e.g. counting the rows in a table.

ADQL Editor

The editor panel is where you enter the ADQL to be executed. It has several features:

Query Validation: The ADQL is checked as you type, and errors are highlighted. As well as standard ADQL syntax, the validator is aware of the tables, columns, and user-defined functions available from the service.

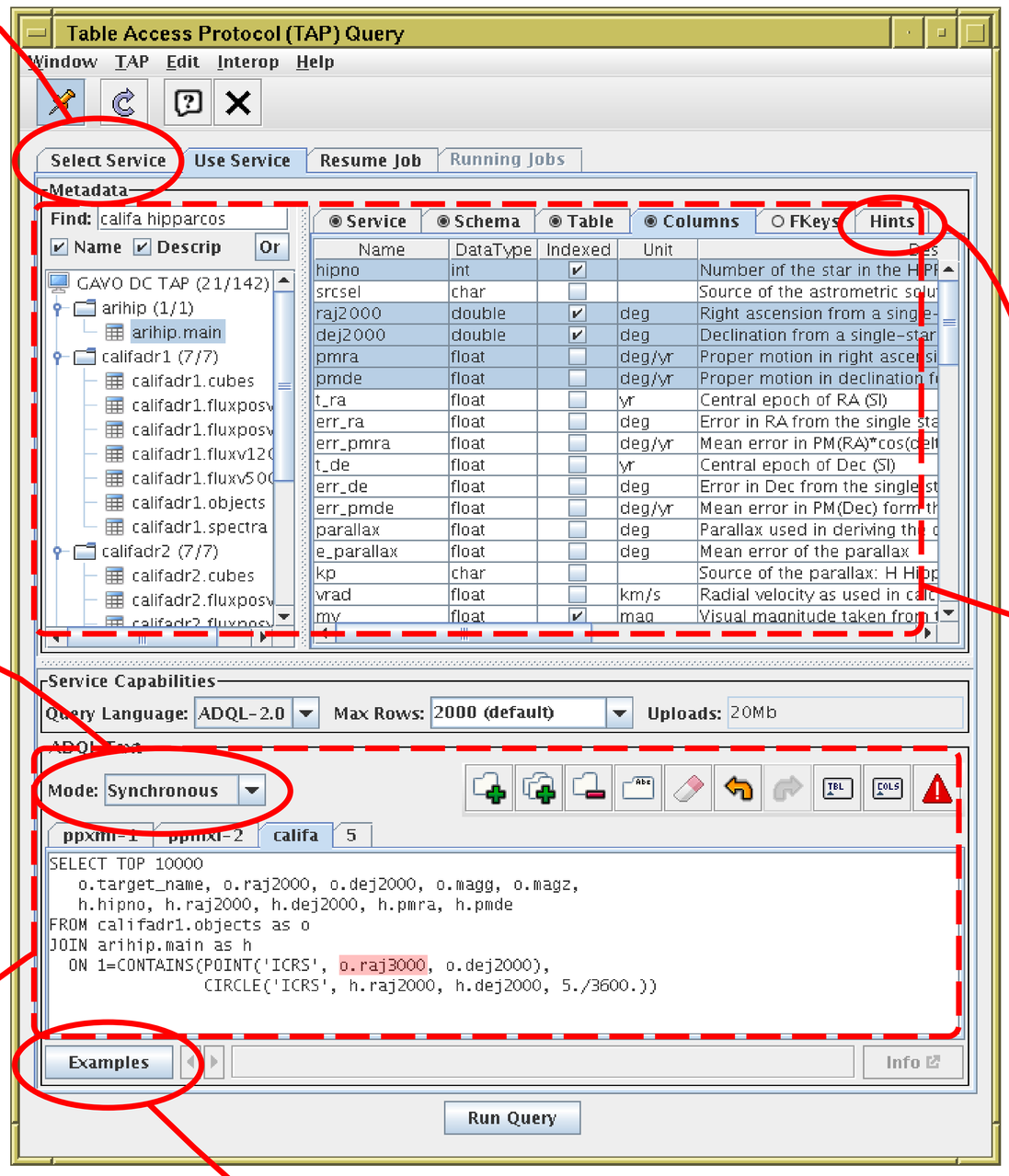
Tabs: You can have multiple queries on the go in different tabs. Tabs can be given names and content copied to new ones.

Undo/Redo: Full undo/redo functionality for text edits is provided from the keyboard or toolbar buttons.

Metadata Paste: There is limited support for selecting table and column names in the metadata GUI and pasting them into the text window, to cut down on typing.

Acknowlegements

This work has benefitted from many members of the TAP/IVOA community; special thanks are due to **Markus Demleitner** (GloTS & expert on all things TAP) and **Grégory Mantelet** (ADQL parser library), both at ARI Heidelberg.



ADQL Examples

Most astronomers are not, at least initially, fluent in ADQL or SQL, so need some help with the syntax. One possible approach is to provide a graphical query builder that constructs a SELECT statement from a series of GUI interactions (e.g. selecting tables, columns and comparison operations from drop-down menus). That can be effective for simple queries, but it's difficult to generalise to sophisticated operations.

Instead we concentrate here on providing a library of example ADQL queries that a user can use, edit, adapt and learn from. These fall into three categories:

Standard: Standard examples use standard TAP features, though some, for instance upload-based ones, may not be applicable to services that lack certain capabilities. These examples use table metadata declared by the service, so can be used as-is to make working (though not necessarily useful) queries on the database at hand.

