Calibration, Imaging and Datasystems for AIPS++ - Report of the meeting held at Green Bank, West Virginia* 3rd - 14th February, 1992

Edited by D.L. Shone & T.J. Cornwell

18th February, 1992

Participants

L.A. Higgs (Chairman) - DRAO, Penticton, Canada & AIPS++ group
S. Bhatnagar - GMRT, India & AIPS++ group
T.J. Cornwell - NRAO, U.S.A.
P.J. Diamond - NRAO, U.S.A.
J.R. Fisher - NRAO, U.S.A.
E.B. Fomalont - NRAO, U.S.A.
J.P. Hamaker - NFRA, Dwingeloo, Netherlands
M.H. Holdaway - NRAO, U.S.A. & AIPS++ group
R.G. Noble - NRAL, Jodrell Bank, U.K.
R.J. Sault - CSIRO-ATNF, Australia & AIPS++ group
D.L. Shone - NRAL, Jodrell Bank, U.K. & AIPS++ group

1 Introduction

In November 1991, the AIPS++ consortium steering committee identified the problem of processing radio astronomical data in AIPS++, through calibration to imaging, as a major issue which should involve a number of "domain experts" from consortium institutions in addition to those working for the initial six-month period in Charlottesville. Consequently, a meeting was held in Green Bank between 3rd - 14th February 1992, with the goal of analysing these areas,

*We gratefully acknowledge the hospitality of NRAO in playing host to this meeting. NRAO is operated by Associated Universities, Inc. under a cooperative agreement with the National Science Foundation.

and producing a definition of the major object-classes involved, their interaction with each other and, where appropriate, further analysis and refinement in those areas identified as being of particular importance.

This report is intended as a working paper for the AIPS++ group. The report gives an overview of how the analysis was attempted, followed by results of the analysis of individual areas their interrelations. We conclude with a summary, and identify some directions for further work.

2 Overview of the Analysis

The approach adopted follows object-oriented analysis procedures, typified by Coad & Yourdon (1991). Indeed, most of the diagrams in this report were generated using OOATool, an object-oriented design tool which follows the notation used by Coad & Yourdon.

2.1 *The problem*

Much of the difficulty experienced in any analysis is in defining the problem domain. In a note distributed via the aips2-structures exploder on 12th December, 1991 (**Message-Id: <9112121756.AA08339@zia.aoc.nrao.edu>**, hereafter referred to as CS1), Cornwell and Shone attempted to analyse the general problem of visibility data processing. This addressed a rather specific problem, namely that of how to handle visibility data for a limited range of applications. Whilst that analysis was adequate for the problem as defined, any attempt to generalise that analysis would inevitably force other applications into an unnatural model.

We have attempted to step back and consider the more general problem of radio-astronomical data processing, as suggested by Hjellming and Glendenning ("An Initial Design for Major AIPS++ Objects" - AIPS++ group internal memorandum; 27th January 1992). Their analysis presented a highly abstract view of the problem, but one which can be used to set further development in context. We have, therefore, tried to steer a middle course between this and the approach in CS1. Our model for the problem assumes that some abstract astronomical instrument (**Telescope**) produces measurements (known as **Vegs**) related to the sky brightness distribution in a way which may be described by an **Imaging Model**. The measurements are assumed to be corrupted in ways which may be described in arbitrarily complicated **Telescope Correction/Calibration Models**. In doing this, we can consider the general problem as a kind of inversion problem, but with special treatment for data calibration and correction. In this approach, specialisation for a particular kind of Telescope may be achieved simply by changing the models for calibration/correction and imaging.

An inversion process using a particular **ImagingModel** produces an estimate of the quantities the astronomer is interested in, typically a **SkyImage** (a kind of

Image with a sky coordinate system), but there may be many other possibilities. Whilst the **ImagingModel** typically involves a Fourier relationship, this may not always be the case.

2.2 *Scope of the analysis*

The analysis described here is almost entirely at the level of an application programming interface, and thus most of the classes identified here should correspond to entities familiar to the observational astronomer using the system. As an object-oriented analysis, it inevitably provides a conceptual design for this level of the system, but details of the precise relationships between objects (and often the location of attributes) is unclear in many cases, and deferred to more detailed design.

Issues of implementation are rarely touched upon, and the development of the system will probably entail analysis and design for a number of "layers", with each being dictated largely by the requirements of the implementation of the layer above. In particular, issues of object persistence are not addressed here, nor do we say anything about strategies for input/output, interprocess communication or task/process management. Whilst this has occasionally led to difficulties in placing our analysis in the larger context of an AIPS++ system, it has also given us some freedom which will inevitably have some impact on the areas mentioned above, as well as the user interface.

2.3 *The principal areas of analysis*

Many individual areas, such as calibration and imaging, were identified for discussion by smaller groups. This eventually led to a coherent model with a number of components which have been analysed in some detail. We give a brief introduction here,

Project - This defines the relationship of the astronomer to data, and provides for management and selection of data. In some respects, this may be seen as fulfilling the role of the AIPS Catalog, but this would be a small subset of the capabilities of the Project system. Furthermore, unlike the AIPS Catalog system, this should be regarded as an optional application layer, in the sense that it may be possible for the astronomer to bypass this, and manage data directly, if he/she so wishes

Telescope - This, like the observed data itself, is a key component of the observation database. Conceptually, it acts as a "gateway", via which any of the available model information for an observation database may be accessed. This may be anything which relates to the way in which the "real" telescope, used for observing, produces and corrupts the measured **Vegs**. Typically, **Telescope** will be an object which will reference **Telescope Correction/Calibration Model** (or simply **TelescopeModel**) objects and

`ImagingModel` objects. Both of these are major areas of analysis which will be introduced next.

Telescope Correction/Calibration Models - These are models of telescopes and their components, the atmosphere, and possibly anything else which can be used to describe corrections to be applied to `Yegs`. Methods for solving for these parameters, and applying them to `Yegs` are strongly dependent on such models, and thus are tightly coupled to the particular model. Models may be as simple or as complex as is required, and their use should be selectable and controlled by the astronomer.

By coupling model parameters to the appropriate solver and corrector methods, model dependencies are largely hidden, and new models, with their appropriate methods for solution and correction may be introduced with minimal impact on the rest of the system.

