


| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

Developing GSM Concepts in PSS4


O.M. Smirnov

| Verified: | | | |
|-------------------|-----------|-------------|---------|
| Name | Signature | Date | Rev.nr. |
| K. van der Schaaf | o.p.v. | 2003-Jul-10 | 0.1 |

| Accepted: | | |
|----------------------|----------------------------|-----------------|
| Work Package Manager | System Engineering Manager | Program Manager |
| J. Noordam | C.M. de Vos | J. Reitsma |
| | | |
| <i>date:</i> | <i>date:</i> | <i>date:</i> |

©ASTRON 2005

All rights are reserved. Reproduction in whole or in part is prohibited without the written consent of the copyright owner.

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

Distribution list:

| Group: | For Information: |
|--|--|
| ASTRON: M. Brentjens (MAB) G. van Diepen (GVD) J. Noordam (JEN) R. Overeem (RO) K. van der Schaaf (KvdS) | ASTRON: C.M. de Vos J. Reitsma M. Loose ORDINA: K-J. Wierenga Snow B.V.: D. Hoogland |


Document revision:

| Revision | Date | Section | Page(s) | Modification |
|----------|-------------|---------|---------|--|
| 0.1 | 2003-Jul-10 | - | - | First public version |
| 0.1.1 | 2003-Aug-4 | 2.1 | 3 | Source representation depends on frequency |

Abstract

The LOFAR Global Sky Model (GSM) will be an all-sky database of some 100 million objects, with flux & polarization measurements in the 20–200 MHz range. The primary function of the GSM is to support LOFAR calibration and data reduction. The GSM is expected to provide a model of all sufficiently bright sources in any given field, having enough detail and precision to calibrate and subtract these sources and yield residual images of the faint background. The GSM is expected to be continuously updated and refined during LOFAR operation in a “closed loop” system. The GSM is also a valuable stand-alone data product that can be integrated into the VO.

An prototype version of the GSM will be developed as part of the Prototype Selfcal System 4 (PSS4, [1]). This document examines requirements, proposes some preliminary design ideas, and works towards a plan for developing the GSM within PSS4 and beyond.

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

1 Preliminary thoughts and requirements

According to JEN's PSS4 document [1]:

1. Stand-alone product
2. Subsets extracted into MEP database, and linked to MeqParms
3. Cat II prediction (Haystack simulator?)
4. Outline the development path to final size and functionality
5. Relation to NVO (interfaces!): Use the VOTable?
6. Put in all the 3C and 4C sources
7. Put in all the 3C84 sources for MAB
8. Continue adding to it with everything we do
9. Think about source representations (parameters, shapelets, pixons, images, etc.)
10. How to find subsets

2 Use cases


In order to limit scope for PSS4, yet stay on the development path towards a full-blown GSM, it is necessary to define a layered interface carefully. To do this, we first need to consider the basic use cases:

2.1 A source maps to a set of MeqNodes?

For use with a MeqTree, a GSM source has to map to a collection of MeqNodes. In the simplest case (i.e. vanilla point source), these are basic MeqParms: RA, Dec, I , Q , U , V fluxes. Any of the parameters may be variable in frequency and/or time for some sources.

The parameters of more complex sources will end up being represented by little sub-trees of their own (i.e., by a MeqExpr node with MeqParm children). For example:

- Specific representations of time or frequency dependence, more “physical” than a polynomial. Representation via spectral index – i.e. flux as an exponential of frequency – is a canonical example. The spectral index would then be the actual MeqParm [solvable if so requested], and it needs to be attached to a MeqExpr (the exponent) to compute flux.

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

- Complex sources and Cat II sources bring in a whole zoo of concepts: images, shapelets, etc. These will definitely require specialized MeqNodes and/or MeqExprs.

The complexity of this representation may also vary depending on domain. For example, a point source at higher frequencies may have spatially extended emission at lower frequencies. Thus, we may need to switch between different sub-tree representations depending on frequency.

Note also that the GSM \leftrightarrow MeqNodes representation has to be bidirectional. Once a source has been solved for, we may want to store the new parameter values back into the GSM. But see below.

2.2 Automated source finding


We need to implement automated source finding, at least in some primitive way. In terms of use cases, this implies being able to update the GSM with new sources, from both the scripting layer, and perhaps C++.

2.3 Inserting and updating sources

Note that there should be a way to treat sources as temporary and local to a specific solution or session. I.e., while the automatic source detection algorithm may generate sources, we don't know if they're really there or not until we have successfully solved for them. Similar considerations apply to updated values of pre-existing GSM sources – we may want to reuse the updated values in another solution, discard them entirely, or really commit them to the GSM.

The commit step needs to be explicit. It's OK if this step is done manually in PSS4, but we should keep an eye on the possibility of some sort of automated strategy. Should it be possible to assign several sets of parameter values to one GSM object? In that case we could also consider assigning some sort of "confidence level" to each set.

It seems we need to introduce the term *Local Sky Model* (LSM), that is, the model being employed for a specific solution. This would be created as a subset of the GSM (by doing a region search), but could then be updated with auto-located sources (or sources explicitly added by the user), refined, etc., before [possibly though not necessarily] being merged back into the GSM. Rather than a separate entity, the LSM could simply be a logical subset of the MEP database, (MEPdb) but see below for a discussion.

