Fig. 1

Fig. 2

1. Astropy and the VO

(cf. Fig. 1)

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

(cf. Fig. 2)

- PyVO and a single image server
- The power of standards: all-VO
- More power: all-VO plus SAMP
- Fun action

This is supposed to be an appetiser (and, to some extent, quarry for sample code) for doing fun and profitable stuff with VAO's astropy extension for VO access (called pyVO) as well as astropy's built-in VO capabilities.

PyVO is available at http://dev.usvao.org/vao/wiki/Products/PyVO

2. Query a single SIAP service

SIAP is the VO's protocol to access image servers. Here's how you'd do a search at multiple positions with date constraints (that's already nontrivial with graphical clients):

```
svc = sia.SIAService(ACCESS_URL)
for pos in [
    (10, 20),
    (240, -10),
    (45, 85)]:
  images = svc.search(pos, (0.5, 0.5), verbosity=2)
  for rec_no, match in enumerate(images):

    if not DATE_MIN<match["dateObs"]<DATE_MAX:
      continue

    match.cachedataset()
```

Note how you don't actually have to know anything about the service except its access URL (that you might get from a registry). Since it uses a standard protocol, pyVO knows enough to be able to, in this case, retrieve the file and (mostly) give it a reasonable name.

Here's how this would look like in a complete program:

```
"""
A trivial example for how to operate a SIAP service from pyVO:
find images from a list of positions and by date.


Get ACCESS_URL from, e.g., http://dc.g-vo.org/WIRR.
"""

from astropy.time import Time
from pyvo.dal import sia

ACCESS_URL = "http://dc.g-vo.org/lswscans/res/positions/siap/siap.xml?"
DATE_MIN = Time("1902-01-01", scale="tt").mjd
DATE_MAX = Time("1922-12-31T23:59:59", scale="tt").mjd


def main():
  svc = sia.SIAService(ACCESS_URL)
  for pos in [
      (10, 20),
      (240, -10),
      (45, 85)]:
    images = svc.search(pos, (0.5, 0.5), verbosity=2)
    for rec_no, match in enumerate(images):

      if not DATE_MIN<match["dateObs"]<DATE_MAX:
        continue

      print "%s Get? "%match.title,
      if raw_input().strip().lower().startswith("y"):
        match.cachedataset()


if __name__=="__main__":
  main()
```

3. And now all-VO

The nice thing about standard services: Handle one, and you get them all. So, let's add a query to the Registry and run our query all over the VO –

```
for svc in registry.search(servicetype="image"):
  try:
    search_one_service(svc.accessurl, image_sender)
  except Exception:
    import traceback; traceback.print_exc()
```

The exception catcher is there since not all services claiming to be standards-compliant actually are. It doesn't hurt to complain to the service operators if a service you're interested in behaves weirdly – sometimes the operators haven't noticed it's broken or just broke.

Again, here's how this would look in a full program:

```
"""
A little script doing an all-VO SIAP query for some positions and a date
range.
"""

from astropy.time import Time
from pyvo.dal import sia
from pyvo import registry

DATE_MIN = Time("1902-01-01", scale="tt").mjd
DATE_MAX = Time("1922-12-31T23:59:59", scale="tt").mjd


def search_one_service(access_url):
  print "Now querying", access_url
  svc = sia.SIAService(access_url)
  for pos in [
      (10, 20)]:
    images = svc.search(pos, (0.5, 0.5), verbosity=2)
    dateName = images.fieldname_with_ucd("VOX:Image_MJDateObs")
    if dateName is None:
      return

    for rec_no, match in enumerate(images):

      if not DATE_MIN<match[dateName]<DATE_MAX:
        continue

      print "%s Get? "%match.title,
      if raw_input().strip().lower().startswith("y"):
        match.cachedataset()


def main():
  for svc in registry.search(servicetype="image"):
    try:
      search_one_service(svc.accessurl)
    except:
      import traceback; traceback.print_exc()


if __name__=="__main__":
  main()
```

4. Add SAMP Magic

Let's send over images we're interested in via SAMP:

```
  def broadcast_image(self, image_url):
    message = {
      "samp.mtype": "table.load.fits",
      "samp.params": {
        "url": image_url,
        "name": NAME_TEMPLATE%self.image_count,
      }}
    self.client.notify_all(message)

...
        image_sender.broadcast_image(match.acref)
...
  with SAMP_conn() as conn:
    for svc in registry.search(servicetype="image"):
      search_one_service(svc.accessurl, image_sender)
```

SAMP-enabling programs may not come quite natural to people that so far have mainly written fairly linear science code. But here it's no big deal except for the necessity to manage the connection to the SAMP hub, which is taken care of by a hand-crafted context manager in our code.

