Multimessenger Astronomy with ObsTAP and PyVO

Hendrik Heinl, Dave Morris, Markus Demleitner

November 8, 2020

Abstract

The shear amount of data available for astronomy today, offered by distributed services not only demands a specific set of skills from (data) scientists: it's obvious that automated processes and machine readable (machine "learnable") interfaces and standards are necessary to combine data from different sources, instruments and messengers. The IVOA defines such standards and protocols and introduces them to the community. At the same time, the ESCAPE project links partners from different instruments with the european VO community to enable a FAIR access to the data.
Here we want to show which data already is available and at the same time want to introduce to the recent standards that enable you access data using pyVO, the Python API to the Virtual Observatory. Eventually we want to give a glance on what's in the pipeline in regard of new standards.
This tutorial is not an introduction in any of the used standards, protocol tools or python. To get more insight into this, you may want to have a look at the References section of this document.

Software: TOPCAT V4.7, Aladin V11.0

1 Starting out

We will use the VO tools TOPCAT, Aladin and pyVO. These client software can be found here:
Aladin: `http://aladin.u-strasbg.fr/`
TOPCAT: `http://www.star.bris.ac.uk/~mbt/topcat/`
We will also use PyVO and astropy. These are best installed from your distribution packages.

A comprehensive collection of the examples in this tutorial as well as installation scripts you will find on github. Checkout the hyantis repository and follow the README instructions.

2 TOPCAT, SCS and TAP

This tutorial will work along the use case of searching for neutrino data from a VO-service and combine these with data from catalogues using Topcat. In the second part we will be using PyVO to find and access observational dataproducts on different VO services. We are well aware that scientifically speaking this does not make a lot sense (yet), but this tutorial is meant to give an overview which standards the VO is providing right now, and what is in the pipeline that will support multimessenger astronomy.

▷ **1** *Finding Neutrino data in the VO-registry –*

We start this use case with searching and accessing neutrino data that is available on VO services. Start Topcat and click on VO → Cone Search. In the Cone Search window enter `neutrino` as Keywords and click on Find Services. After a few moments you see the results coming in as a list of services providing data related to the keyword. You see that the list provides you with some information of what the data of the service contains. In the description you also get information about the different data provides.

VO Registry

The Registry is the central point for searches for services in the VO. VO services identify themselves to the Registry, providing metadata about the data they serve and the service protocols they support. Thus, the Registry is the entry point for data discovery in the VO.

Mark different rows and watch the URL change in the field below. This meta data is meant to give you an idea of what you can expect from the services and if they are of interest for you.

Exercise: Of course you may not be interested in neutrino data. Try to find the messengers that are of importance for you instead: try radio, infrared or gamma ray, or whatever is suitable for you. Note: here the registry is searching for SCS services only.

▷ **2** *Retrieve neutrino data through a Cone Search –*

For our example, we select the service provided by km3net by marking the service by the title ANTARES 2007-2017. We will get a bunch of neutrino events to use in the next steps. In Cone parameters for RA enter 43, for Dec enter 35 and and give a Radius of 20 degrees. Then click on OK. Within seconds the data will be loaded into Topcat and you see the table appear in the main window of Topcat.

The protocol we used is the Simple Cone Search (SCS).

With the neutrino data at hand, let's have a look at it in Topcat. We can click on Views → Column Info. Here you can get some information about the columns. You see that the data provides us with a modified Julian Date of when the neutrino event was ovserved. We can plot some of the data with Graphics, but so far we wouldn't have much more than positions to see.

▷ **3** *Adding catalogue data to the neutrino candidates with TAP –*

In this step we want to combine our neutrino data with catalogue data from SDSS. Catalogue data typically is table data, so we will use the Table Access Protocol and the corresponding Query language ADQL to do so. There is a

SCS

SCS (Simple Cone Search) https://ivoa.net/documents/cover/ConeSearch-20060908.html defines a standard web service interface for requesting data based on a cone projected on the sky described by a sky position and an angular distance. The protocol defines the query parameters the service understands and the format of the response. This enables users to query multiple services using the same criteria. Each service returns a machine readable VOTable document listing the astronomical sources or objects which are within the search criteria.

lot more to say about TAP and ADQL than we will do here. A comprehensive ADQL introduction is linked in the References.