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

2.4 Region search

Extract the subset of the GSM (i.e. the LSM) for a given region of the sky, within a given brightness range, etc. The query can and will incorporate other criteria, but selection by coordinate is the biggest challenge. This operation could be initiated from the scripting layer, or perhaps even from C++. Whether the subset is extracted directly into data objects in the scripting language, or first into the MEPdb (whatever that is) remains to be determined. See discussion below.

2.5 Populating from existing catalogues

The GSM will be pre-populated from existing data sets. This is generally a “unique”, one-time – or at least one-time-per-catalogue operation – never done on-the-fly.

2.6 NVO, VOTable and friends

The NVO effort has produced the VOTable definition (*Proposed XML Format for Astronomical Tables*, [2]). This is directly relevant to the GSM in several ways. Despite being out of PSS4’s current scope, we should consider:


- If we want to play with NVO as a “data provider”, then at some point we need to be able to export subsets of the GSM in VOTable format.
- With a GSM→VOTable converter, we could profit from outside software packages, such as VOTable visualizers. There’s definitely a trend in developing this stuff around the world.
- More and more catalogues are being made available online in VOTable form, so a VOTable→GSM importer could become important.

As a third possibility, I could imagine using the VOTable internally at some point – see discussion later on.

3 Design considerations

3.1 Layered interfaces

Common sense & design principles call for a layered interface, with one or more layers between the database implementation and the application. At the bottom, we have the database engine itself (which,

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

in the case of PSS4, is simply an AIPS++ table). An [optional] intermediate layer could provide query-lookup-read-update-insert functions, mapping primitive structures in the database into compound objects such as “a source” or “a collection of sources for a region”.

What is clear, is that at the application level, the interface has to eventually present each source as a collection of MeqNode objects (or their defrecs). If one needs to insert, e.g., the flux of a source into a tree, one don't usually want to know if it is a single MeqParm, or a complicated MeqExpr – hence the MeqNode treatment. Similar considerations apply to, e.g., visualization.

My thinking at the moment that as far as PSS4 is concerned, we only need to elaborate the Glish (i.e. scripting layer) interface – the topmost layer – and that no knowledge of the GSM is required on the C++ side. The underlying functionality may be rapidly implemented in Glish alone, using AIPS++ Tables for storage. This is probably sufficient for PSS4 purposes, represents very little investment, and leaves us free to rip the guts out and replace them with a real database later.

3.2 Do we have an LSM?


The LSM does not necessarily need to exist as a separate entity. It could be seen as simply a logical subset of the MEPdb that deals with sources. We should, however, consider:

Lifecycle: how long does an LSM exist, before being discarded and/or merged back into the GSM?
 Certainly as long as the same observation is being processed. However, I can see a user wanting to keep his LSM around longer, perhaps to apply to a future observation of the same region.

Portability: will user John want to pass a copy of his LSM to user Jane? How?

Structure: is it sufficient to consider the LSM as simple “flat” set of parameters? If we allow complicated sources to be represented by sub-trees, then I could imagine trying to solve for one of these sources and discovering it is better modelled by a somewhat different tree. How do I represent this new tree in the LSM? Clearly, the LSM needs to contain not only parameters, but some structural relationships between them (actually, this applies to the GSM as well).

These considerations seem to imply that the LSM is something more than a simple subset of the MEPdb, so perhaps it should exist on its own at some level of the implementation. Unless, that is, we decide that similar considerations apply to the other MEPs, in which case the LSM concept can be merged into the more general concept of a “local” MEPdb.

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

3.3 VOTable

There is nothing particularly “magic” about the VOTable format, apart from the fact that it’s becoming the international data exchange standard. Being based on XML, it solves half of the I/O problem (that of reading/writing structured data), since XML parsers are available for almost all conceivable languages and platforms. The other half (making sense of the structure read, and, conversely, writing sensible structure) has to be addressed separately.


VOTable allows for a very rich semantic structure, so it’s certain that whatever specific representation we choose for the GSM, any subset can be easily mapped onto the format. Rich semantics are a double-edged sword though: mapping the other way (VOTable→GSM) can be a lot trickier, since the semantics of a VOTable from a different source may not be directly compatible, so we can only hope to support some sensible subset (and write custom conversion scripts otherwise). I propose we leave the issue at that for PSS4.

