Fig. 1

Fig. 2

1. Datalink: The GAVO perspective

(vgl. Fig. 1)

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

(vgl. Fig. 2)

- Datalink scope
- Intercessions
- Response Data model
- Pieces of responses
- Suggested Parameters
- (Service operators' view)
- (URLs to play with)

2. Datalink's Scope

The Heidelberg consensus appears to be: Datalink transports information on

- Links to ancillary or supplementary data (previews, rendered plots, errors,...) and to typed services (e.g., SSAP, SIAP) exposing the data
- Enough information to operate "cutout-type" services on the data.

"cutout-like" for us meant: including stuff GAVO has been pushing in the context of our SSAP getData efforts, e.g., recalibration.

3. Intercessions

I implore you to consider the following intercessions – most of this is the result of lessons learnt implementing SSAP client and server side:

1. Each individual service must fully describe its interface in machine-readable way, easily accessible. This means: No underspecified mini-languages in the parameters, no deferring of interface descriptions to registry extensions.
2. No MAY and SHOULD (unless they're part of the interface description). That's actually a corollary to (1)
3. Do not overspecify – Cutout and other subsetting, header selection and similar are very generic operations. Let's not artificially restrict our scope unless we're actually significantly gaining functionality for the special case.
4. No specification without implementation
5. No ad-hoc mogrifications of STC. There's an STC data model, and there's a quasi-standard for how to use it in VOTables, so stop improvising.
6. Please use UCDs
7. Keep the number of requests low. That's not quite as important as the others, but my SSAP use case, for example, would be severely damaged if the initial datalink response didn't contain the entire interface description.

4. Response Data Model I

The basics of the data model are pretty clear:

- a sequence of plain "links" having at least attributes url, content-type, relation-type
- a sequence of service description giving an access URL, input parameters, and maybe additional service metadata

We now have a language to express such data models. A first draft is in volute:

trunk/projects/dm/vo-dml/datalink

Please contribute!

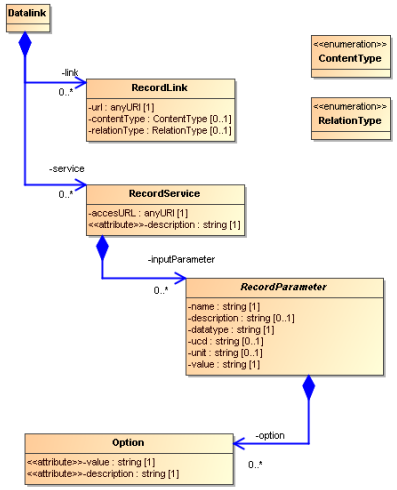

Fig. 3

5. Response Data Model II

(vgl. Fig. 3)

6. Datalink Response: Links

The serialization of the actual links is a breeze with current VO-DML drafts. The service parameters currently are tricky since we want them to reside in physical VOTable PARAM elements to accomodate current VO viewing habits. Putting them standard VO-DML serialization might offer enough advantages to pursue this (e.g., declaration of parameter dependencies).

```
<RESOURCE name="datalinkDescriptor">...
  <GROUP name="DatalinkDM" utype="vo-dml:Model">
    <PARAM utype="vo-dml:Model.name" value="dl"/>
  <GROUP ID="_dl" utype="vo-dml:ObjectType.instance">
    <PARAM utype="vo-dml:Instance.type" value="dl:Datalink"/>
           <PARAM utype="dl:Service.accessURL"
             value="http://localhost:8080/mlqso/q/d/dlget"/>
  <GROUP ID="rr" utype="vo-dml:ObjectType.instance">
    <PARAM utype="vo-dml:Instance.type" value="dl:RelatedResource"/>
      <FIELDref ref="_urlField" utype="dl:RelatedResource.url"/>
      <FIELDref ref="_contentTypeField" utype="dl:RelatedResource.contentType"/>
      <FIELDref ref="_rtField" utype="dl:RelatedResource.relationType"/>
  <TABLE name="relatedData">
    <FIELD ID="_urlField" arraysize="*" datatype="char" name="url"/>...
          <TR>
            <TD>http://localhost:8080/mlqso/q/d/static/Q0142_err.fits</TD>
            <TD>image/fits</TD>
            <TD>errors</TD>
          </TR>
```


7. Datalink Response: Service

Note how we tried to keep the service interface clear: floats are floats (and not elements of some ad-hoc query language), and the ranges of valid values are part of the metadata.