Telescope models may include representations of any hardware which may be relevant (*i.e.*, the appropriate parameters may be updated, and used to correct Yegs. For example, a simple interferometer `TelescopeModel` might include `Receptors`, (to which antenna-based gains are attached) which represent inputs to a `Correlator` (with which baseline-gains are associated).

Imaging Models - These describe the way in which a sky brightness distribution may be transformed to produce `Yegs`, together with the inverse transformation. In practice, these provide for the prediction of observed data from a model image of the sky brightness distribution, and the inverse process, whereby observed data (usually corrected and calibrated) may be transformed to give an estimate of the sky brightness distribution.

These are usually related to a particular kind of telescope (*e.g.*, an interferometer or a single dish), but there may be several models for each telescope; *e.g.*, an interferometer user may require various approximations to the measurement equation, such as 2-dimensional approximation instead of the correct but computationally more expensive 3-dimensional version.

Like the Telescope calibration/correction models, it should be possible for the user to select and control imaging models, and it should be possible to implement new models without changing anything else in the system.

Image Tools - Tools for performing operations on one or more images to produce another might be regarded as being outside the scope of the meeting; however, deconvolution in particular is an important part of the imaging process which may often be intimately connected to the inversion and correction processes.

Data System - The data system is analysed as a database system which must be able to provide a variety of views and aggregations of data, as required by a particular kind of algorithm. The impact of this on the internal workings of the database has been discussed to some extent, but the implementation is not covered here.

2.4 *Reality and models - crucial distinctions*

A number of classes we have identified match real objects, particularly in Telescope and its constituents. It is important to realise that these are not intended as complete models of these real objects; rather, our models are an abstraction relevant to the data processing problem, and they can only be described in terms of components for which parameters are available. There is no point in building a complicated model of the real telescope, when we cannot hope to be able to parametrize such a model. However, our scheme does not exclude the possibility of more complicated models; it simply ensures that instead of adopting fixed, predetermined models, where change has wide impact in the system, one can replace models in a modular fashion.

Abstract models are useful in associating entities which have something in common in the "real world" (*e.g.*, such as Telescopes and models for gain errors), and we believe they aid in understanding the way in which data structures and associated methods are organised.

2.5 *General comments*

We recognise that our analysis is incomplete in many areas, and suggest that it should be seen as framework for more detailed analysis and design. We believe we have recognised the important classes, and their principal attributes and services, although these may have to be modified on closer inspection.

Some of the object classes we propose (particularly the `ImageTools`) seem procedural in nature, and initially, we tried to avoid them. However, it became apparent that these are valid and useful objects which may be identified as a particular kind of tool, with individual instances representing an algorithm with a particular set of parameters. We suggest that it may be interesting to carry-out further investigation of this idiom, in the context of the "Tools" specified in the user requirements, in order to determine whether it my be more generally useful.

2.6 *Note on the following sections*

The following sections describe the major areas of analysis in more detail. These should be read in conjunction with the Coad-Yourdon diagrams and associated notes. The diagrams immediately follow the text for each section, but the associated notes, which contain supplementary information are contained in

appendices at the end of the document, in order to prevent the main body of the report from being disrupted.

