Fig. 1

Fig. 2

Fig. 3

1. Solar System Data in DaCHS

(cf. Fig. 1)

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

(cf. Fig. 2)

- DaCHS is a VO publishing toolkit...
- ...that has built-in support for EPN-TAP...
- ...which helps publishing solar system data.

(cf. Fig. 3)

2. Publishing Data Using DaCHS

DaCHS is a package (based on python and twisted) for:

- describing data to be published
- ingesting raw data
- running the services
- and doing assorted management tasks on that

Essentially: Concentrate on what's special for your data rather than re-writing code for generic VO protocols.

3. A Quick Example

If you want to try it out, see

http://dachs-doc.readthedocs.io/tutorial.html#epn-tap

There's an archive there for a DaCHS service serving images of Lutetia:

```
$ ls -R
.:
README  bin/  data/  q.rd

./bin:
makePreview.py

./data:
README            W13463ID20F17.IMG
W58919ID20F31.IMG W11079ID20F15.IMG
W40177ID20F17.IMG
```

(slightly redacted)

4. RD 1: Resource Metadata

That's just like with any other VO resource:

```
<meta name="creationDate">2014-08-15T13:00:00</meta>
<meta name="source">ftp://psa.esac.esa.int/pub/mirror...</meta>
<meta name="description">Images from the Rosetta flyby at asteroid
  (21) Lutetia.  Very few of them.  That's because this is just
  a fairly dumb example for how to use DaCHS to publish to EPN-TAP.
</meta>
<meta name="creator">
  Sierks, H.
</meta>
...
```

This usually comprises further information like keywords, coverage, etc: Let people find your data in the Registry.

5. RD 2: Table Metadata

This defines the table(s) the service deals with. For EPN-TAP, a set of columns is predefined:

```
<table id="epn_core">
  <mixin
    spatial_frame_type="celestial"
    optional_columns="access_url access_format
     access_estsize thumbnail_url
     bib_reference publisher"
    >//epntap2#table-2_0</mixin>
  <mixin>//epntap2#localfile-2_0</mixin>
</table>
```

„Mixins“ is now DaCHS ensures table structure and metadata matches the expectations of certain protocols. They can have parameters, which encode global table properties.

For the epntap mixin, this is the spatial frame type (because it influences units and UCDs on lots of columns) and the names of non-mandatory columns you want in your particular table. There are quite a number here because you have image-like products.

There is a second mixin, localfile, which endows the table with extra columns needed for DaCHS-managed data products (as opposed to data products lying elsewhere); if DaCHS manages your files, you can have embargos, more or less automatic previews, and similar goodies.

You could give further, custom columns here in a straightforward way.

That's enough to get the standard schema with the standard metadata.

6. RD 3: Ingestion Rules

```
<data id="import">
  <property key="previewDir">previews</property>
  <sources pattern="data/*.IMG"/>
  <pdsGrammar>
    <rowfilter procDef="//products#define">
      <bind name="table">"\schema.epn_core"</bind>[...]

  <make table="epn_core">
    <rowmaker idmaps="*">
      <var key="exptime">float(@SR_ACQUIRE_OPTIONS[
          "EXPOSURE_DURATION"].split()[0])</var>

      <apply procDef="//epntap2#populate-2_0" name="fillepn">
        <bind key="access_format">"PDS"</bind>
        <bind key="c1min">@ra</bind> [...]

      <apply procDef="//epntap2#populate-localfile-2_0"/>

      <map key="bib_reference">("ftp://psa.esac.esa.int/pub/mirror/"
        "INTERNATIONAL-ROSETTA-MISSION")</map>
```

Ingestion rules give

- sources (where to get the raw files)
- a grammar (how to turn the raw files into semistructured data; here, that's for planetary data system files)
- a make element (how to turn the semistructured data into database tables)

The main complication usually is the last item. Here, you have to convert and map the messy stuff from (in this case) the headers to the types and names expected in the database. This is a mixture of simple mapping rules, a bit more complicated python expressions, and applys

(procedural specification with parameters). In this case, we're using special applys used for epntap, which let you perform the mapping with some sort of checklist of what you can and/or should give.

The property element is used with DaCHS's preview/thumbnail facility. Don't worry about it now.

7. RD 4: Service Definitions

In general, you'd now define the services exposing the data you've just given now. For EPN-TAP, you're just using the TAP service centrally defined by DaCHS.

Hence, no further work.

8. RD 5: Regression Tests

Good for your peace of mind: Add a regression test so you'll catch obvious problems after updates or other natural hazards:

```
  <regTest title="Lutetia access url resolved">
    <url parSet="TAP"
      QUERY="SELECT TOP 1 access_url FROM lutetia.epn_core"
      >/tap/sync</url>
    <code>
      row = self.getFirstVOTableRow()
      stuff = urllib.urlopen(row["access_url"]).read()
      self.failUnless('"OSIRIS - WIDE ANGLE CAMERA"' in stuff)
    </code>
  </regTest>
```

Essentially, you define a URL to retrieve in a way that what happens still is readable (we also take care to complete URLs so the tests don't depend on where RD runs). Then, you write a couple of lines of python, inspired by the pyUnit framework.

With this, after an update, you run `dachs test ALL`, and if this just runs, you've probably not hit anything life-critical.

9. Nice Touch: Previews

When DaCHS can't make previews by itself, you have to help it with a little program:

```
class PreviewMaker(processing.PreviewMaker):
  def getPreviewData(self, srcName):
    f = open_pds(srcName)
    [...]
    imageData = f.read(imageDataLength)
    pixels = numpy.fromstring(
      imageData, dtype=numpy.int16).reshape(imageWidth, imageHeight)
    pixels[pixels<0] = 0
    return imgtools.jpegFromNumpyArray(
      imgtools.scaleNumpyArray(pixels, 400))

if __name__=="__main__":
  processing.procmain(PreviewMaker, "lutetia/q", "import")
```

To build all previews (skipping those already built), just run `python bin/makePreview.py`.

10. Thanks

More info on DaCHS: http://soft.g-vo.org