What could be considered in more detail at this point are *unified content descriptors* (developed at CDS Strasbourg, [3], [4], [5]). UCDs are a set of standard strings (labels) used in VOTable to specify, essentially, what a datum means in astronomical or physical terms. What UCDs provide is a concise, unified vocabulary for describing astronomical data. Here’s a sample (see full list at <http://vizier.u-strasbg.fr/viz-bin/UCDs>):

```

POL                Polarization Related Quantities
...
POL_STOKES         Polarization Stokes Parameters
POL_STOKES_I       Stokes parameter I (total power)
POL_STOKES_Q       Stokes Parameter Q (absolute, or relative Q/I)
POL_STOKES_U       Stokes Parameter U (absolute, or relative U/I)
POL_STOKES_V       Stokes Parameter V (absolute, or relative V/I)
POS                Position Related Quantities
POS_ANG            Angular Position
POS_ANG_DIST       Angular Distance and related quantities
  POS_ANG_DIST_GENERAL  Angular Distance Or Separation
  POS_ANG_DIST_REL      Relative or Normalized Angular Distance
  POS_ANG_DIST_SQ       Quadratic Angular Distance
  POS_ANG_VEL           Rate Of Position Change (drift motion, angular velocity)
...
POS_EQ             Equatorial Coordinates and related quantities
...
POS_EQ_DEC         Declination related quantities
  POS_EQ_DEC_3T         Third Term in Declination
  POS_EQ_DEC_MAIN       Declination
  POS_EQ_DEC_OFF        Declination or North-South Offset Difference
  POS_EQ_DEC_OTHER      Declination in Non-Standard Units or partial values
  POS_EQ_DEC_PRECESS    Precession Variation in Declination
  POS_EQ_DEC_REL        Relative Declination in a Special Scale

```

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

```

...
POS_EQ_PREC                Annual Precession Quantities
  POS_EQ_PREC_DEC          Annual Precession In Declination
  POS_EQ_PREC_RA           Precession Variation In RA
...
POS_EQ_RA                   Right Ascension related quantities
  POS_EQ_RA_2T             Second Component in right Ascension
  POS_EQ_RA_3T             Third Term In Right Ascension
  POS_EQ_RA_CORR           Correction in Right Ascension
  POS_EQ_RA_MAIN           Right Ascension
  POS_EQ_RA_OFF            RA Offset or Residual In Right Ascension or along East-West
  POS_EQ_RA_OTHER          Right Ascension in Non-Standard Units or partial values
  POS_EQ_RA_REL            Relative Right Ascension in a Special Scale

```

Adopting the official UCD list as a source of [software] vocabulary (internally in the GSM, and perhaps elsewhere in the system?) would generally simplify interaction with the VOTable format. Besides, it would reduce confusion arising from different developers inventing their own identifiers for the same things.

3.3.1 Added value within the project?

Does VOTable have some added value that can be exploited internally in the project? One application to keep in mind is the GSM↔LSM interface. Having a VOTable representation of the LSM would address some of the concerns raised in the previous section. This is out of scope for PSS4 though.


3.4 The Glish interface: sources and nodes

NB: the following code examples are meant as just that – examples, a departure point for thinking about how things would work, so they shouldn't be taken too literally. The interface we eventually implement may or may not be dissimilar. Note also that the concepts used here are in no way unique to Glish; one could easily imagine the same things done in Python.

3.4.1 Extracting an LSM

To begin, we need to extract a subset of the GSM into the LSM. This is done “once”, before commencing calibration of a data set.

```
include 'GSM.g'
```


| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

```
# Attaches to global GSM table. Optional tablename argument
# allows for testing with different versions of the GSM
GSM := attach_gsm([table_name]);
# Extract an LSM from the GSM.
# This record specifies the query parameters. Using a record
# allows for maximum flexibility in the interface:
query := [ ra=ra,dec=dec,radius=radius,other optional fields ];
# If the LSM exists as part of the MEPdb, then this will copy
# a subset of the GSM into the MEPdb. The MEPdb will perhaps be
# specified here in the call. Alternatively, the LSM could be
# extracted into a separate table of its own. In that case, the
# table name should be specified here.
lsm := GSM.extract_region(query,???)
```

Once an LSM is extracted, we need to be able to reuse it in future sessions without going back to the GSM:


```
# if the LSM exists in its own table
lsm := attach_lsm(tablename)
# ... or if the LSM lives inside the MEPdb
lsm := attach_lsm([mepdb]);
# ... the mepdb argument specifies the MEPdb somehow.
```

3.4.2 Source lists

Now that we have an LSM, we need to insert sources into MeqTrees. But first we need a list of the sources in the LSM. It is useful to have this list already pre-sorted in some order (e.g. by brightness – if you want to peel in order of brightness¹). Also, perhaps we want only a subset of the sources?

```
# Extracts source list. Both arguments are optional:
# if no sort_by is given, returns unsorted list;
# if no subset is given, returns all the LSM sources.
# The subset argument could be defined in the same way as the
# query argument in the GSM examples above. Note also the
# sneaky use of a UCD for sort_by:
```

¹There's a separate issue here, that a source does not necessarily have a single value for brightness – in many cases it would be a function of time and/or frequency. How would we sort on that? One answer is to store a separate “representative” brightness – an average or approximate value. You could then sort on this approximate value, and use the full functional representation in calibration. This means that MeqNodes would have to be responsible for calculating “representative” values.