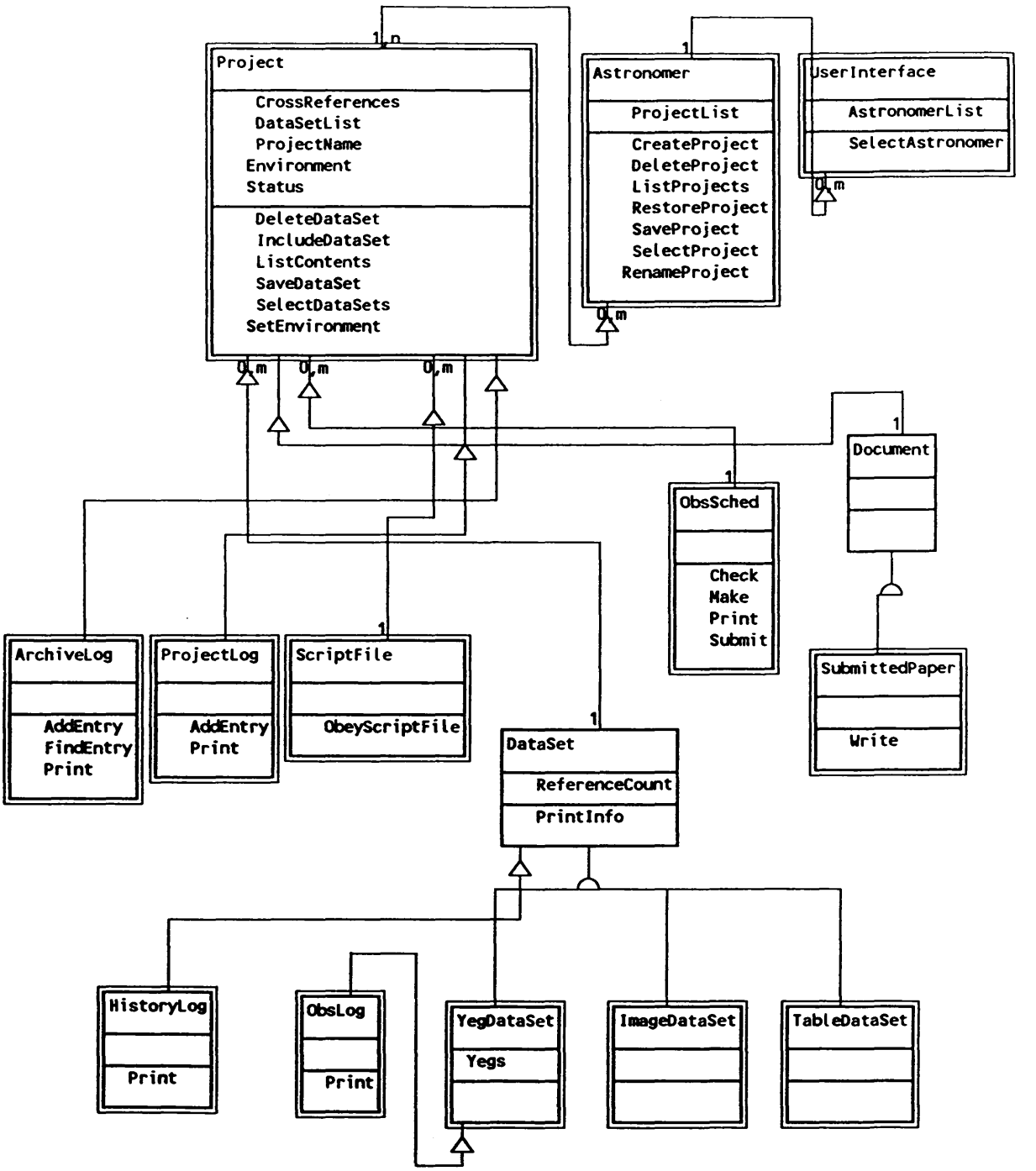
3 Project

The **Project** area of the analysis deals with capabilities the system should provide in the area of work organization for the user. A **Project** is any user-defined self-contained work unit.

A user defines and accesses his/her various **Projects** through the user interface. Depending upon implementation, the user interface may serve many astronomers, each with a defined list of **Projects**. The astronomer must be able to create new **Projects**, list existing **Projects** and select a **Project** to be worked upon. The astronomer should also be able to save a **Project** and all its associated data on a permanent storage medium, and to restore it later.

A **Project** should consist of a list of all data sets associated with the **Project** and should define any special environment (*e.g.*, Single Dish) within which the astronomer operates. The **Project** provides methods for associating new data sets, removing (and optionally deleting) data sets, and listing names of the various data sets in use. In this context, data sets can include observational data, derived images, catalogues, *etc.* A **Project** should also keep track of other types of associated data such as a log of all operations on the data sets, collections of useful scripts, text documents such as draft manuscripts, etc. The astronomer may also wish to include observing schedules with the **Project**. It should also keep track of data sets that have been stored to permanent media through an archive log.

Project Analysis 2/13/92 13:00



4 Telescopes and Basic Observable (Yegs)

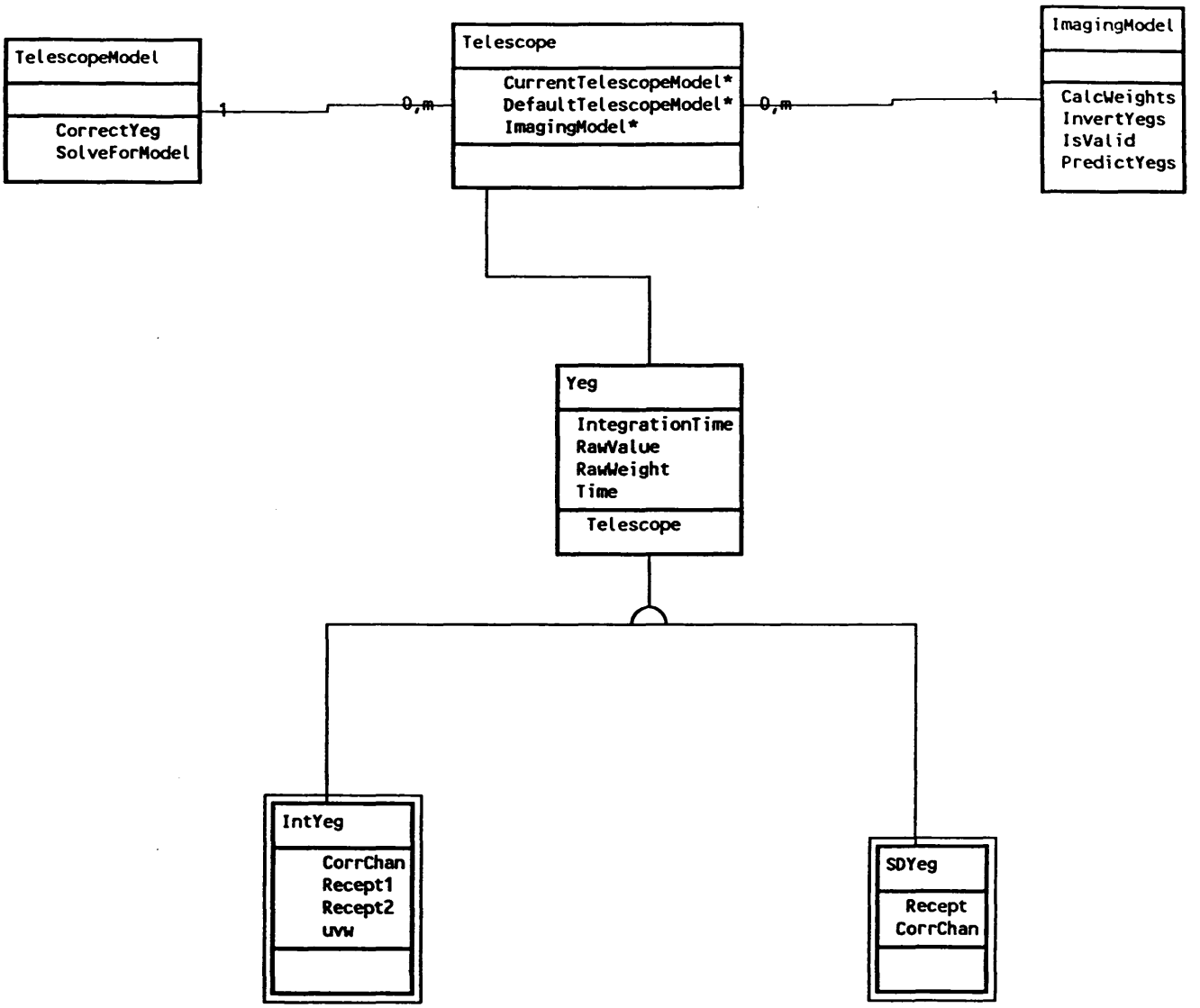
The "atomic measurement" produced by a telescope is known as a **Yeg**. Exactly what is the most atomic piece of data will vary between the different types of telescope. For example, for an interferometric array, a correlation or visibility (the output of a single correlation channel, for a given baseline, at a given time) is the most atomic piece of data. For a single dish, it may be a single total power measurement for a single integration time. Although the **Yeg** is the atomic unit as seen at the level of an application program, this is not to say that this will be the basic unit of storage. The result of a real observation includes a set of Yegs and other information (which can be placed in TelescopeModel as described below).

