Fig. 1

Fig. 2

1. Datalink: The GAVO perspective

(vgl. Fig. 1)

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

(vgl. Fig. 2)

- Datalink's scope
- Datalink response
- Service operators' view

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 we've been pushing in the context of our SSAP getData efforts, e.g., recalibration.

3. Datalink Response

There's still no updated draft of Datalink reflecting the Heidelberg consensus, but the data model is fairly 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

How should this be serialized? In the VO of 2013 I'd say *someone* has to write the VO-DML, and the rest follows.

Full disclosure: At this point, I'm not that someone.

4. Datalink Response ad-hoc

We're modelling the links as a simple three-column table, and the services as groups of (essentially) params.

Note that we take care to have clearly declared metadata; floats are floats (and not elements of some ad-hoc query language), and the ranges of valid values are part of the metadata.

There's a special serviceAccessURL param (in the end probably identified by utype) that tells the client where to post queries.

The service paramater PUBDID would probably be standard (though it might be worth figuring out how to declare "fixed" parameters and leave out PUBDID then).

```
<RESOURCE name="datalinkDescriptor">
  <GROUP utype="datalink:service">
    <PARAM utype="datalink:accessURL"
      name="serviceAccessURL"
      value="http://localhost:8080/data/test/foo/dlget">
      <DESCRIPTION>Access URL for this service</DESCRIPTION>
    </PARAM>
    <PARAM name="PUBDID" value=""/>
    <PARAM datatype="float" name="DEC_MIN" unit="deg" value="">
      <DESCRIPTION>The latitude coordinate, lower limit</DESCRIPTION>
      <VALUES>
        <MIN value="30.9831815872"/>
        <MAX value="30.9848485045"/>
      </VALUES>
    </PARAM>
    <PARAM datatype="float" name="DEC_MAX" unit="deg" value=""> ...
  <TABLE name="relatedData">
    <FIELD arraysize="*" datatype="char" name="url"/>
    <FIELD arraysize="*" datatype="char" name="contentType"/>
    <FIELD arraysize="*" datatype="char" name="relationType"/>
```

5. 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?

6. Datalink implementation status

Petr Skoda and I have tried to rally support for SSAP getData for a while.

Our proposal has been quite parallel to Datalink, and previous getData functionality can now be exposed through our Datalink prototype.

As soon as there's a draft with a sensible metadata format, SPLAT will grow support for Datalink-in-SSA.

Otherwise, WCS-based FITS cutouts are generically supported, extensible to let people define rich metadata.

In the GAVO DC, all relevant services will have grown datalink interfaces by the Waikoloa interop.

7. 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.

8. 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>
```

9. 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.