In Topcat go to VO → TAP. The TAP window will open and after a few moments you will see a list of services in the window. Analogously to the Cone Search window, you can use the TAP window to search the VO-registry for data. At Keywords we will enter `SDSS` to search for services providing us with those data. Often you will find the most recent catalogue data from dedicated services related by the publishers of the catalogues. You may also find them on the TAP Service of Vizier. Many Services providers try to anticipate which data combination might be of interest for the users. So it's not uncommon to find surveys like 2MASS or SDSS on several services.

TAP

TAP (Table Access Protocol) defines a standard web service interface for querying tabular catalogs. The protocol defines how to represent the query parameters and the available response formats. This enables users to query multiple services using the same client software and query language.

As for SDSS you see a few options to chose from. We will select the GAVO data center (Note: the reasons here is due to the performance of the tutorial: we want query results to come in fast. Real science takes time and it is completely acceptable to run queries for hours). Click on Use Service to get to the TAP window of the Service.

Give it a few moments to load the metadat of the service into the TAP window. On the left you see a tree of the catalogues and tables on thes service, on the right is the metadata browser on these tables. Mark any table and see what the description of the table and the columns tell you about the data in the table. This meta data still is part of the TAP standard and is very helpful for data discovery and of course for the data access using ADQL.

In the upper left of the meta data at Find: enter sdss to find the table containing this survey. In this example we just want to collect the identifier of an object, the position in ra and dec and the colours in the u, g, r, i and z band. We will make use of the TAP feature TAP UPLOAD, to join our local table with results from the remote service. In the lower window at ADQL Text click on Examples → Upload → Upload Join. You see an example ADQL query appearing in the field:

```
SELECT
   TOP 1000
   *
   FROM sdssdr7.sources AS db
   JOIN TAP_UPLOAD.t1 AS tc
   ON 1=CONTAINS(POINT('ICRS', db.ra, db.dec),
                 CIRCLE('ICRS', tc.ra, tc.decl, 5./3600.))
```

Now at the first glance that may look confusing, so let's go through it step by step. Each ADQL query starts with `SELECT` followed by specifications of "what" to select, what to do with the selected records and how to return the results. `TOP 1000` means the first 1000 data records will be returned, which match the whole query. With `*` we select all columns of matching data records. Here will will modify the query, because we are not interested in all of the columns of SDSS. Instead of `*` we will write: `tc.*, db.objid, db.ra, db.dec, db.u, db.g, db.r, db.i, db.z`. This means we want to keep all columns from our local table (which is called tc) and add the columns from the remote table (which is called db). `FROM sdss.sources` specifies the table of the TAP service we want to query. The next lines are the query conditions. In our example we use the ADQL built in functions that defines the upload join. This means we define a circle around every object from our local table, and the database sell check if any data set of the sdss table lies within any of these circles. A little confusing may be the `1=CONTAINS`. This is due to the boolean result returned by the function `CONTAINS`. A result of `1` means **`true`** whereas `0` means **`false`**. `POINT` expresses a point, the right ascension and the declination in degrees. Our query takes the coordinates from the table using the columns `ra` and `dec`. Analogously `CIRCLE` expresses a cone in space, taking `ra` and `decl` from our local table. The last entry defines the radius of the circle in an angle in decimal degrees. The whole query looks like this:

```
SELECT
   TOP 1000
   tc.*, db.objid, db.ra, db.dec, db.u, db.g, db.r, db.i, db.z
   FROM sdssdr7.sources AS db
```

```
  JOIN TAP_UPLOAD.t1 AS tc
  ON 1=CONTAINS(POINT('ICRS', db.ra, db.dec),
                CIRCLE('ICRS', tc.ra, tc.decl, 10./3600.))
```

Click on Run Query and the result should come in rather fast. Again have a look at the new table. You see that we added a few columns, but it seemed that we lost some rows. This is due to the query: the database did only return those data records that matched our conditions. Not matching records are discarded.

ADQL

ADQL (Astronomical Data Query Language) is a query language based on a subset of SQL used to query tabular data in TAP services. The ADQL standard defines a common subset of SQL that can be used to access data in all the common relational database platforms used in astronomy. This enables users to query multiple services using the same query without having to worry about the specific dialect of the relational database platform hosting the data.

Exercise: You can control the ADQL JOIN behaviour by using the alternatives `LEFT OUTER JOIN`, `RIGHT OUTER JOIN` or `FULL OUTER JOIN`. The default command is `INNER JOIN`. Play around with these options, they might become usefull in future.