A **Yeg** is basically unintelligent. Apart from the data values, it knows little more than the parameters needed to tie it to the telescope setup used to observe the **Yeg**. For example, a **Yeg** for an interferometer will contain a real and an imaginary part, a weight (or perhaps noise estimate), time stamp, some way of determining its correlator element, and perhaps some direct way of determining its receptors. An Interferometric Yeg may also contain u,v and w coordinates although these are examples of attributes which might be calculated "on the fly". The split between what belongs in a **Yeg**, and what should be determined using the **TelescopeModel** is discussed later, but may not always be obvious.

All **Yegs** contain a reference to their parent **Telescope** object. A **Telescope** object embodies a framework which relates the Yeg to the sky. This framework is embodied in references to two sorts of objects, a **TelescopeModel** and an **ImagingModel**. The intelligence for interpreting a **Yeg** (*e.g.*, how to calibrate a **Yeg** or how to form an image from a set of Yegs) resides in these models. The **Telescope** object does not attempt to describe a real telescope – it merely references models of how we believe the telescope behaves. The actual models used will vary according to the actual telescope, the observing modes, and the astronomical requirements. The models may be very simple or as sophisticated as required. For example, a simple **TelescopeModel** may simply include the antenna-base gains for each antenna or **Receptor**; a complex VLBI model would include many more effects such as polar axis motion. Similarly the **ImagingModel** might be a conventional two-dimensional Fourier transform method, or a three-dimsnsional Fourier transform for wide-field imaging.

A complete observation database will be described by a set of **Yegs** known as a **YegSet** (the measured data) and the associated **Telescope**, embodying models of how these are related to the sky.In this sense, a GBT observation would have a single **Telescope** object, as would a VLBI observation or a 12 hour run by the VLA, *etc.*

Telescope 2/13/92 13:00



5 Telescope Correction/Calibration Models

TelescopeModel is used to encapsulate all relevant information about the model of the components if the telescope, mainly thosed required for correction and calibration. A **Telescope** refers to two **TelescopeModels**: the default and the current. The former is used to contain initial information (*e.g.*, as read from tape), and is "read-only" while the latter contains the model which is actually to be used, and which may be modified after initialisation using the default or some other model. **TelescopeModels**, are objects in their own right, and other models may be attached to a **Telescope** as desired, thus allowing, for example, **TelescopeModels** calculated during intermediate stages of processing to be preserved and reused.

TelescopeModel has services for correction of **Yegs** (**CorrectYeg**) and for updating and improving the **TelescopeModel** given measured and predicted **YegSets** (**SolveFromYegs**). Summarising, **Yegs** and **TelescopeModels** interact in a number of ways:

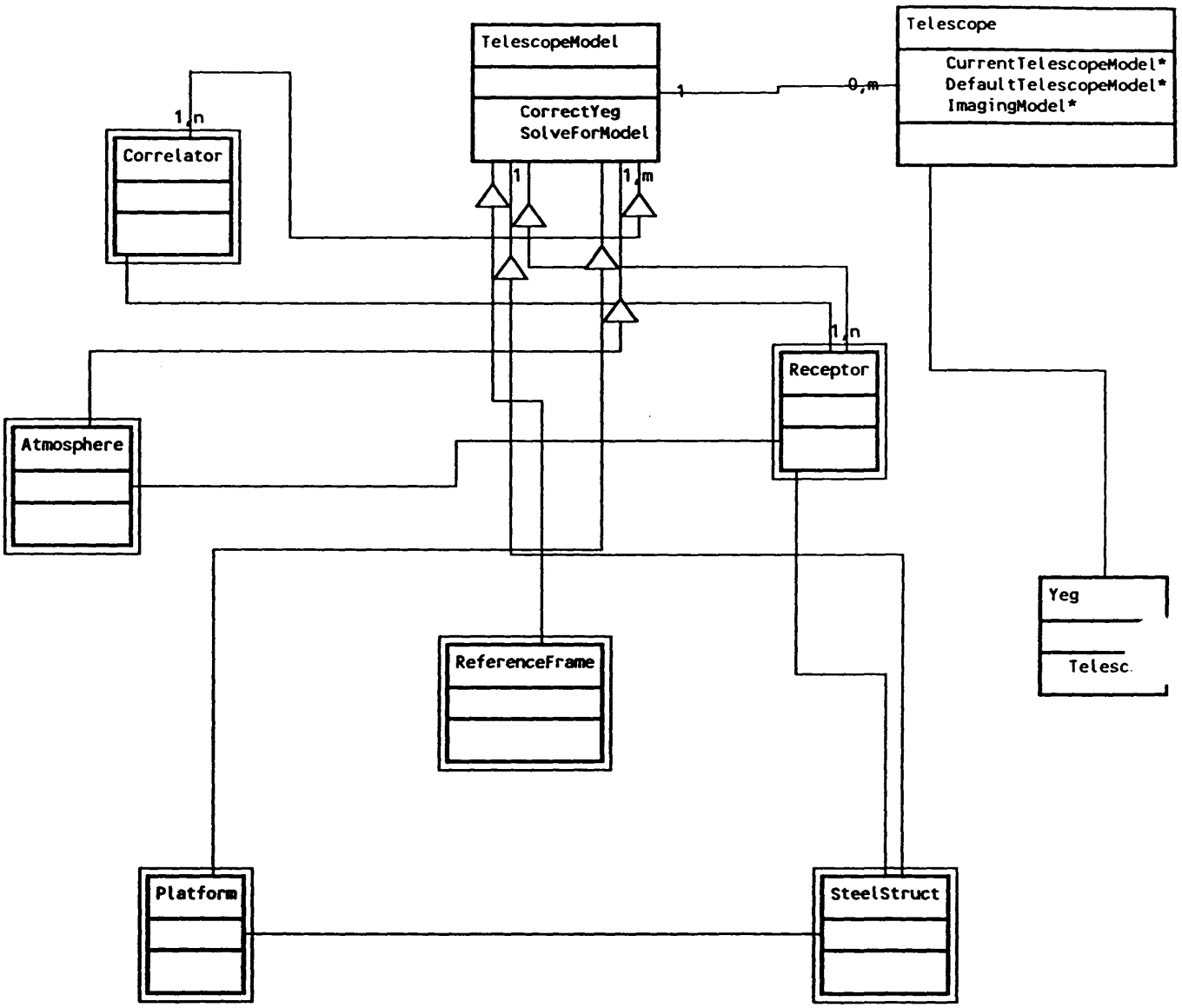
- Yegs, predicted Yegs (see ImagingModel below) and the TelescopeModel are used together to perform calibration solution and correction.
- Yegs contain things such as data values, u, v, w coordinates or pointing coordinates which are generally changed by the application of corrections.
- Correction/calibration solutions may be distributed among things like **Receptor and Correlator** - which are themselves part of **TelescopeModel** - using their solution services to perform updates.

