1. PyVO!

Markus Demleitner *(msdemlei@ari.uni-heidelberg.de)*
Hendrik Heinl *(heinl@ari.uni-heidelberg.de)*

Agenda

- Introduction to PyVO
- Synchronous TAP queries on three services
- Asynchronous TAP queries and SED selection
- Search and receive spectra
- Get PyVO

The full source code for the programs discussed here is also available as an attachment in this pdf. One way to retrieve them is to get pdftk (there's packages for it for Debian-derived systems), run `pdftk pyvo.pdf unpack_files`. Other PDF tools may also support attachments; in Adobe's proprietary Acrobat Reader 9, for instance, attachments can be retrieved through the paperclip icon in the lower left corner.

2. What's PyVO?

Astropy affiliated package for accessing Virtual Observatory data and services.

PyVO provides APIs for:

- Simple Image Access (SIA)
- Simple Spectral Access Protocol (SSAP)
- Simple Cone Search (SCS)
- Simple Line Access Protocol (SLAP)
- Table Access Protocol (TAP)*
- Simple Application Messaging Protocol (SAMP)*

3. The Use Case

Integrate results from potentially lots of different sources.

The example: Take photometry from three catalogs from three services using TAP.

Use different methods:

- Synchronous Queries + SAMP
- Asynchronous Queries combined with „your“ code
- Search for Spectra

```
QUERIES = [
  ("twomass", "http://dc.zah.uni-heidelberg.de/tap",
    """SELECT TOP 1000000 raj2000, dej2000, jmag, hmag, kmag
        FROM twomass.data
        WHERE 1=CONTAINS(
                    POINT('ICRS', raj2000, dej2000),
                    CIRCLE('ICRS', {ra}, {dec}, {radius}))"""),
  ("allwise", "http://tapvizier.u-strasbg.fr/TAPVizieR/tap",
    """SELECT raj2000, dej2000, w1mag, w2mag, w3mag, w4mag
        FROM "II/328/allwise"
        WHERE 1=CONTAINS(
                    POINT('ICRS', raj2000, dej2000),
                    CIRCLE('ICRS', {ra}, {dec}, {radius}))"""),
  ("sdss", "http://gea.esac.esa.int/tap-server/tap",
    """SELECT ra, dec, u_mag, g_mag, r_mag, i_mag, z_mag
        FROM gaiadr1.sdssdr9_original_valid
        WHERE 1=CONTAINS(
                    POINT('ICRS', ra, dec),
                    CIRCLE('ICRS', {ra}, {dec}, {radius}))"""),]
```

Fig. 1

```
with vohelper.SAMP_conn() as conn:
  topcat_id = vohelper.find_client(conn, "topcat")

  for short_name, access_url, query in QUERIES:
    service = pyvo.dal.TAPService(access_url)
    result = service.run_sync(query.format(**locals()), maxrec=90000)
    vohelper.send_table_to(conn, topcat_id, result.table, short_name)
```

Fig. 2

4. Synchronous Queries!

(cf. Fig. 1)

5. Run Queries and Use SAMP

Running the queries and sending the results to TOPCAT is actually close to trivial with a bit of support code:

(cf. Fig. 2)

∩ Note: „with“ does „context managers“; these are a python trick to ensure programs clean up after themselves („maintain external invariants“). Use them whenever you can, your code will become much more robust.

vohelper is a bunch of python functions you'll find linked from at the end of this document. The ∪ same goes for all the code we're running here for re-use.

```
jobs = set()
for short_name, access_url, query in QUERIES:
  job = pyvo.dal.TAPService(access_url).submit_job(
    query.format(**locals()), maxrec=9000000)
  job.run()
  jobs.add((short_name, job))
```

Fig. 3

6. Asynchronous Jobs and „your“ code

When doing a lot of queries or long-running queries, it may be a good move to run them asynchronously and in parallel. We will

- run Asynchronoues Jobs on three services,
- use your code to cluster the received positions,
- build SEDs,
- plot SEDs and select objects of interest.

Given async is a fairly complex pattern and this kind of orchestration is fairly advanced, the actual code is quite simple.

