Volume Title
ASP Conference Series, Vol. **Volume Number**
Author

Table Access Protocol applied to the Simbad Database

Grégory Mantelet[1], Marc Wenger[1], and Laurent Michel[2]

[1]*Centre de Données astronomiques de Strasbourg*

[2]*Observatoire Astronomique de Strasbourg*

Abstract. We have implemented in the Simbad service the Table Access Protocol (TAP), a standard of the Virtual Observatory defining a protocol for accessing astronomical catalogs and database tables using queries written in the Astronomical Data Query Language (ADQL), another VO standard.

Implementing TAP requires several steps: the definition of a database schema with the data exposed to the users, the translation of ADQL queries into regular SQL language performing the queries in Simbad, and the implementation of the Universal Worker Service (UWS) standard to manage asynchronous queries, useful for long queries, either by their duration or their output size.

These standards were implemented as much as possible in a generic way, allowing them to be reused in other services, as it has already been done in the database generator SAADA. The ADQL to SQL translator uses callbacks to implement the specific routines for a given service. All these librairies have been designed as autonomous packages, easy to reuse with very few specific developments. The whole development is in the Java language.

1. TAP : Table Access Protocol

The Table Access Protocol (TAP) (Dowler et al. 2010) is a protocol of the Virtual Observatory (IVOAProject 2002) for accessing any kind of table data, astronomical catalogs as well as relational databases. The main query language that must be implemented is the Astronomical Data Query Language (ADQL) (Ortiz et al. 2008). Queries can be submitted in either synchronous or asynchronous mode. Asynchronous mode is based on the Universal Worker Service (UWS) (Harrison et al. 2010). Outputs should at least be VOTable, but any other output can be implemented (e.g. `CSV`, `TSV`, `JSON`).

2. Implementing TAP for SIMBAD

SIMBAD (Wenger et al. 2006) is a relational database implemented on Linux servers running the Postgresql DBMS. Without changing anything to the existing query modes, we have added an interface to the TAP service. This will allow more flexibility for users to perform complex queries not always available in the current query modes. It will also allow generic client applications, like Topcat (Taylor 2011) and more recently TapHandle (Louys et al. 2012) to access SIMBAD using the TAP protocol. Implementing TAP

requires several steps:

- the definition of a database schema containing the tables and columns exposed to the users.
- the translation of ADQL queries into regular SQL queries adapted to the SIMBAD database.
- the implementation of the UWS standard to manage asynchronous queries.

TAP on SIMBAD is then available through URL access and a dedicated web page. All the libraries described below are written in Java.

2.1. Defining a database schema

It may be useful, if not mandatory, to expose to the users only a subset of the whole database schema (for hiding confidential data, or some technical data for instance). Therefore a sub-schema of the database describes tables and columns available through ADQL. Tables and columns can also be renamed. This allow evolutions of the database schema itself that will have no incidence on already existing, and maybe stored, ADQL queries. A dictionary of these names is then used during ADQL query parsing to translate the names from external ADQL queries into internal SQL ones.

2.2. Translating ADQL queries

Translating ADQL queries into SQL queries requires an ADQL parser and an SQL generator.

The ADQL parser has been developped using a lexical analyzer and a grammar definition in the JAVACC parser generator (JAVACCProject 2011). It implements ADQL release 2.0. Several complex functionalities are not yet implemented: STC-S, REGION functions and coordinate system conversions.

The SQL generation can be partly database dependant. In order to build an ADQL parser as generic as possible, several entry points allow interventions on the final result (the SQL statement): therefore, the ADQL tree has been made accessible to change for instance the table and column names. The SQL translator can also be specialized by overlaying generic methods or implementing abstract ones to adapt the translation to some specific feature.

For instance, in SIMBAD, we implemented most of the geometrical functions by installing pgsphere, a spherical data types extension for Postgresql (pgsphereTeam 2011). Queries using these types are not regular SQL instructions and need therefore a specific syntax, which need to be written in some overlayed method of the SQL translator. Another coordinate management library in Postgresql is Q3C (q3cProject 2011), which requires also its own implementation, will be integrated in the library.

Several functionalities have also been implemented like an ADQL tree browser (search and replace methods) and a simple ADQL query checker, returning detailed error messages.

ADQL is packaged in a library (Mantelet 2011a), associated with a documentation describing the API, as well as the interfaces and abstract methods, allowing to personalize the translation, for instance mapping table and column names. The current version is 1.0 beta.