Many **TelescopeModels** will contain parameters and functional forms which are clearly associated with the hardware components of a "real" telescope, together with other items which affect the measured **Yeg**. These might include antenna structure, RF and IF systems, the correlator and the atmosphere. The form of the **TelescopeModel** and its reference to the receptor or correlator objects for information will depend on the nature of the Telescope and the goals of the observations. The solution methods for updating the **TelescopeModel** will also vary enormously in complexity.

It is likely that all models would have additional services for printing and displaying parameters such as gains, or at least, for making these available to appropriate applications. It may also be useful to allow these parameters to be modified directly, and not simply by using the solver services.

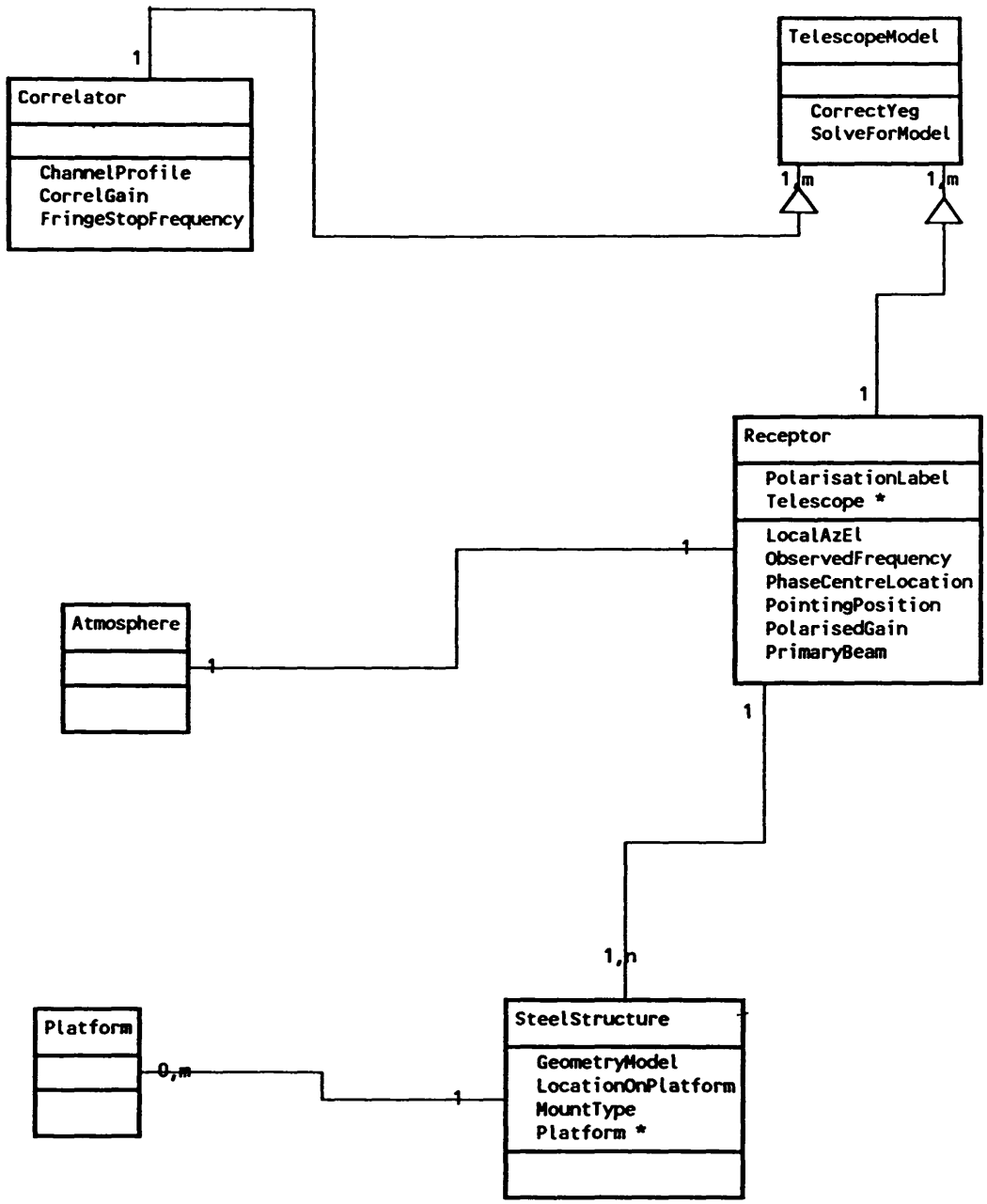
File telemode.oot Printout Date 92-02-17 Time 08:39:49

Telescope Model 2/13/92 13:00



File receptor.oot Printout Date 92-02-17 Time 08:19:56

Receptor 2/13/92 13:00



6 ImagingModel

The **ImagingModel** is a means of connecting a specific observation or observing mode (*e.g.*, VLA or MERLIN observation) to a method which is to be used in making images from **Yegs** associated with that observation, together with the inverse operation for predicting **Yegs** from a model or estimated **Image**. The purpose of this is to provide a flexible means of switching between the various methods for a given observation, and to encapsulate all information about the relevant measurement equations in one place. A telescope forms an image which we will represent by a vector $\mathbf{x}$ which can be described by a linear equation $\mathbf{y} = A\mathbf{x}$ where $\mathbf{y}$ denotes the observed, calibrated Yegs. This linearity holds for most cases currently of interest, but is not crucial. An interferometer without primary beam correction has elements of A which are simply complex phasors whereas for a single dish the elements correspond to points in the primary beam. PredictYegs thus corresponds to calculating $\mathbf{y} = A\mathbf{x}$ whereas InvertYegs corresponds to performing the transpose operation $\mathbf{x} = A^T(\mathbf{yw})$ where the weights $\mathbf{w}$ can be chosen using CalcWeights.