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

```
sources := lsm.select_sources(sort_by='pol_stokes_i',subset=query);
```

What is `sources`? Maybe just a list (i.e. vector, in Glish terms) of names, or IDs, or in any case “thingies” we can later refer to a source by. But I think it would be even more liberating (think “Freedom layer”) if this was actually a list of records (record of records, in Glish terms) with additional information. E.g. each source record would be something like:

```
[ id=source_id,name=descriptive_name,
  ra=ra,dec=dec,pol_stokes_i=... ]
```

The `id` field is what we use to refer to a source later (i.e. index). From a purely functional standpoint, this is sufficient, since you should also be able to access the full source data via the `id`. However, the other fields contain information about the source that could be very useful in making calibration decisions later on (not to mention visualization etc.), so it’s handy to have it around from the start. How much or how little information do we want to provide here? I suggest we make it all optional, that is, determined by an optional argument to the `select_sources()` method:

```
sources := lsm.select_sources(sort_by='pol_stokes_i',
                             subset=query,fields="name ra dec pol_stokes_i");
# fields argument is optional; the default value would be
# something like the one shown here
```


Note that in this form the method mirrors the `SELECT` statement in SQL (as in `SELECT columns FROM table WHERE subset_criteria ORDER BY what`). So we’re really dealing with an ubiquitous concept here – which probably shows that we’re on the right track.

3.4.3 Hanging sources off trees

At some point we get to constructing `MeqTrees`, where we’ll probably loop over sources (see [1]).

```
for( i in 1:len(sources) )
{
  defrec := lsm.source_node(sources[i].id,'pol_stokes_i');
  node_index := MeqNode.define(name,defrec);
  ...
}
```

The `source_node()` call here returns the *node definition record* for the Stokes *I* parameter of the given source. This is, of course, just the `defrec` in JEN’s terms [1], with sufficient information to create the

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

node, so it can be passed directly to `MeqNode.define()`. For example, if I is a single `MeqParm` with polynomial t, f dependence, the defrec returned would look something like:

```
[ class    = 'meqparm',
  id       = parm_id,
  name     = 'pol_stokes_i',
  domain   = ...,
  polc     = [array of polcs],
  ... ]
```


What if we want the `MeqParm` to be solvable? JEN [1] proposes a `MeqParm.set_solvable()` method. Which is good, but we could also provide the additional ability to mark a parm as solvable at creation time, by saying

```
defrec.solvable := T
```

before passing the defrec to `MeqParm.define()`. This is in keeping with the concept of the defrec containing all the necessary information to create a node.

Let's take it one step further. Suppose I was represented in exponential form, with a spectral index. What would its defrec look like? Without meaning to go into detail about `MeqExpr` semantics – what's shown here is just a conceptual defrec – how about:

```
[ class    = 'meqexpr',
  ...
  func     = 'exp',
  children = [
    *1 = [ class    = 'meqparm',
          ...
          name     = 'spect_sp-index', # another UCD
          polc     = [...],
          solvable = T
          ... ],
    *2 = [ class    = 'meqparm',
          name     = 'freq',
          polc     = [0,1],
          solvable = F
          ... ]
  ]
];
```

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

The top-level defrec corresponds to a MeqExpr node implementing the $\exp(x_1 \dots x_n)$ function. The `children` field contains a list of child defrecs – in this case, x_1 is a MeqParm representing the [solvable] spectral index, and x_2 is frequency term (the `polc` array given corresponds to f).² Essentially, we’re representing a tiny tree here. The `MeqParm.define()` method can then recursively define the child nodes (to any level of nesting!), followed by the top-level node.

A note on terminology. *To avoid confusion, we need to clearly distinguish source parameters from atomic MeqParms. In the example here, Stokes I is a source parameter, which could be represented by a single MeqParm, or by a compound expression – subtree – involving one or more atomic MeqParms (themselves polynomials of t, f). In this document, I use **source parameter** to refer to things like $RA, Dec, Stokes I, \dots$, and **MeqParm** to refer to their constituent MeqParm nodes.*

Note three emerging powerful concepts here:

- At the script level, when constructing a tree, one **does not care** how a source parameter is represented. It could be a single MeqParm, it could be a subtree – the code to insert this parameter at a given point in the MeqTree remains exactly the same.
- GSM sources can be represented to any level of complexity, by using subtrees to represent their parameters.
- You don’t even need a GSM source! Suppose you want to add an extra source to see if that improves calibration. No need to insert it into the LSM – just construct the appropriate defrecs in Glish, and insert them into your trees. The C++ side of things (and trees in general) don’t know or care whether the sources come from the GSM, or have been added by the user on-the-fly. Once you’ve determined that the source fits the data, then you can commit it to the LSM.


The same goes for modifying a source parameter. A source not being fitted properly, because I seems to have a more complex t, f dependence than that stored in the LSM? Modify its defrec (perhaps setting up a more complex subtree) before passing it to `MeqParm.define()`, and if that works out, you can commit the new representation back to the LSM.

3.4.4 Viewing and committing the results

Once we’ve solved for source parameters, we want to (a) look at them, and more importantly (b) commit them back to the LSM if the solution is good. At this point, the information resides in MeqNodes on the C++ side. Using JEN’s mechanism [1], you would do something like:

```
state_record := MeqNode.get_state(node_index, [recurse=-1, ...]);
```

²The conventional spectral index representation ($\nu^n, n < 0$) differs only by a renormalization term.

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

...to obtain the state of a node together with all of its children (`recurse=-1`), down to the MeqParm leaves. (Of course we need to know the node index first – more on that below.) At this point, the state record contains everything you need to know (provided the sufficient level of detail is specified in the "... " part of the call, somehow) about the source parameter³, which is sufficient for visualization, etc.

When and how should this information be written back to the LSM? Someone, somewhere has to say “commit”. This is a control decision, and belongs on the policy side of the policy barrier [1], so it should be initiated in Glish. A general solution would be a call like...