Also, the way arguments are passed between SAMP services may seem a bit funky – but think of the mtype as the function name, and passing arguments in dictionaries instead of tuples isn't that far-fetched, either.

The full program:

```
"""
A little script doing an all-VO SIAP query for some positions and a date
range; the results can be sent to SAMP clients.
"""

import contextlib

from astropy.time import Time
from astropy.vo.samp import SAMPIntegratedClient
from pyvo.dal import sia
from pyvo import registry


DATE_MIN = Time("1990-01-01", scale="tt").mjd
DATE_MAX = Time("2005-12-31T23:59:59", scale="tt").mjd
NAME_TEMPLATE = "global-%03d"


class ImageSender(object):
  """A SAMP handler broadcasting images.

  Construct this class with a SAMP connection and call broadcast_image
  to have SAMP clients load the images.
  """
  def __init__(self, client):
    self.client = client
    self.image_count = 0

  def broadcast_image(self, image_url):
    message = {
      "samp.mtype": "table.load.fits",
      "samp.params": {
        "url": image_url,
        "name": NAME_TEMPLATE%self.image_count,
      }}
    self.client.notify_all(message)
```

```
def search_one_service(access_url, image_sender):
  print "Now querying", access_url
  svc = sia.SIAService(access_url)
  for pos in [
      (10, 20)]:
    images = svc.search(pos, (0.5, 0.5), verbosity=2)
    dateName = images.fieldname_with_ucd("VOX:Image_MJDateObs")
    if dateName is None:
      return

    for rec_no, match in enumerate(images):

      if not DATE_MIN<match[dateName]<DATE_MAX:
        continue

      if not match.format.endswith("fits"):
        continue

      print "%s Show? "%match.title,
      if raw_input().strip().lower().startswith("y"):
        image_sender.broadcast_image(match.acref)


@contextlib.contextmanager
def SAMP_conn():
  """a context manager to give the controlled block a SAMP connection.

  The program will disconnect as the controlled block is exited.
  """
  client = SAMPIntegratedClient(name="smpsmp",
    description="A sample for the use of SAMP")
  client.connect()
  try:
    yield client
  finally:
    client.disconnect()


def main():
  with SAMP_conn() as conn:
    image_sender = ImageSender(conn)
    for svc in registry.search(servicetype="image"):
      try:
        search_one_service(svc.accessurl, image_sender)
      except Exception:
        import traceback; traceback.print_exc()


if __name__=="__main__":
  main()
```

5. Higher Magic

Let's say you're debugging your pipeline and want to manually inspect "weird" objects by checking what a set of other catalogs have on them.

So, a script would have to notice when a table row is selected:

```
class VicinitySearcher(object):
  def __init__(self, client):
    self.client.bind_receive_call(
      "table.load.votable", self.load_VOTable)
    self.client.bind_receive_notification("table.highlight.row",
      self.handle_selection)

  def load_VOTable(self, private_key, ...
    self.cur_table = Table.read(params['url'])
    self.cur_id = params["table-id"]
    self.client.reply(msg_id,
      {"samp.status": "samp.ok", "samp.result": {}})

  def handle_selection(self, private_key, ...
    if params["table-id"]!=self.cur_id:
      return
    table_index = int(params["row"])
    response = self.make_response_table(table_index)
    if response is not None:
      self.send_table(response)
```

Essentially, we're telling the SAMP hub we're interested in highlights of rows in tables. However, as multiple tables may be around, in our handler we make sure we only do something if we're dealing with the last table we got sent.