Note that the scheme should cope with other inversion schemes which are specific to a given type of **ImagingModel**, but where the transpose relationship does not apply. So long as **PredictYegs** and InvertYegs exist, any other methods can be attached to the **ImagingModel**.

To treat two telescopes as the same (e.g., for making uniformly weighted images using data from multiple telescopes such as MERLIN and the VLA), each must use the same instantiation of the ImagingModel; this might look something like:

```
// Instantiate imaging model.
IntImagingModel intim(FALSE,FALSE);

// Use the same imaging model for the MERLIN
//   and VLA Telescopes.
VLA.ImagingModel = intim;
MERLIN.ImagingModel = intim;

// Perform the inversion on a mixture of data.
Dirty = intim.invert (SomeYegSet);
```

Here, **IntImagingModel** is a class derived from **ImagingModel**, which applies for interferometers. It requires some parameters for initialisation (*e.g.*, is the model to include primary beam effects?). This allows the **Yegs** from the two different **Telescopes** to be recognised as requiring the same treatment, whilst any others would require a different model.

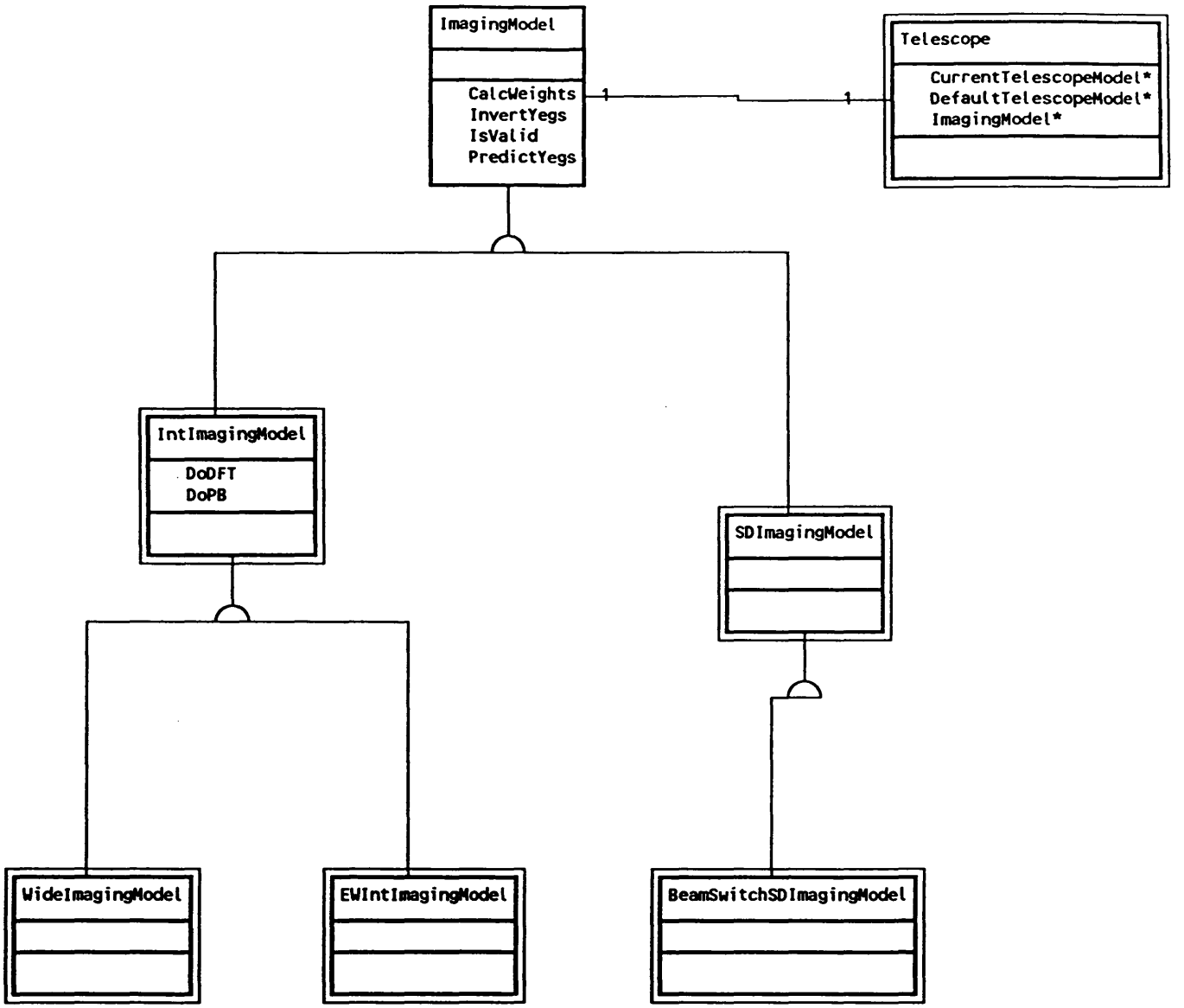
ImagingModels are typically used by a higher level application such as mosaicing methods. One makes appropriate instances of **ImagingModels** and connects them to the **Telescopes** which require them:

```
// Instantiate imaging models for interferometer and single dish.
IntImagingModel intim(TRUE,FALSE);
BeamSwitchSDImagingModel bssdim(FALSE);

// Use appropriate imaging models
VLA.ImagingModel = intim;
GBT.ImagingModel = bssdim;

// Instantiate image tool...
MosaicTool mt(100,10.9,11.6);

// ...and use it.
SkyImage Result = mt.Estimate(YegSet);
```

In this example, **BeamSwitchSDImagingModel** is a derived class, for single dish beam-switched experiments. We assign these two imaging models to a VLA and a GBT **Telescope**, respectively.

