Astronomical Data Analysis Software and Systems XXIX
ASP Conference Series, Vol. 527
R. Pizzo, E.R. Deul, J-D. Mol, J. de Plaa and H. Verkouter, eds.
©2020 Astronomical Society of the Pacific

Implementing, with Python and PostgreSQL, Virtual Observatory Services for Publishing Survey Data from the Observatorio Astrofísico de Javalambre

Javier Hernández and Tamara Civera

CEFCA - Centro de Estudios de Física del Cosmos de Aragón, Teruel, Aragón, Spain

Abstract. The Centro de Estudios de Física del Cosmos de Aragón (CEFCA) is carrying out two large area multiband photometric sky surveys, J-PLUS and J-PAS, from the Observatorio Astrofísico de Javalambre (OAJ, Teruel, Spain) covering the entire optical spectrum using narrow and broad band filters. As an effort to make the data public, we offer Virtual Observatory (VO) compliant services to make the access to the data more versatile through the multiple VO compliant existent tools. For example, catalogues are offered through TAP protocol and images can be searched and downloaded using the SIAP protocol. This contribution summarizes why we decided to make our own implementation, the process we followed to choose which services to implement according to the kind of data generated by the survey, why Python and PostgreSQL were chosen and, finally, the lessons learned.

1. J-PLUS and J-PAS Data Publication

As large area multiband photometric sky surveys, J-PLUS[1] and J-PAS[2] have images, catalogue data, and photo-redshift computations as its main output artifacts. The final data release will contain hundreds of thousands images (about 4000 pointings in 59 filters for J-PAS) with tens of thousands of objects detected in each image. Besides single image catalogues, the main catalogue is obtained detecting objects in a reference filter image (R filter) and measuring the object's fluxes in all of the other filters (using Sextractor *dual* mode), the result is a *photo-spectra* of each object detected in the reference filter.

The obvious decision to publish survey data was to create a custom web portal (Civera 2020), offering advanced tools for data search, visualization, and download. The portal includes a powerful sky navigator (where we can select an object and see it properties and photo-spectra), image search, cone search, object list search, and image catalogue download. But we realized that the tools were not appropriate for bulk data processing, automation, and user customization (for example catalogue files just include a fraction of the data because including all the properties would bloat the files), and also did not offer a standardised way of accessing the data.

[1]`http://www.j-plus.es/`
[2]`http://j-pas.org/`

Due to the success of IVOA[3] Virtual Observatory standards, and availability of tools supporting them, we decided to offer support for some of the services so astronomers can use existing tools and create scripts to operate with our data. There were already exiting open source software to publish data as IVOA services, like those offered by GAVO[4] and the Spanish VO[5], but they did not fulfil all our needs. For example we wanted integration with our portal, secure access for certain catalogue versions, and support for a few custom extensions like array data types in TAP. We saw that IVOA services are web based, and in general they are simple, so we decided to create our own implementation of the few services needed, and integrate them in our portal.

Our database includes information about images, object catalogues, and computed redshifts. We decided to implement the Simple Image Access (SIA) service for image search on top of it, that returns links to our download service of full FITS images, *cutouts*, or colour images. We also included Cone Search for each object table that included indexed coordinates, and Table Access Protocol (TAP) to allow execute ADQL queries on the tables storing information about images, filters, objects and different photo-redshifts computed using several tools.

In our web pages we included also certain IVOA functionality, we included support for Simple Application Messaging Protocol (SAMP) so for example the image search page includes a button that can send a found image to an open window of the Aladin[6] tool (which supports SAMP). We also included a web page for executing TAP asynchronous queries for users that do not want to use external tools.

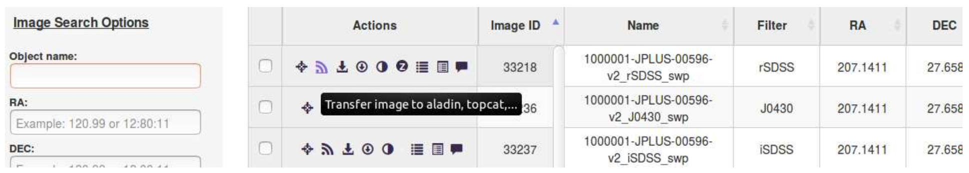


Image Search Options

Object name:

RA: Example: 120.99 or 12:80:11

DEC:

	Actions	Image ID	Name	Filter	RA	DEC
☐		33218	1000001-JPLUS-00596-v2_rSDSS_swp	rSDSS	207.1411	27.658
☐	Transfer image to aladin, topcat,...	36	1000001-JPLUS-00596-v2_J0430_swp	J0430	207.1411	27.658
☐		33237	1000001-JPLUS-00596-v2_iSDSS_swp	iSDSS	207.1411	27.658

Figure 1. SAMP image transfer from web portal.