7. Asynchronous Jobs continued

Create jobs, start them, memorize short names and jobs:

(cf. Fig. 3)

```
with vohelper.SAMP_conn() as conn:
  topcat_id = vohelper.find_client(conn, "topcat")

  while jobs:
    for short_name, job in list(jobs):
      print short_name, job.phase
      if job.phase not in ('QUEUED', 'EXECUTING'):
        jobs.remove((short_name, job))
        vohelper.send_table_to(
          conn,
          topcat_id,
          job.fetch_result().table,
          short_name)
        job.delete()

    time.sleep(0.5)
```

Fig. 4

8. Asynchronous Jobs continued

Then poll the status of the jobs until all are done:

(cf. Fig. 4)

∩ We're using a bit larger cones now. Nnote that for VizieR, right now there are relatively tight limitations on the result size.

The warnings are caused by botched metadata. We'll have to fix it later in our code. Whenever you notice bugs like this, please complain to the providers.

∪

9. Combine with „your" Code

The cool thing about doing this in python is that you can easily do your own logic now.

Here: Cluster our data by position and use the photometry from the different tables for our SEDs. Then, display the curves and let the user interactively select „interesting" cases.

∩ Prerequisite for crossmatching: make „points". In Astropy, you can do vector-computing, numpy- and matlab-style. Plus, thanks to UCDs, you don't have to figure out the names of the positional
∪ columns manually:

```
def get_coordinates_for_table(table):
  ra_column = vohelper.get_name_for_ucd(
    "pos.eq.ra;meta.main", table)
  dec_column = vohelper.get_name_for_ucd(
    "pos.eq.dec;meta.main", table)

  # fix broken metadata (sigh)
  if table[ra_column].unit=="Angle[deg]":
    table[ra_column].unit = "deg"
  if table[dec_column].unit=="Angle[deg]":
    table[dec_column].unit = "deg"

  return coordinates.SkyCoord(
    work_around_vizast_bug(table[ra_column]),
    work_around_vizast_bug(table[dec_column]))
```

Fig. 5

```
  for row in rows:
    for index, col in enumerate(row):
      name = row.columns[index].name
      ucd = work_around_sdss_ucd_bug(name,
        row.columns[index].meta.get("ucd", "").lower())

      if ucd.startswith("phot.mag"):
        col = force_scalar(col)   # workaround for broken VizieR
        if ucd in UCD_TO_WL:
          phots.append((UCD_TO_WL[ucd], col))
      elif ucd=="pos.eq.dec;meta.main":
        pos[1] = force_scalar(col)
      elif ucd=="pos.eq.ra;meta.main":
        pos[0] = force_scalar(col)

  return tuple(pos), sorted(phots)
```

Fig. 6

10. Cluster by Position

Get coordinates through UCDs:

(cf. Fig. 5)

You can see how annoying it is if people don't follow standards, in this case in the units – ugly, special-cased code. Complain to the data providers if you see this kind of thing („virtuous cycle" where people increasingly see the right thing and therefore do the right thing)

11. Build SEDs

We then compute (table, index) pairs of rows that need to be combined (use your favourite clusterer instead of the ad-hoc n-way crossmatch we've included in vohelper). When we have those, we build our SSEDß from a sequence of rows that should belong together.

Get magnitudes through UCDs (and workaround):

(cf. Fig. 6)

Again, workarounds because people strech standards: force_scalar is because VizieR has arraysize=1 for scalars (bug reported, being fixed) and astropy interprets that as producing arrays (nobody else does, this is a bug in progress).

```
UCD_TO_WL = {
  "phot.mag;em.opt.u": 3.5e-7,
  "phot.mag;em.opt.b": 4.5e-7,
  "phot.mag;em.opt.v": 5.5e-7,
  "phot.mag;em.opt.r": 6.75e-7,
  "phot.mag;em.opt.i": 8.75e-7,
  "phot.mag;em.ir.j": 1.25e-6,
  "phot.mag;em.ir.h": 1.75e-6,
  "phot.mag;em.ir.k": 2.2e-6,
  "phot.mag;em.ir.3-4um": 3.5e-6,
  "phot.mag;em.ir.4-8um": 6e-6,
  "phot.mag;em.ir.8-15um": 11.5e-6,
  "phot.mag;em.ir.15-30um": 22.5e-6,
}
```

Fig. 7

Also SDSS has wrong and bad UCDs, so we have to manually translate them to proper UCDs from their column names. Not reported, this is something astronomers need to complain about.