File imagingm.oot Printout Date 92-02-17 Time 07:49:19

Imaging Model 2/13/92 13:00



7 Image Tools

ImageTools are used to contain (a) algorithms which do something to images, and (b) the parameters which must be remembered. Rather than derive a bloated set of descendents of Image for keeping results of operations on images, we chose to define a class ImageTool which is pointed to by ImageTool* in Image, and which contains parameters as attributes and the algorithm as a service. ImageTool must be self-identifying via the ID attribute.

As an example, to use an ImageTool such as MEMDeconvolver, first instantiate the tool with the parameters as arguments to the constructor:

```
MEMDeconvolver mem (1E6, -0.00156, 1.0, 100) ;
```

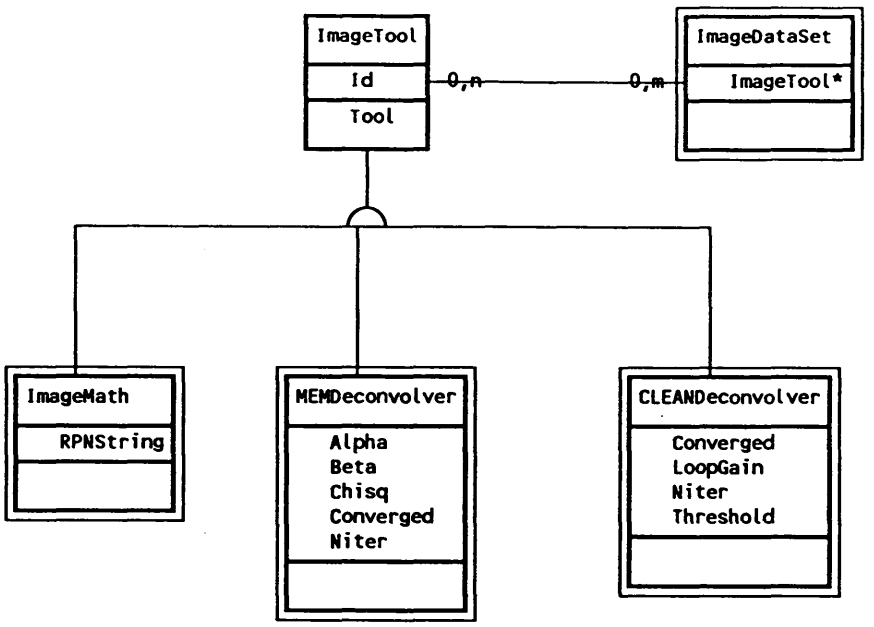
It then can be used by invoking the Tool service:

```
MemImage = mem.Tool (DirtyImage, PSF, DefaultImage) ;
cout << "Alpha is " << mem.alpha << ", Beta is " << mem.beta << eol ;
if (!mem.converged)
{
  cout << "Not yet converged: doing 100 more iterations" << eol ;
  mem.niter += 100 ;
  MemImage = mem.Tool (DirtyImage, PSF, DefaultImage, MEMImage) ;
}
```

Note that we obtained the parameters from **mem** directly and were able to continue by incrementing **niter**.

File imagetoo.oot Printout Date 92-02-17 Time 07:46:57

Image Tool 2/13/92 13:00



8 Data System

The Data System provides the means of handling the contents of **DataSets**. How it handles **DataSets** themselves is not yet defined. We merely sketch an application interface for selecting **Yegs** and various aggregates thereof, wherby the required functionality is attached as methods to aggregates of data, permitting new selections and aggregations to be made.

A **DataSet** is a generic term used to cover any sort of data that is both complete and potentially permanent. A specific example is a **YegDataSet**, others could be **ImageDataSet** and **TableDataSet**. A **DataSet** might be a file, or it might be a view into a file, but this is not defined at present.

A **YegDataSet** contains a number of **YegSets** and **Telescopes**. A **YegSet** is an arbitrary collection of **Yegs** which may be associated with more than one **Telescope**. It can be regarded as consisting of a number of arbitrary subsets, each of which can be considered as a **Yegset**.

YegSet objects provide (at least) four services. Three of these are related to selecting subsets and obtaining specific aggregates of Yegs or YegSets in some order, and the remaining one is used to check if the contents of a YegSet matches certain criteria.

Assert is used to check the characteristics of a **YegSet**. For example, a mapping application might only be able to deal with a single pointing, and so it would need to check that the **YegSet** to be processed only contains one pointing.

Select returns a subsection of a **YegSet** as another **YegSet**; whether this is a copy or a window onto the existing **YegSet** is not yet defined, although it is likely that both will be required.

SortYegSet and SortYegs return an iterator (of type **YegSetIterator** and **YegIterator** respectively) that can be used to step through subsections of a **YegSet** and through **Yegs** respectively in some defined order.

The Sort services take as arguments the items that it is desired to sort on and, optionally, the values required. For example, to obtain Yegs in time-baseline order the following could be used:

```
YegIterator it = myset.SortYegs (time, baseline) ;
```

The iterator **it** can then be used to step through Yegs in either time or baseline order: thus **it.next(time)** steps to the Yeg at the next time and **it.next(baseline)** steps to the Yeg for the next baseline. **it.next()** would step to the next Yeg in the order defined by the order of the arguments to SortYegs. Wildcard arguments are used to SortYegs and SortYegSet.