3 PyVO and ObsTAP

PyVO is the Python API to the VO standards. You can use it for data discovery and data access in the same way as you would use VO clients like Topcat or Aladin. But you can also use it embedded in your own code, so that you may access data remotely from an automated script. We will show with three simple (and one not so simple) scripts how you can use PyVO in the multimessenger context.

▷ **4** *Example 1: PyVO and TAP –*

Have a brief look at the few lines in example1.py. Note, that two lines are necessary to supress warnings – services and clients often are a bit off, especially in regard of recent standards. They still work, but warnings will be raised. We don't want to pretend that everything in the VO is perfect, but within a tutorial, warnings might be confusing. If you are curious what's going on, simply comment the according lines and run the script. The script performs a very simple query on the GAVO obscore service. Have a look how the service object is built by `pyvo.dal.TAPService()`. This is the convention within PyVO: `pyvo.dal.SERVICETYPE(parameters)`. The actual search is performed with the service object method `search`. This will send the query to the server, and the last line saves the resulting VOTABLE into the same folder as the script.

▷ **5** *Example 2: PyVO and SCS –*

This example shows analogously the Simple Cone Search from PyVO. The service object is built following the convention introduced in the last tutorial step, and as you see, we again use the service provided by km3net to obtain a list of neutrino events. Here we chose a much smaller radius though. We want to keep the following steps fast. Note the last lines of the script: we save the result to a local file, but we also use SAMP to broadcast the result to topcat. To make this work, you need to make sure that topcat is running before you run the script. You will find the result as a table in topcat now.

SAMP

SAMP (Simple Application Messaging Protocol) https://ivoa.net/documents/SAMP/ enables astronomy applications on the same desktop or laptop machine to share data with each other. It also enables applications to notify each other about what data points a user has selected, enabling the two applications to coordinate their display and selection views.

▷ **6** *Example 3: PyVO and Obscore –*

Example now is a bit closer to the real world. The script will make use of the neutrino data we get from example 2, which we saved locally to the hard drive. So we take different steps from here: we load the data from the folder exampl2. Note that we have a fallback exception in the code, in case something went wrong during the last tutorial step. We also introduce a longer TAP query. From the first part of the tutorial you are familiar with the TAP-upload already. But here we are comparing our local data with a special table on the TAP service: ivoa.obscore. A service providing this table follows a the special standard Obscore, which is dedicated to observational data. Each obscore-service provides this table with exactly defined columns which enables the users to use the same query repeatedly on each obscore-service. Of course there is no garantuee that one will receive any results, but it's making Searches all across the VO possible, and comparable easy. Here we query the gavo dc. only, and sent the results to Topcat.

▷ **7** *Topcat, SAMP and Aladin –*

Look at the results in Topcat. We will have a list of 50 images. Note: it's a table that does not contain the actual images, but urls to the images. Topcat

ObsCore

ObsCore `https://ivoa.net/documents/ObsCore/` Is a common data model for describing astronomical observations. It defines the core components of metadata needed to discover what observational data is available from a service. If a service advertises itself as an ObsTAP service, it means that it provides this standard view of metadata about the observational data. This means that a user can build an ADQL query based on the ObsCore data table and apply the same query to all the ObsTAP services.

can not deal with these links. Therefore, start Aladin, then go back to Topcat and use SAMP to sent the data to Aladin. In Aladin you now find this table and can download the fits files to your local machine. Hover over the rows to see the coverage of the images in relation to the position on the sky you were searching. Scroll a bit left and click on the buttons in the column ”Preview” to download smaller Versions of the images in Aladin.

4 Acknowledgements

The Authors thank ESCAPE WP4 ”CEVO” for making this contribution to ADASS 2020 possible.

Dave Morris works as a Research Software Engineer at the Royal Observatory, Edinburgh (ROE)

Hendrik Heinl works as Ingénieur d'études at the Centre de Données astronomiques de Strasbourg (CDS)

Markus Demleitner works as postdoc at the Astronomisches Rechen-Institut, Universität Heidelberg (ARI)

5 Standards

ADQL

Obscore

ObsLocTAP

SCS

SAMP

VOTable

6 References

Demleitner, M. ADQL-Course

Demleitner, M., Becker, S. PyVO Documentation

Fernique, P. Aladin Documentation

Taylor, M. TOPCAT Documentation