12. Build SEDs continued

Also, no good, interoperable way to precisely annotate broadband photometry („photometry DM“; DM efforts in the VO always take a lot longer than one can imagine). Hence, we're happy with a rough qualification:

Mapping UCDs to band width:

(cf. Fig. 7)

You'll see that we're mapping both SDSS i and z to em.opt.i. Yeah, sucks. Of course, you can improve on this by using domain knowledge about SDSS, but that's not our point here. We, we want interoperable annotation of photometric bands for all/most resources.

```
for pos, phots in seds:
  to_plot = np.array(phots)
  plt.semilogx(to_plot[:,0], to_plot[:,1], '-')
  plt.ylim(reversed(plt.ylim()))
  plt.ylabel("Mag",fontsize=15)
  plt.xlabel("Wavelength", fontsize=15)
  plt.show(block=False)
  selection = raw_input("s)elect SED, q)uit, enter for next? ")
  if selection=="q":
    break
  if selection=="s":
    selected.append(pos)
  plt.cla()

return selected
```

Fig. 8

```
t = table.Table()
t.add_column(table.Column(name='ra',
  data=selected[:,0],
  unit=u.degree,
  meta={"ucd": "pos.eq.ra;meta.main"}))
t.add_column(table.Column(name='dec',
  data=selected[:,1],
  unit=u.degree,
  meta={"ucd": "pos.eq.dec;meta.main"}))
with open("selected_positions.vot", "w") as f:
  t.write(output=f, format="votable")
```

Fig. 9

13. Plot SEDs

Use matplotlib to select SEDs of interest.

(cf. Fig. 8)

14. Save selected Positions

Make a votable from the selected positions.

(cf. Fig. 9)

Don't try to save time by skipping proper annotation with units and UCDs – you'll regret it later, and your colleagues will, too. P

15. Overview

- Little more than 200 lines of code
- .. of which almost half are workarounds for interoparability bugs (sigh).

Imagine how great things will be when people will have embraced interoperable services and annotations.

```
# use raw RegTAP until pyVO registry is up to the task
result = pyvo.dal.TAPService("http://reg.g-vo.org/tap"
  ).run_sync("""
  SELECT DISTINCT access_url AS url
  FROM rr.interface
  NATURAL JOIN rr.capability
  NATURAL JOIN rr.res_detail
  WHERE standard_id='ivo://ivoa.net/std/tap'
     AND intf_type='vs:paramhttp'
     AND detail_xpath='/capability/dataModel/@ivo-id'
     AND 1=ivo_nocasematch(detail_value,
       'ivo://ivoa.net/std/obscore%')""")

for url in result["url"]:
  yield url
```

Fig. 10

```
for access_url in iter_obscore_urls():
  sys.stdout.write("Querying {} ...".format(access_url))
  sys.stdout.flush()

  spectra.extend(
    get_spectra_for_table(access_url, pois, radius, n_samp))
  sys.stdout.write(" done.\n")

with vohelper.SAMP_conn() as conn:
  target_id = vohelper.find_client(conn, "splat")

  for ds_name, access_url in spectra:
    print("Opening {}..."%access_url)
    vohelper.send_spectrum_to(conn, target_id, access_url, ds_name)
```

Fig. 11

16. Looking for Spectra

SSAP is the VO protocol to access spectra. Hence, it only lets you access one object at a time, which is kind of tedious. Let's use `obscore` instead. Luckily `obscore` is just TAP with a special table structure. It's good to find spectra, cubes, timeseries, etc.

- Search for `obscore` services
- Use TAP upload to search to collect spectra
- Send spectra to SPLAT

You can query all Obscore services (essentially) in a uniform way.

17. Query the registry

First query the Registry for all Obscore services:
(cf. Fig. 10)

18. Collect Spectra

Collect spectra and send them to SPLAT to display and investigate
(cf. Fig. 11)

```
result = vohelper.run_sync_resilient(svc,
  """SELECT TOP {samplesize} obs_publisher_did, access_url
      FROM ivoa.obscore
      JOIN TAP_UPLOAD.pois AS up
      ON 1=CONTAINS(
        POINT('ICRS', s_ra, s_dec),
        CIRCLE('ICRS', up.{ra_column_name}, up.{dec_column_name}, {radius}))
      WHERE dataproduct_type='spectrum'
      """.format(**locals()),
      # add more constraints (spectral region, resolution,.. here)
      uploads = {"pois": ('inline', pois)})

if result is None:
  return

for row in result.table:
  yield unicode(row[0]), unicode(row[1])
```

Fig. 12

19. Collect Spectra continued

The TAP upload query:

(cf. Fig. 12)

20. How to get PyVO

`pip install pyvo`

Credits go to:
Markus Demleitner & Stefan Becker (both ARI)
Thank you for your attention.