`MeqNode.commit_node(node_index or indices)`, with a recursion argument, to commit all MeqParms in the subtree rooted at this node⁴. Note that this applies to all MEPs, not just source parameters.

Where would we get the `node_index` argument? A source has parameters, and these become associated with MeqNodes (either single MeqParms or entire subtrees). The mapping between a source parameter and its node is transient – established only at run-time, when the tree is constructed. In the code fragment above:

```
defrec := lsm.source_node(sources[i].id, 'pol_stokes_i');
node_index := MeqNode.define(name, defrec);
```


...something has to happen to associate `node_index` with the *I* parameter of this source. Perhaps the `lsm` object should be allowed to call `.define()` by itself, and put the resulting node index into the `defrec`? An alternative suggestion is proposed below. This requires further discussion.

Another thing to consider is that the nodes associated with a source belong together and should usually be committed as a unit. In database parlance, they should be committed in a single transaction. If your program happens to crash (for whatever reason) while the MeqParms are being stored, this leaves the database in an unknown, possibly corrupt state (which values have been written? which haven't?) It's even more dangerous if you're storing the subtree representation (i.e. structural relationships) of a source. All modern DBMSs provide a transaction mechanism to avoid this problem. A program signals the start of a transaction, stores new values, then commits the transaction – and only at the commit point do all the new values appear in the database, as a unit. If anything fails at any point before or during the commit, then the database is automatically rolled back to the previous “known good” state, that before the start of the transaction. While we can't easily implement transactions with AIPS++ tables alone, we should certainly take it into account while designing the interface. Therefore, something like

```
lsm.commit_source(source(s));
lsm.commit_all_sources();
```

³During discussions with JEN, the concepts of a state record and a defrec have been stealthily converging. A defrec is basically the complete *initial* state record of a node.

⁴Here's a good question: should this recursion be always implicit?

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

seems entirely appropriate. Future implementations can then employ transactions inside these methods.

3.4.5 The source as a compound object

Up till now, I've treated the source parameters as completely independent entities (except for the transaction discussion above), with each parameter represented by its own MeqParm or subtree. But are they always? One could imagine an extended source where the brightness is a function of position. Depending on how that is modelled, the nodes representing position may be shared with the subtree representing brightness. For that matter, the Stokes parameters may all depend on the same MeqParm (spectral index?). This means that you can't quite treat the parameter nodes (subtrees) separately from each other, since they may share child nodes (and not just the leaf MeqParms – maybe whole subtrees as well). On the other hand, you don't want the application layer to bother with this complexity when defining trees.

Fortunately, this doesn't break the paradigm at all. Glish (and Python, and most mature languages) allows objects to be multiply referenced. Consider:

```
i_rec := lsm.source_node(sources[i].id, 'pol_stokes_i');
q_rec := lsm.source_node(sources[i].id, 'pol_stokes_q');
```


The `i_rec` and `q_rec` defrecs (more specifically, their `children` field) can refer to the same child defrec, representing a single MeqParm (e.g. spectral index). This multiple reference can be set up inside the `lsm` object (via the Glish `ref` statement). If a user wants to construct a source on-the-fly, he can use the same technique.

On the `MeqParm.define()` side of things, when you say:

```
i_index := MeqParm.define(name_i, i_rec);
q_index := MeqParm.define(name_q, q_rec);
```

someone has to figure out that a node is shared. This can be elegantly handled in `define()` by *inserting the node index into the defrec* once a node is created. So, the first `define()` call above would create the shared MeqParm (perhaps somewhere far down the subtree), and insert a `node_index` field into its defrec. The second `define()` call, while recursively creating its own subtree, would eventually come across a defrec with an already defined node index field. It would then know that this node has already been created, and just use its index directly when creating the parent node.

To conveniently address this data together, the source record (the one returned by `lsm.select_sources()`) could actually include refs to the defrecs of its parameters. This would allow the application layer to manipulate each source as a single entity – and easily address questions like, if this is the source, what are its parameters' nodes, and what are their values?

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

3.4.6 Where is the code?

Note that in all of the examples above, no explicit GSM support is implied on the C++ side, since everything is done in terms of generic MeqNode functionality. Stuff specific to GSM/LSM can stay entirely on the scripting side. This plays very well with the “policy barrier”, and prevents extra complexity from entering the C++ domain (as if it needed any extra complexity!)

Initial GSM/LSM support can thus be done entirely in Glish, with use of the `table.g` module and/or MEPdb functions (however those are implemented). When Glish performance becomes an issue – whether in the scope of PSS4, or further down the road – critical parts can be reimplemented in C++. (Besides, once the D-team adds support for a real DBMS, a lot of scripting code is going to be phased out, so the performance issue may not come up at all.) The really good thing about this is that Glish code represents relatively little investment, compared to C++, from a man-hours/functionality point of view. This will allow us to move forward rapidly, and go back for a more in-depth implementation only when performance becomes a limiting factor.

3.4.7 Some preliminary conclusions


1. While the GSM as a data product (together with its support tools) is a stand-alone beast, its design should be considered together with the MEPdb, as it shares much of the underlying model.
2. Trees and GSM sources go hand-in-hand, since source parameters need to be represented by sub-trees.
3. Initial development (and perhaps all PSS4-scope development) can be done rapidly in Glish.