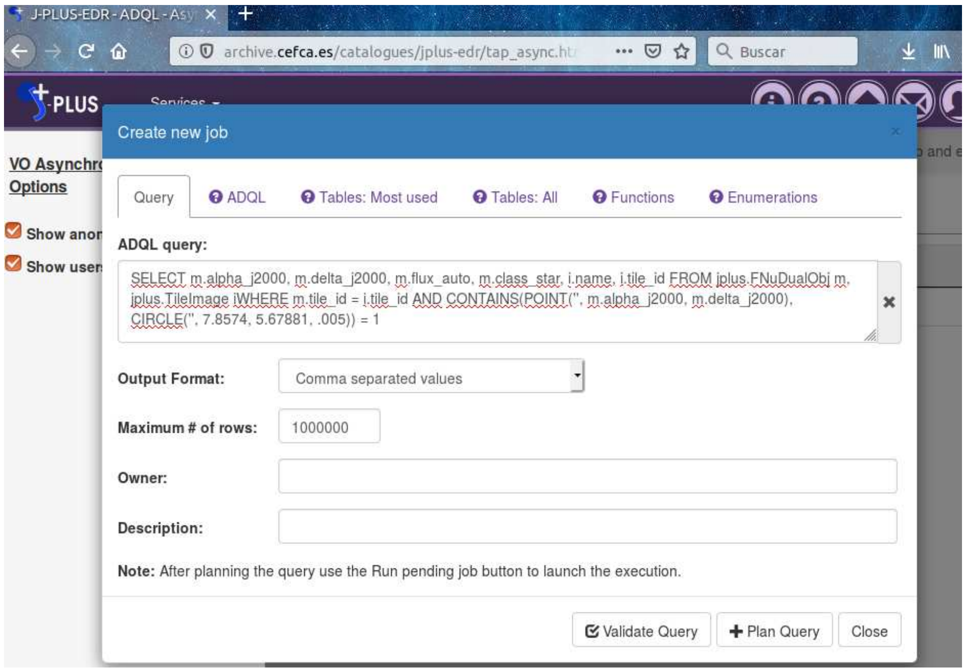
2. Implementation Tools

For the implementation the main parts to address were the storage for the catalogue data and the engine for web applications.

2.1. The Database Layer

The J-PLUS and J-PAS Catalogue final data is expected to occupy several terabytes so one of our priorities was to have a solid database server, known to manage terabyte sized databases, with also rich functionality like the possibility to add new functions and new data types. We also wanted an Open Source Relational database so we decided to use PostgreSQL. PostgreSQL has another very important feature for us, support for *array* data types. With flux and error measures for about sixteen apertures in 59 filters that

[3]`http://www.ivoa.net/`
[4]`https://www.g-vo.org/`
[5]`http://svo.cab.inta-csic.es/main/index.php`
[6]`http://aladin.u-strasbg.fr/aladin.gml`



ADQL query:

```
SELECT m.alpha_j2000, m.delta_j2000, m.flux_auto, m.class_star, i.name, i.tile_id FROM jplus.FNuDualObj m,
jplus.TileImage iWHERE m.tile_id = i.tile_id AND CONTAINS(POINT('', m.alpha_j2000, m.delta_j2000),
CIRCLE('', 7.8574, 5.67881, .005)) = 1
```

Output Format: Comma separated values

Maximum # of rows: 1000000

Owner:

Description:

Note: After planning the query use the Run pending job button to launch the execution.

Figure 2. Asynchronous TAP page.

give us, just for photometry and its error, about 1888 columns of data in the object table, a nightmare. We found that using a column of array type and assigning a position for each filter in the array we simplified the problem a lot, now users can easily obtain the whole photo-spectra or using an index to extract a certain band.

The ability to add new data types and complex functions in different programming languages was also very important for the implementation of TAP that includes a lot of math and geometric functions.

For cone search, and the use of similar functions in TAP, having a spatial index in the database is very important. Although PostgreSQL has some support for spatial indexes, we decided to use Healpix[7] for indexing object coordinates. Healpix indexes are integer values that are very compact to store in the database and allows the most reliable and efficient index type in PostgreSQL (B-Tree), additionally it allows users other uses like the creation of property maps (e.g. density maps).

2.2. The Web Layer

The Python programming language was already in use for the reduction pipelines, so we evaluated its suitability for creating web applications. With a little research we found that the Python WSGI specification with a lot of supporting servers (Apache mod_wsgi, Unicorn, uWSGI, ...) and frameworks (Pyramid, Flask, Django, ...) gave us an excellent support for the task. Another advantage is the large amount of astronomy, database access, and miscellaneous libraries available in Python.

[7]`https://healpix.jpl.nasa.gov/`

We opted to use the Pyramid framework that allows to easily map a Python function to a web URL, to extract web parameters and access to url segments. Also the framework allows different forms to generate the response, like using templates or returning a stored file.

3. Lessons learned

Regarding TAP, our initial idea was to pass the queries to the database directly, but for security reasons and SQL dialect translation, we opted to create a Python parser which needed a significant effort but offered interesting opportunities like table rename and hiding, inserting a *limit* expression to set the maximum output rows, or include what we called enumerations that allows to use names when referring the array position of filters in a magnitude/flux array column (for example 'jplus::rSDSS' for the 'r' filter position). We found that implementing the geometric functions in TAP is very complex due to the rich functionality that is defined, so at the moment only partial support for that functions exists (extra work is needed to support all coordinate frames or all figures operations).

The IVOA centres on public data so security access is not standardised. We have some catalogue versions with limited access for a certain period of time so we need registration and authentication, but some VO tools do not offer support for authentication or it is not documented. We finally achieved to support authentication for some tools like Topcat using Basic HTTP authentication.

Performance is always something that you have improve at some point. The large number of objects in the catalogue databases makes that performance of functions in the database very important. We initially implemented the needed functions for ADQL in Python but later we moved to a C implementation because it is ten times faster. A TAP query can take some minutes to execute so executing concurrently them is mandatory. Python threads was our first selection, but Python threads have some blocking issues. Fortunately the Python *multiprocessing* package has a similar API ands lacks that locking problems, so migration was quick.

Acknowledgments. Fondo de Inversiones de Teruel, PGC 2018-097585-B-C21 (MCIU / AEI / FEDER, UE), Aragón Research Group E16_17R (Gobierno de Aragón, Programa Operativo FEDER Aragón 2014-2020).

References

Civera, T. 2020, in ADASS XXIX, edited by R. Pizzo, E. Deul, J.-D. Mol, J. de Plaa, & H. Verkouter (San Francisco: ASP), vol. 527 of ASP Conf. Ser., 101