The service paramater PUBDID would probably be required on all datalink services.

```
<PARAM arraysize="*" datatype="char" name="PUBDID" ucd="" value="">
  <DESCRIPTION>The pubisher DID of the dataset of interest</DESCRIPTION>
</PARAM>
<PARAM datatype="float" name="DEC_MIN" unit="deg"
    ucd="par.min;pos.eq.dec">
  <DESCRIPTION>The latitude coordinate, lower limit</DESCRIPTION>
  <VALUES>
    <MIN value="-9.75494659974"/>
    <MAX value="-9.75449779811"/>
<PARAM datatype="float" name="DEC_MAX" unit="deg"
     ucd="par.max;pos.eq.dec">
<PARAM datatype="float" name="RA_MIN" ucd="par.min;pos.eq.ra">
<PARAM datatype="float" name="WAVELEN_1_MIN" unit="nm">
```

See full versions of those by using the links at the end of this document (you'll want to pipe those through an XML pretty printer; the author's choice is xmlstarlet).

Or rather: Just use PDL?

8. Declaring STC metadata

We use STC-in-VOTable to declare STC metadata of the params (abridged):

```
<GROUP utype="stc:CatalogEntryLocation">
  <PARAM utype="stc:AstroCoordSystem.SpaceFrame.CoordRefFrame"
    value="ICRS"/>
  <PARAM utype="stc:AstroCoordSystem.SpaceFrame.ReferencePosition"
    value="BARYCENTER"/>
  <PARAMref ref="l_min"
    utype="stc:AstroCoordArea.Position2VecInterval.LoLimit2Vec.C1"/>
  <PARAMref ref="l_max"
    utype="stc:AstroCoordArea.Position2VecInterval.HiLimit2Vec.C1"/>
  <PARAMref ref="b_min"
    utype="stc:AstroCoordArea.Position2VecInterval.LoLimit2Vec.C2"/>
</GROUP>
```

Recommend supporting ICRS positions and reserve names for that?

9. Implementation status

Our proposal is live in the GAVO DC on slit spectra, spectral cubes, plain images, and spectra.

For spectra, it replaces Skoda&Demleitner's getData proposal. Since that's been proven to work client-server, we're confident clients are workable here, too.

10. Suggested parameters

I believe it would be smart to reserve and define some "standard" parameter names. Here's some I'd like to see:

- `(RA|DEC)_(MAX|MIN)` – ICRS coordinate limits
- `(LAT|LON)_(MAX|MIN)` – coordinate limits in service system, cf. STC declaration
- `WHATEVER_(MAX|MIN)` – limits on other axes
- `PIXn_(MAX|MIN)` – cutout in pixel indices
- `KIND=HEADER|PREVIEW|DATA` – retrieve header, preview, actual data

11. For more exciting material...

...– namely, how to define such services in DaCHS and live URLs to play around with – check out the PDF on the Wiki and/or

http://docs.g-vo.org/DaCHS/ref.html#datalink-cores[1]

12. Service operator's view

In DaCHS, datalink services are defined using several types of code:

- exactly one descriptor generator,
- zero or more data functions, generating and manipulating data
- zero or one formatters, formatting the generated and/or manipulated data
- zero or more meta makers, generating input parameter descriptions for data functions and any formatter present and/or related links

I'm not quite happy that the input parameters are generated in entities separate from the data functions they declare the input for, but the alternatives I've investigated were worse.

In normal operation, a metadata query is handled by running the descriptor generator and the meta makers, which are supposed to be fast (e.g., just read the header of a FITS file).

The data functions are only executed when a request for data comes in. In DaCHS' interface, that's when there are service parameters besides PUBDID.

[1] `http://docs.g-vo.org/DaCHS/ref.html#datalink-cores`

13. DaCHS datalink examples I

A service for spectrum recalibration, cutouts, and reformatting:

```
<datalinkCore>
  <descriptorGenerator procDef="//datalink#sdm_genDesc">
    <bind name="ssaTD">"\rdId#hcdtest"</bind>
  </descriptorGenerator>
  <dataFunction procDef="//datalink#sdm_genData">
    <bind name="builder">"\rdId#datamaker"</bind>
  </dataFunction>
  <FEED source="//datalink#sdm_plainfluxcalib"/>
  <FEED source="//datalink#sdm_cutout"/>
  <FEED source="//datalink#sdm_format"/>
</datalinkCore>
```