3.5 Automatic source finding

From an algorithmic standpoint, this is an entirely separate problem. The AIPS++ `image` tool provides a function for finding point sources [6]. It remains to be determined how useful or functional this is.

The problem has also been widely studied in the literature, so, should the power of AIPS++ prove not up to the task, there’s a wealth of experience to draw upon. In particular, the CLEAN algorithm [7] [8] has been widely used by radio astronomers to “decompose” an image into a set of point sources. Looking across disciplines, optical astronomers have worked on similar problems for ages, for applications such as crowded-field photometry (see, e.g., the DAOPHOT II package [9]).

From a data management point of view, any source finder, be it based on CLEAN or something else, can be viewed as a “black box” that produces a collection of point sources, given a set of images. This collection can be in the form of a conventional list (positions/brightnesses), or perhaps an image (with each non-zero pixel representing a point source). These lists or images may be organized into data cubes,

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

if frequency and polarization are taken into account. However, as far as GSM design is concerned, these algorithmic issues are completely orthogonal. Therefore, we can assume that we'll eventually have a "magic tool" that can find sources in an image, and produce them in the form of a source list. What are the implications for GSM?

- The process can be handled entirely from the scripting side. Imaging will be done from Glish anyway; the tool can be applied to the initial dirty image and/or to residual images produced by successive calibration steps.
- Once the tool has produced a list of sources (as a minimum, position & brightness guesstimates), these can be turned into defrecs compliant with the overall scheme, and appropriate MeqParms can be created and solved for.
- Sources for which a solution is unsuccessful can be discarded (their nodes trimmed from relevant trees).
- Sources solved for successfully can then be committed to the LSM.

Note that this is effectively no different than having a user add sources on-the-fly himself, which has already been touched upon in the discussion above. The source of the sources, so to speak, is different, but further mechanics would be the same. By handling the sources entirely in Glish, we can rapidly "glue" an automated source finder into the overall scheme of things.


3.6 Database considerations

The complex nature of GSM sources does not play well with the traditional relational database at all, and thus presents a challenge for the database designer. The fact that sources and parameters need to be represented in a non-uniform way (images, shapelets, the whole zoo), and that there is complex metastructure linking the parameters (e.g. subtrees), complicates matters considerably. The full scope of this problem will have to be addressed by the D-team. In the meantime, we need to outline a path within PSS4 and beyond.

3.6.1 Prior art

Astronomical catalogues of the first electronic generation were firmly rooted in FORTRAN legacy. They were flat ASCII tables, with a uniform format for every single source.⁵ The shift to FITS Tables and/or relational DBMSs did little to introduce any new paradigms, as a uniform table format continued to be

⁵In fact, the *VizieR* service at CDS Strasbourg [10] – the biggest consolidation of astronomical catalogues to date, including the latest monsters such as USNO-B (over 1 billion objects!) still specifies source catalog format in FORTRAN77 terms.

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

employed throughout⁶. Any variability (in our case, t, f dependence) in the source was represented in the same some sort of fixed and uniform format (via optionally-filled columns for proper motions, spectral indices, a fixed set of bandpass fluxes, or perhaps spectra).

In the past several years, people have been looking towards object-oriented databases (OODBs – OODBMSs) to address the question of representing complex and non-uniform data. A few examples:

- Baruffolo & Benacchio 1998 [13] have evaluated an “object-relational” approach, with a look towards implementing complex queries (i.e. region search) and multidimensional indices. The engine employed was PostgreSQL 6.0.
- The Science Survey Centre for the XMM-Newton mission has deployed an object-oriented data depository for the mission’s science products [11]. This uses the now-discontinued O2 OODBMS. Their object model includes more than 300 distinct classes.
- CDS Strasbourg has been evaluating OODB technology for their SIMBAD⁷ system [12]. The results (as expected) did not match up too well, performance-wise, with SIMBAD’s dedicated C software (speed was 20% to 75% slower, and disk space consumption went up by a factor of 3.) On the other hand, an OODBMS clearly offers far more powerful capabilities where heterogenous data is concerned, and allows new features to be added much more rapidly, while Moore’s law mitigates the poorer performance, making it even irrelevant in some cases.
- The AMASE project (Astrophysics Multi-spectral Archive Search Engine, [14], [15]) consolidates heterogenous observational data from several space missions. The project seems to be hibernating at the moment (the last publication I could locate was from 1999, and their website hasn’t been modified since 2001). It uses (or used) the Informix-Illustra DBMS engine. The last reported DB size was less than staggering – on the order of 10^5 entries, for a total size of 250Mb.

Finally, the NVO effort is clearly taking the right approach by basing VOTable on XML. XML excels at representing non-uniform structure of arbitrary complexity. This makes VOTable a powerful *data interchange* format, but does not address operational data management at all. Current NVO efforts are mostly aimed towards converting the output of existing systems to VOTable form.