2.3. Implementing UWS

The Universal Worker Service is a VO service pattern which modelizes the management of any kind of asynchronous tasks (also called jobs). In TAP, these tasks are ADQL queries. They are sent by the user, put in a stack managed by the UWS service and executed at their turn. Once finished, their output is stored on server side and returned to the user on his request. At any moment, the user is able to query the service to know the status of his queries (e.g. pending, executing, done). He has also the possibility to cancel his own queries, and to configure the UWS service.

We have developed an UWS library (Mantelet 2011c). The library has a documented API and describes the two elements left to be developed by the user:

- the implementation of the *AbstractJob* class to say what a job is supposed to do (e.g. cross-matching, image processing, executing an ADQL query).
- the servlet which forwards the requests to UWS and return the answers. This includes the management requests (e.g. get the query status, cancel a query, get the query result).

2.4. The TAP library

Like the ADQL and UWS libraries, we have developed a generic TAP library (Mantelet 2011b) which can be used for any implementation. The specific development required by a given service (e.g. SIMBAD) has been moved to interfaces and classes left to the user:

- the *DBConnection* interface which manages all the connections with the database. Mainly database interrogation, but also temporary table creation, table deletion and row insertion if the Upload facility is enabled in the service.
- the *ServiceConnection* interface to access all the UWS execution parameters and the TAPFactory class which provides object creation for the *DBConnection*, request executor and so on.
- the *TAPFactory* class must provide five object creators: request executor, query checker, ADQL translator, database connector and uploader object.
- the *OutputFormat* interface which is used to format outputs. One class must be developped for each output format. VOTable is mandatory. Any other output format can be offered.

3. Present status

The UWS library is already widely distributed. It is accessible at
`http://cds.u-strasbg.fr/resources/doku.php?id=uwslib`.

The ADQL library is operational except for some complex requirements. It is already used in SAADA (Michel et al. 2010) and in SIMBAD.

The TAP library is used in SIMBAD and its SIMBAD interface has its own web page for submitting queries. It is also available through generic clients like Topcat and TapHandle.

It will be officially opened to the community at the end of 2011. This will be advertized through the SIMBAD web pages and the CDS news.

Conclusion

Developing a TAP service from scratch implies developing several librairies. We choosed to split the development into two parts: a generic part building up common librairies which can be reused for any service; and interfaces and abstract classes which must be implemented to plug in the specific parts requested by a given service. The whole development has been validated by two concrete implementations: SAADA and SIMBAD. The Vizier database service has begun to use these libraries for developing its own TAP service.

References

Dowler, P., et al. 2010, Table access protocol. http://www.ivoa.net/Documents/TAP/
Harrison, P., et al. 2010, Universal worker service pattern.
http://www.ivoa.net/Documents/UWS/
IVOAProject 2002, International virtual observatory alliance. http://www.ivoa.net/
JAVACCProject 2011, Javacc. http://javacc.java.net/
Louys, M., et al. 2012, in ADASS XXI, edited by Ballester, P. and Egret, D., vol. TBD
Mantelet, G. 2011a, Adql library. http://cdsportal.unistra.fr/adqltuto/
— 2011b, Tap library. http://cdsportal.unistra.fr/taptuto/
— 2011c, Uws library. http://cdsportal.unistra.fr/uwstuto/
Michel, L., et al. 2010, in Astronomical Data Analysis Software and Systems XIX, edited by Yoshihiko Mizumoto, Koh-Ichiro Morita, and Masatoshi Ohishi (San Francisco: Astronomical Society of the Pacific), vol. 434 of ASP Conference Series, 491.
http://saada.u-strasbg.fr/saada/
Ortiz, I., et al. 2008, Astronomical data query language.
http://www.ivoa.net/Documents/ADQL/2.0
pgsphereTeam 2011, pgsphere postgresql extension.
http://pgsphere.projects.postgresql.org/
q3cProject 2011, q3c postgresql extension. http://code.google.com/p/q3c/
Taylor, M. 2011, Topcat software. http://www.star.bris.ac.uk/~mbt/topcat/
Wenger, M., et al. 2006, in Astronomical Data Analysis Software and Systems XV, edited by Carlos Gabriel, Christophe Arviset, Daniel Ponz, and Enrique Solano (San Francisco: Astronomical Society of the Pacific), vol. 351 of ASP Conference Series, 703.
http://simbad.u-strasbg.fr/simbad/