Data Model-Specific: TAP services may declare that they support certain standard data models, for instance ObsTAP, which stores astronomical observation metadata in a standard format for querying. Standard queries based on such common data models (currently: RegTAP, ObsTAP and TAP_SCHEMA) are available for services that support the relevant models.

Service-Provided: TAP services may provide their own lists of data-specific examples from the standard `/examples` endpoint. These can be extremely useful to guide users in making best use of the available data holdings. The examples document is XHTML marked up with RDFa; the details of the format are currently under discussion, but if you're a TAP service provider not already providing examples in this way, please consider it!

Metadata Acquisition

The client has to know information about the tables provided by the service, (1) to present to the user and (2) to be able to validate ADQL queries (check existence of the referenced tables and columns).

There are a couple of ways to get this information from the service; from the `/tables` endpoint as a flat XML document or by querying the standard `TAP_SCHEMA` tables in the database itself.

For small databases (a few tables) it makes sense to read all the metadata at once. For large databases (e.g. VizieR has 30k tables, 430k columns, 100Mbyte? table metadata, nearly all of which the user won't need) it's a bad idea. So metadata acquisition is done in a pluggable way; different backends exist for different acquisition strategies. By default an adaptive strategy is used (<5 000 columns, read all metadata up-front; more than that read just table names up front and defer reading column content until it is required). But the expert user is able to choose a strategy to taste.

In the deferred case care has to be taken to acquire column metadata in a way that appears responsive from the GUI without overloading the service. A bounded LIFO queue of asynchronous metadata requests is maintained to achieve this.

ADQL Hints

A **Hints** tab shows a very basic ADQL Cheat Sheet, with reminders about SELECT statement syntax and pointers to a few other resources on the web.

Metadata Display

The user has to be able to see metadata describing the TAP service to be able to formulate queries. This metadata is rich and potentially large.

A standard combination of a tree and tabbed pane is used for presentation. the user selects a table of interest in the tree, and the tabs on the right are populated with different information about it:

Service: Service metadata including that gathered from the IVOA Registry (name, description, organisation, curation, external links, ...) and from its `/capabilities` endpoint (supported data models, query languages, and user-defined functions).

Schema: Name and description of the schema-level grouping of tables (only useful for services that use this grouping structure).

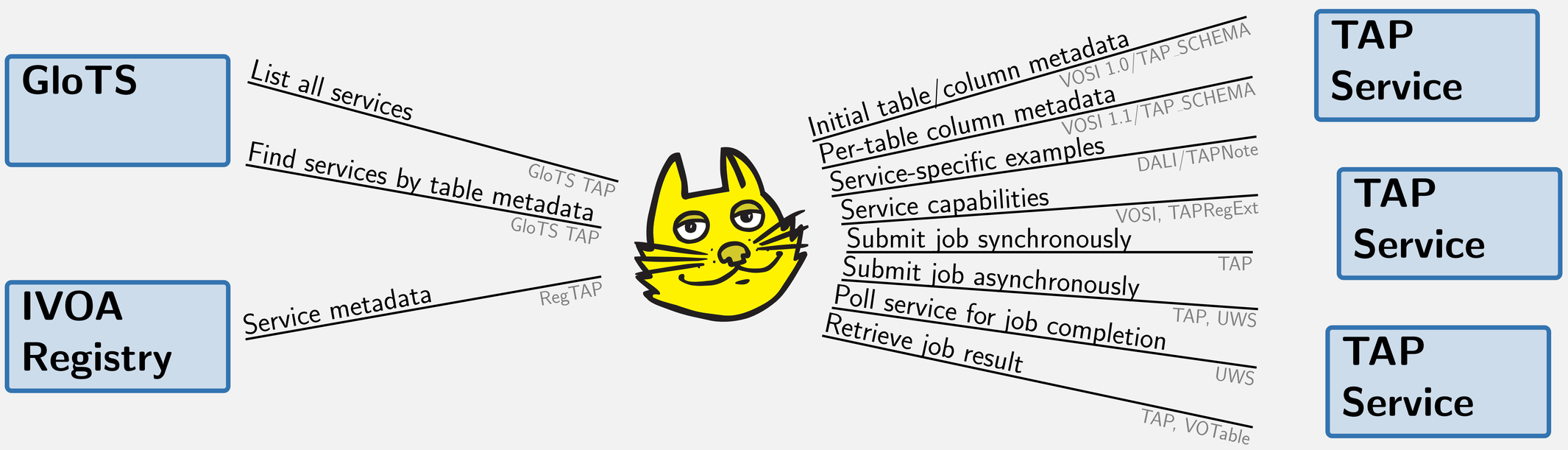
Table: Table name and (perhaps detailed) description.

Columns: List of all columns in the selected table, including name, data type, units and description, presented in tabular form. Tables can be wide (∼500 columns for SDSS `Photo*`), so browsing this can be challenging. Currently you can't search (should this be added?) but you can sort the list, e.g. by column name (useful if you know or guess the name of the column you want) or units (useful to group e.g. magnitudes, positions or proper motions).

Foreign Keys: Lists relationships between tables.

For large services, browsing the tree of thousands of tables is not useful, especially if the tables have unintuitive names. The **Find** box allows you to enter search terms that instantly restrict the visible tables by table name and/or description. Thanks to the authors of Seleste (CfA), from which I stole this idea.

Communications between TOPCAT and Services



$Id: poster.tex,v 1.14 2015/10/16 09:52:31 mbt Exp $