6. United Services of the VO

If we query multiple services and want to get a single table back, we must combine their results. The Cone Search standard and UCDs help:

```
def make_response_table(self, table_index):
  ras, decs, pmras, pmdecs, svcs = [], [], [], [], []
  for service in self.services:
    cone_result = service.search((ra,dec),
      self.vicinity_size/3600.)
    ras.extend(cone_result.getcolumn(
      cone_result.fieldname_with_ucd("POS_EQ_RA_MAIN")))
    decs.extend(cone_result.getcolumn(
      cone_result.fieldname_with_ucd("POS_EQ_DEC_MAIN")))
    pmra_name = cone_result.fieldname_with_ucd("pos.pm;pos.eq.ra")
    if pmra_name:
      pmras.extend(cone_result.getcolumn(pmra_name))
    else:
      pmras.extend([None]*nrecs)
```

The UCDs (those for RA and DEC look odd because the cone search standard is so ancient) help us retrieve columns by physics (what's the RA? what's the PM in RA?).

Incidentally, the units on the PMs are not defined by the standard, and to actually unify them we'd need to do unit conversion. That's standard astropy, though.

So, finally, here's the full program with a usage scenario in the module docstring:

```
"""
A quick example showing astropy and pyvo working hand in hand with the
rest of the VO

This program expects Aladin to run.  It then waits for a tables to be sent,
and when a row is selected, it will search some (SERVICE_META) cone
search services.  The results are joined and sent to aladin with
```

```
positions, proper motions, and source.

Sample use:

(1) start TOPCAT, aladin, then python smpsmp.py
(2) in TOPCAT, open VO/Cone Search, look for "transitional YSOs"
(3) select the Magnier+ 1999 service, make RA and DEC 0, SR 180, "ok"
(4) broadcast table
(5) in Aladin, pan and zoom until you have a catalog object in a FoV of
    an arcminute or so
(6) hover over the object to pull in the potential matches
(7) select the items to see the catalog entries.
"""

import contextlib
import os
import tempfile
import traceback

from astropy.vo.samp import SAMPIntegratedClient
from astropy.table import Table
from pyvo.dal import scs

SERVICE_META = [
  ("PPMXL", "http://dc.zah.uni-heidelberg.de/ppmxl/q/cone/scs.xml?"),
  ("2MASS", "http://dc.zah.uni-heidelberg.de/2mass/res/2mass/q/scs.xml?"),
  ("UCAC4", "http://dc.zah.uni-heidelberg.de/ucac4/q/s/scs.xml?")]


@contextlib.contextmanager
def debug():
  """dumps tracebacks of exceptions within the controlled block.

  That's useful here as astropy's SAMP silently swallows exceptions
  in handlers.
  """
  try:
    yield
  except:
    traceback.print_exc()
    raise


@contextlib.contextmanager
def samp_accessible(astropy_table):
  """a context manager making astroy_table available under a (file)
  URL for the controlled section.

  This is useful with uploads.
  """
  handle, f_name = tempfile.mkstemp(suffix=".xml")
  with os.fdopen(handle, "w") as f:
    astropy_table.write(output=f,
      format="votable")
  try:
    yield "file://"+f_name
  finally:
    os.unlink(f_name)


def get_name_for_ucd(ucd, table):
  """returns the name of a column having ucd in table.

  If there are multiple such columns, a random one is returned.  If there
  are none, a key error is raised.
  """
  ucd = ucd.lower()
```

```
  for col in table.columns.values():
    if col.meta.get("ucd").lower()==ucd:
      return col.name
  raise KeyError(ucd)


class VicinitySearcher(object):
  """The SAMP handling class.

  This is where the action takes place: receiving VOTables, handling
  notifications of selected rows, querying the remote services.

  True, in a less one-off program this should be less god-like, and
  at least make_response_table shouln't be part of this.
  """
  vicinity_size = 30  # arcsec
  client_name = "Aladin" # samp.name of the client for the match table

  def __init__(self, client):
    self.client = client
    self.cur_table = self.cur_id = None
    self.dest_client = self.find_client(self.client_name)

    self.services = []
    for short_name, access_url in SERVICE_META:
      self.services.append(scs.SCSService(access_url))
      self.services[-1].my_tag = short_name

    self.client.bind_receive_call(