My conclusion is that while some current systems do deal with complex and non-uniform data, they’re geared towards archiving, research, and data mining. The approach seems to be, make sure we store structure and throw all related data at the user, and let him make sense of it. Catalogues employed in

⁶Though (as a curious sidenote) Rots et al. 2001 [16] have proposed an embedded function format for FITS binary tables. This represents multi-dimensional data in terms of mathematical expressions of various parameters. While hardly relevant to our purposes, this demonstrates that you can get a long way with a crusty old format, given enough will and imagination!

⁷SIMBAD “brings together basic data, cross-identifications, observational measurements, and bibliography, for celestial objects outside the solar system: stars, galaxies, and nonstellar objects within our galaxy, or in external galaxies.

The acronym SIMBAD stands for Set of Identifications, Measurements, and Bibliography for Astronomical Data.

SIMBAD contains information for about 1 million objects, for which 3.3 million identifiers, more than 1.5 million observational measurements and 1.4 million bibliographical references are available.” (<http://cdsweb.u-strasbg.fr/Simbad.html>)

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

operational use still use a fixed, uniform data format, with very minimal representation of variability. And I could not find any parallels to the GSM → LSM → calibration → LSM → GSM cycle that we are contemplating here. Clearly, we're treading in unexplored territory.⁸

3.6.2 Defining the scope for PSS4


Given this complex problem, what can we hope to accomplish in PSS4? AIPS++ tables are not intended to represent complex non-uniform objects. As a reasonable first-stage compromise, we can implement a uniform flat-table structure such as that used by NEWSTAR, with a fixed set of MeqParms per each source:

- RA/Dec;
- I_0, Q_0, U_0, V_0 fluxes;
- Spectral index;
- Rotation measure;
- Spatial extent/orientation/ellipticity (for modelling extended sources with a 2D Gaussian).

Note that this collection of MeqParms already implies construction of sub-trees. For example, the Stokes I of a source would be represented by a subtree involving the I_0 MeqParm and the spectral index MeqParm. The *knowledge* required to construct such a subtree would be hardwired into the GSM code – i.e. not yet reside in the database. (Implementation-wise, the nested defrecs (see above) defining the sub-trees would be hard-wired in the `lsm` Glish code.)

This should allow us to play with complex source parameter representation inside MeqTrees, while maintaining a simple AIPS++ table layout. We should review the interface and the underlying data model, and answer the question, does this satisfactorily provide for complex sources? Can we add them to the GSM at a later date without breaking application-level scripts [too much]? For the examples above, the answer seems to be “yes”. These examples are, however, just a departure point – we will surely evolve the interface as the design is elaborated. This is the question we should return to, to make sure it's evolving in the right direction.

⁸At least in the astronomy domain. It could be worthwhile to cast an eye across other disciplines. Do gene-sequencing people perhaps deal with similar problems?

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

3.7 Region search

One of the most critical – performance-wise – functions of the GSM is that of the region search (a.k.a. conesearch), as in, give me all the sources within this field of view.⁹ As the dataset grows larger and larger, implementing this function efficiently becomes more important, since existing databases do not provide intrinsic multidimensional indexing capabilities of this kind.

This has been studied by various astronomical data people at great length, yielding various clever *sky indexing* schemes (see [17], [18], [19] for a sampling).

Fortunately, the details of this can be completely hidden within the implementation layer. The application layer need only expose a generic query function. As far as PSS4 is concerned, we can probably get away with a simple linear search of the entire database – this is not an operation that happens frequently, and the GSM is still small enough. In the future, we will certainly have to implement true sky indexing. This may lie in the D-team domain.


As a final note, NVO defines a “Simple Cone Search” interface for data providers [20]. This specifies a syntax for a web service that takes RA, Dec & search radius as arguments, and returns a VOTable of all sources within the specified cone. It also imposes [very few] simple specifications on the VOTable layout. Given a GSM→VOTable conversion tool, and the region search function, producing an NVO-compliant cone search service is a trivial exercise. A list of currently available Cone Search services can be found at [21]. (Radio appears to be pathetically under-represented.)

4 Stepwise implementation plan

A proposed plan for implementing a GSM prototype and evolving it towards a “production” version:

1. Elaborate the design proposed here. Produce a few trial Glish scripts to get a feel for the interface.
2. Finalize interface design. Develop an AIPS++ table structure along the lines suggested in section 3.6.2.
3. Produce pilot GSM (GSM-1). This should be sufficient to support PSS4 targets.
 - Associated software fully implemented in Glish;
 - Populated with 3C/4C sources;
 - Contains a detailed source model for 3C84;

⁹Additional search criteria – such as brightness cut-offs – are certainly possible and should be supported. However, they are mostly trivial algorithmically. The problem of selecting by coordinate is by far the toughest nut, given a large enough database.

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

- Resulting size: ≈ 5000 sources;
- All queries done via linear search.


GSM-1 would already be a stand-alone data product, with a documented AIPS++ table format. As such, it could potentially be forked and extended by our end-users.