The FEED elements hide combinations of metaMaker and dataFunctions for the manipulations this core is supposed to do. The last one, sdm_format pulls in a dataFormatter. To give you an idea how such a thing might look like, here's it's source:

```
  <STREAM id="sdm_format">
    <doc>A formatter for SDM data, together with its input key
    for FORMAT.
    </doc>

    <metaMaker>
      <code>
        formatsAvailable = {
            "application/x-votable+xml": "VOTable, binary encoding",
            "application/x-votable+xml;encoding=tabledata":
              "VOTable, tabledata encoding",
            "text/plain": "Tab separated values",
            "text/csv": "Comma separated values",
            "application/fits": "FITS binary table"}

        if descriptor.mime not in formatsAvailable:
          formatsAvailable[descriptor.mime] = "Original format"

        yield MS(InputKey, name="FORMAT", type="text",
          multiplicity="single",
          description="MIME type of the output format",
          values = MS(Values,
            options = [MS(Option, title=value, content_=key)
              for key, value in formatsAvailable.iteritems()]))
      </code>
    </metaMaker>

    <dataFormatter>
      <code>
        from gavo.protocols import sdm

        if len(descriptor.data.getPrimaryTable().rows)==0:
          raise base.ValidationError("Spectrum is empty.", "(various)")

        return sdm.formatSDMData(descriptor.data, args["FORMAT"])
      </code>
    </dataFormatter>
  </STREAM>
```

14. DaCHS Datalink examples II

A generic FITS WCS cutout service with a link to an error file:

```
<datalinkCore>
  <descriptorGenerator procDef="//datalink#fits_genDesc"/>
  <metaMaker procDef="//datalink#fits_makeWCSParams"/>
  <dataFunction procDef="//datalink#fits_makeHDUList"/>
  <dataFunction procDef="//datalink#fits_doWCSCutout"/>
  <dataFormatter procDef="//datalink#fits_formatHDUs"/>
  <metaMaker name="makeErrorLink">
    <code>
      yield LinkDef(makeAbsoluteURL(
        "get/"+descriptor.accref[:-5]+".err.fits"),
        "image/fits", "errors")
      yield LinkDef("http://foo.bar/raw/"+descriptor.accref.split("/")[-1],
        "image/fits", "raw")
    </code>
  </metaMaker>
</datalinkCore></service>
```

The idea here is to let people easily replace components by elements tailored to their concrete data, which is why there are so many building blocks.

Also note how the metaMaker generates links to related artefacts, in this case by plain string manipulations.

15. Play...

In the PDF, you're going to find links for those:

- Metadata for a slit spectrum: `http://dc.g-vo.org/mlqso/q/d?PUBDID=ivo://org.gavo.dc/~/mlqso/data/slits/Q0142_data.fits`
- spectral cutout: `http://dc.g-vo.org/mlqso/q/d?PUBDID=ivo://org.gavo.dc/~/mlqso/data/slits/Q0142_data.fits&WAVELEN_1_MIN=850`
- spatial and spectral cutout: `http://dc.g-vo.org/mlqso/q/d?PUBDID=ivo://org.gavo.dc/~/mlqso/data/slits/Q0142_data.fits&WAVELEN_1_MIN=850&RA_MIN=26.319`
- retrieve a single spectrum: `http://dc.g-vo.org/mlqso/q/d?PUBDID=ivo://org.gavo.dc/~/mlqso/data/slits/Q0142_data.fits&RA_MIN=26.319&RA_MAX=26.319&DEC_MIN=-9.7548&DEC_MAX=-9.7548`
- a datalink resource within an SSAP response: `http://dc.g-vo.org/q/s/ssap.xml?REQUEST=queryData` – look for datalinkDescriptor
- a cut-out spectrum: `http://dc.g-vo.org/mlqso/q/ssaaux/dlget?PUBDID=ivo://org.gavo.dc/mlqso/mlqso/data/slits/Q0142_data.fits&FORMAT=text/plain&LAMBDA_MIN=4.5e-8&FLUXCALIB=relative` –out of the above response, re-normalized to max=1 (where the maximum is outside the cutout; the first column is the error)
- a FITS with lots of relations: `http://dc.g-vo.org/cars/q/dl?PUBDID=ivo://org.gavo.dc/~/cars/data/images/W1m1p3/W1m1p3_z.V1.7A.swarp.cut.fits`
- a good ol' datacube: `http://dc.g-vo.org/califa/q/dl?PUBDID=ivo://org.gavo.dc/~/califa/data/V1200/reduced_v1.3c/UGC12864.V1200.rscube.fits` – note the relation to the lo-res version. The WCS of these files is slightly broken, hence the spectra coordinate comes out as COO_3 here; people *could* figure out it's spectral by the STC clause, but admittedly this needs fixing.
- cut out a single frequency: `http://dc.g-vo.org/califa/q/dl?PUBDID=ivo://org.gavo.dc/~/califa/data/V1200/reduced_v1.3c/UGC12864.V1200.rscube.fits&COO_3_MIN=4000&COO_3_MAX=4000` – yeah, we're aware we're throwing away the other extensions; the sane behaviour here needs specification.

- retrieve the header of the thing: `http://dc.g-vo.org/califa/q/dl?PUBDID=ivo://org.gavo.dc/~/califa/data/V1200/reduced_v1.3c/UGC12864.V1200.rscube.fits&KIND=HEADER`