      "table.load.votable", self.load_VOTable)
    self.client.bind_receive_notification("table.highlight.row",
      self.handle_selection)

  def load_VOTable(self, private_key, sender_id, msg_id, mtype,
      params, extra):
    """the SAMP handler to load VOTables.

    (binding is done in the constructor)
    """
    self.cur_table = Table.read(params['url'])
    self.ra_name = get_name_for_ucd("POS_EQ_RA_MAIN", self.cur_table)
    self.dec_name = get_name_for_ucd("POS_EQ_DEC_MAIN", self.cur_table)
    self.cur_id = params["table-id"]

    self.client.reply(msg_id,
      {"samp.status": "samp.ok", "samp.result": {}})

  def handle_selection(self, private_key, sender_id, msg_id, mtype,
      params, extra):
    """the SAMP handler for a row selection in our current table.
    """
    with debug():
      if params["table-id"]!=self.cur_id:
        # doesn't concern us, not our table
        return
      table_index = int(params["row"])
      print "Row selected:", table_index
      response = self.make_response_table(table_index)

      if response is not None:
        self.send_table(response)

  def send_table(self, response_table):
    """sends the (astropy) response_table via SAMP.
    """
    with samp_accessible(response_table) as table_url:
```

```
    message = {
      "samp.mtype": "table.load.votable",
      "samp.params": {
        "url": table_url,
      }}
    self.client.call_and_wait(self.dest_client, message, "10")

def find_client(self, samp_name):
  """returns the SAMP id of the client with samp.name samp_name.

  This will raise a KeyError if the client is not on the hub.
  """
  for client_id in self.client.get_registered_clients():
    if self.client.get_metadata(client_id).get("samp.name")==samp_name:
      return client_id
  raise KeyError(samp_name)

def make_response_table(self, table_index):
  """returns an astropy table (or None) for the row table_index.

  This is essentially the "user code" that reacts on the incoming
  messages.
  """
  ra = self.cur_table[self.ra_name][table_index]
  dec = self.cur_table[self.dec_name][table_index]

  ras, decs, pmras, pmdecs, svcs = [], [], [], [], []
  for service in self.services:
    print "Querying ", service.my_tag
    cone_result = service.search((ra,dec),
      self.vicinity_size/3600.)
    # len(core_result) appears broken ATM
    nrecs = len(cone_result.getcolumn(
      cone_result.fieldname_with_ucd("POS_EQ_RA_MAIN")))

    ras.extend(cone_result.getcolumn(
      cone_result.fieldname_with_ucd("POS_EQ_RA_MAIN")))
    decs.extend(cone_result.getcolumn(
      cone_result.fieldname_with_ucd("POS_EQ_DEC_MAIN")))

    pmra_name = cone_result.fieldname_with_ucd("pos.pm;pos.eq.ra")
    if pmra_name:
      # TODO: check unit and covert if necessary
      pmras.extend(cone_result.getcolumn(pmra_name))
    else:
      pmras.extend([None]*nrecs)

    pmdec_name = cone_result.fieldname_with_ucd("pos.pm;pos.eq.dec")
    if pmdec_name:
      pmdecs.extend(cone_result.getcolumn(pmdec_name))
    else:
      pmdecs.extend([None]*nrecs)

    svcs.extend([service.my_tag]*nrecs)

  if not ras:
    return None
  else:
    print "Found %d matches"%len(ras)

  res = Table([ras, decs, pmras, pmdecs, svcs],
    names=("ra", "dec", "pmra", "pmdec", "svc"))
  res["ra"].unit = "deg"
  res["dec"].unit = "deg"
  res["pmra"].unit = "deg/yr"
  res["pmdec"].unit = "deg/yr"
  # TODO: enter UCDs in the meta dict
```

Fig. 3

```
    return res


@contextlib.contextmanager
def SAMP_conn():
  """a context manager to give the controlled block a SAMP connection.

  The program will disconnect as the controlled block is exited.
  """
  client = SAMPIntegratedClient(name="smpsmp",
    description="A sample for the use of SAMP")
  client.connect()
  try:
    yield client
  finally:
    client.disconnect()


def main():
  with SAMP_conn() as client:
    handler = VicinitySearcher(client)
    print "Listening.  Send me a table, hit return to exit."
    raw_input()


if __name__=="__main__":
  main()
```

7. The End

When riding home…

(cf. Fig. 3)

…take this with you: http://docs.g-vo.org/pyvo.pdf

To prevent formatting issues, the source code given 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.