It is difficult to plan specific steps beyond this stage, since there are several parallel directions of development, and a large part of the work is expected to move into the D-team domain. The following general directions and/or milestones may be projected:


1. (D-team) Work towards a “production” implementation of the GSM (GSM-2) using a commercial database engine. Besides the main body of database design & implementation work, this should also include:
 - (a) Re-implementing the application interface layer in Python.
 - (b) Implementing a backwards-compatible Glish interface to GSM-2 (if Glish support is still required, which it probably will be). It should be possible to replace GSM-1 with GSM-2 (this will probably imply an upgrade in MEPdb as well) without disrupting the rest of the PSS system.
2. (R-team) Experiment with Cat II sources and tree representations of complex extended sources. Add primitive support for storage of subtrees to GSM-1. Full support should be provided by GSM-2.
3. Work on populating GSM-2 from larger and larger catalogues. The VizieR service at CDS Strasbourg [10] will be extremely useful here.
4. When the performance of linear search becomes unacceptable due to growing size, implement a multidimensional indexing scheme for region search in GSM-2.
5. Develop a VOTable interface to GSM-2.

References

- [1] Noordam, J.E. 2003, *Prototype Selfcal System 4 (PSS4)*, LOFAR-ASTRON-DOC-?????
- [2] Williams, R. et al. 2002, *VOTable: A Proposed XML Format for Astronomical Tables*, <http://cdsweb.u-strasbg.fr/doc/VOTable/>
- [3] CDS 2002, *Unified Content Descriptors*, <http://vizier.u-strasbg.fr/doc/UCD.htm>

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

- [4] CDS 2002, *UCD Tools*,
<http://vizier.u-strasbg.fr/UCD/>
- [5] Derriere, S. et al. 2003, *Metadata for the VO: The Case of UCDs*, in ASP Conf. Ser., Vol. 295 (*ADASS XII*), 69, <http://adass.org/adass/proceedings/adass02/P1-1/>
- [6] AIPS++ Documentation, User Reference Manual, tool image, function image.findsources
<http://www.astron.nl/aips++/docs/user/General/node54.html#images:image.findsources.function>
- [7] Cornwell, T. & Braun, R. 1989, *Deconvolution*, in Synthesis Imaging in Radio Astronomy: Third NRAO Summer School 1988, ASP, 178
- [8] *CLEAN Algorithm*,
<http://scienceworld.wolfram.com/physics/CLEANAlgorithm.html>
- [9] Stetson, P. 1992, *Initial Experiments With DAOPHOT II and WFC Images*, in Third ESO/ST-ECF Data Analysis Workshop, eds. P.J. Grosbol & R.H. Warmels (Garching: ESO), 187
- [10] *The VizieR Catalogue Service*,
<http://vizier.u-strasbg.fr/cgi-bin/VizieR>
- [11] Michel, L. et al. 2003, *The XMM-Newton SSC Database: Taking Advantage of a Full Object Data Model*, in ASP Conf. Ser., Vol. 295 (*ADASS XII*), 291, <http://adass.org/adass/proceedings/adass02/P5-7/>
- [12] Wenger, M. et al. 2000, *SIMBAD as a Test Bed for two Object Oriented Database Management Systems: Objectivity/DB and O2*, in ASP Conf. Ser., Vol. 216 (*ADASS IX*), 247, <http://adass.org/adass/proceedings/adass99/09-06/>
- [13] Baruffolo, A. & Benacchio, L. 1998, *Object-Relational DBMSs for Large Astronomical Catalogue Management*, in ASP Conf. Ser., Vol. 145 (*ADASS VII*), 382, <http://adass.org/adass/proceedings/adass97/baruffoloa1.html>
- [14] Cheung, C. Y. et al. 1999, *A Search and Discovery Tool – AMASE*, in ASP Conf. Ser., Vol. 172 (*ADASS VIII*), 213, <http://adass.org/adass/proceedings/adass98/cheungcy/>
- [15] AMASE Project Website,
<http://amase.gsfc.nasa.gov/amase/WelcomeToAMASE.html>
- [16] Rots, A.H. et al. 2001, *The FITS Embedded Function Format*, in ASP Conf. Ser., Vol. 238 (*ADASS X*), 479, <http://adass.org/adass/proceedings/adass00/P1-33/>
- [17] Ortiz, P.F. 2003, *Why Indexing the Sky is Desirable*, in ASP Conf. Ser., Vol. 295 (*ADASS XII*), 35, <http://adass.org/adass/proceedings/adass02/010-2/>
- [18] Page, C. G. 2003, *A New Way of Joining Source Catalogs using a Relational Database Management System*, in ASP Conf. Ser., Vol. 295 (*ADASS XII*), 39, <http://adass.org/adass/proceedings/adass02/010-4/>

| | | | |
|----------------------|---|---|---|
| Author: O.M. Smirnov | Date of issue: 2003-Jul-10 Kind of issue: Public | Scope: CEP Doc.nr.: LOFAR-ASTRON-MEM-099 |  |
| | Status: Draft Revision nr.: 0.1 | File: cvs:LOFAR/doc/GSM/gsm-prototype.tex | |

- [19] Wicenec, A.J. & Albrecht, M. 1998, *Methods for Structuring and Searching Very Large Catalogs*, in ASP Conf. Ser., Vol. 145 (*ADASS VII*), 512, <http://adass.org/adass/proceedings/adass97/wiceneca.html>
- [20] NVO, *NVO Compliance: Conesearch*, <http://www.us-vo.org/metadata/conesearch/index.html>
- [21] NVO, *VO Conesearch Profile Services*, <http://voservices.org/cone/register/showlist.asp>