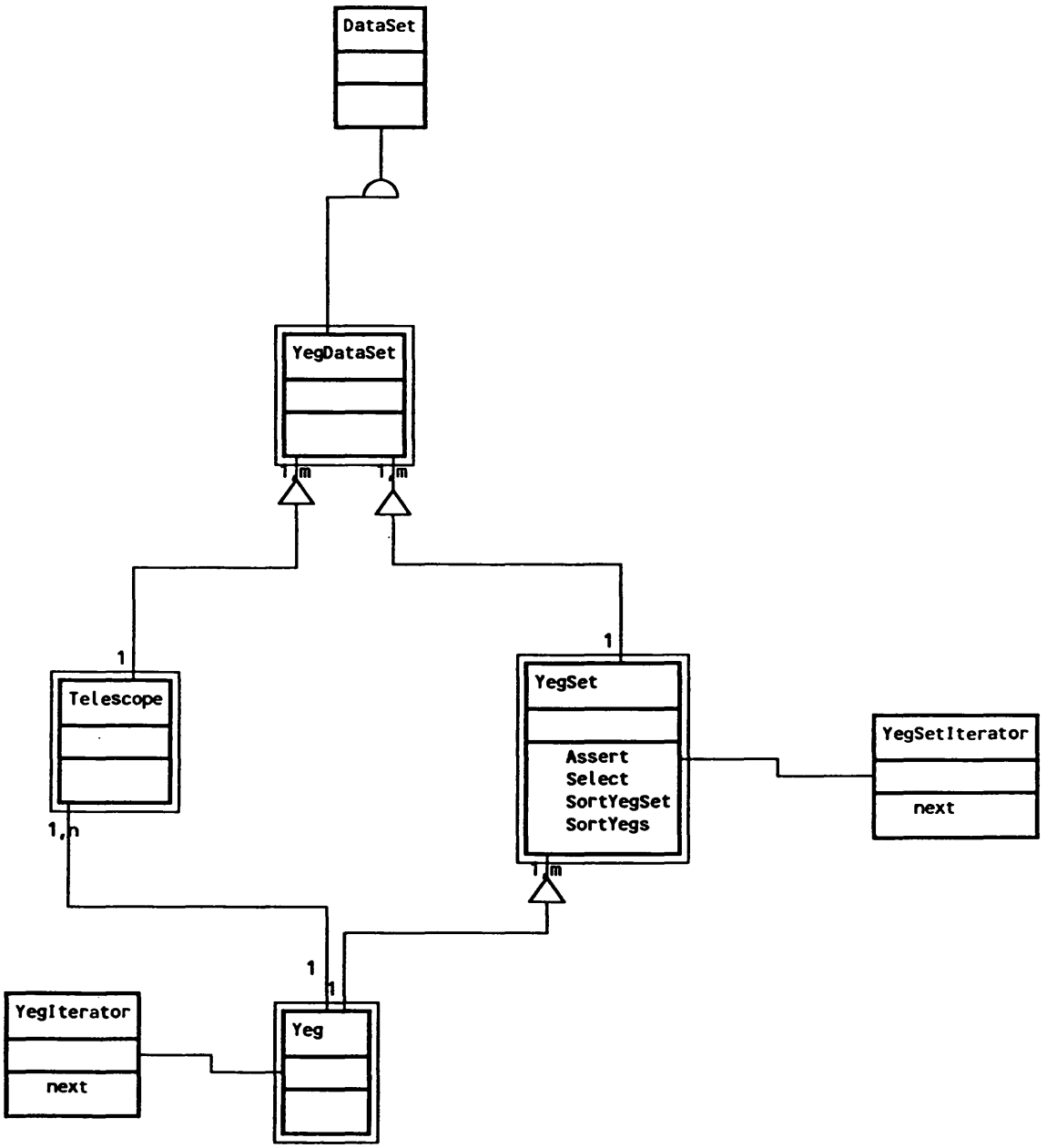
Aggregates are an important way of accessing **Yegs** or **YegSets**; they make it possible for an application to obtain the **Yegs** that it wants in the order that

it wants, irrespective of how they are actually stored. Thus an application doing a self-calibration can obtain, say, all baselines at any one time, while a data-viewing program can obtain all times for any one baseline just by requesting the order in which the **Yegs** are to be presented to them. The ordering is achieved by supplying an iterator that steps through the **Yegs** or **YegSets**; if there is more than one parameter involved (*e.g.*, baseline and time) then the iterator can be used to step through either (*e.g.*, to the next baseline or to the next time).

It may be found that certain aggregates are commonly used, for example spectra or Stokes parameters, and in these cases, standard classes can be provided with standard iterators and possibly other standard services.

File aggregat.oot Printout Date 92-02-17 Time 07:43:57

DataSet 2/13/92 13:00



9 Concluding remarks

In this document, we have described our analysis of the AIPS++ calibration and imaging requirements. Our key decision concerning calibration and imaging in AIPS++ is not to build a very general calibration/imaging model which could be specialised to particular telescopes, but rather to set up a scheme whereby support for a new telescope or new methods of calibration and imaging can be easily inserted. We feel that this provides the best guarantee that AIPS++ will be able to support calibration and imaging in the most flexible and efficient way. Clearly, there will be much to be gained by careful planning and coordination of the development of different `TelescopeModels` and `ImagingModels`.

We accept that our analysis is necessarily incomplete and perhaps incorrect in some parts. There are a number of points that we have left undecided in this document but which must be fixed in any design. For example, questions of whether operations act at the Yeg or at the YegSet level must be deferred until detailed consideration of various applications. Furthermore this analysis/design must be checked against the various user specifications in some detail to see if it conflicts with some requirement. While we have tried to ensure that no user requirement constraints are violated, this deserves closer scrutiny. We have tested our model for calibration and imaging both against simple cases such as self-calibration of antenna gains on a source using the ordinary two-dimensional Fourier Transform relation and beam-switched single dish imaging, and against very complex cases such as pointing self-calibration during mosaicing, and non-isoplanatic self-calibration as will be needed for the GMRT. In all these cases, it passes and leads to a pleasing simplicity. In the more complicated cases, the imaging and telescope models must be coupled so that, for example, the imaging model can incorporate knowledge of pointing errors.

Following the conventional wisdom in object-oriented analysis, we deliberately decided to avoid concentrating upon efficiency issues during the analysis and design stages. We managed to follow this course most of the time, with the exception of discussions of the database system where questions of efficiency lies very close to the heart of any analysis. Further refinement of the proposals here should necessarily concentrate upon implementation concerns.

We have assumed little about progress in other areas of AIPS++. We assume something called SkyImage, which is derived from Image, and which knows about coordinate systems. We have made no demands upon the user interface or task management sub-systems. Despite this decoupling from the user interface, we suggest that some of our concepts, such as ImageTool, would be easily implemented in many different types of interface and form a model for the implementation of the tools requested in some versions of the user specifications.

The Project sub-system could usefully be open to user comments immediately, perhaps from astronomers closely involved with the AIPS++ project but not involved in this meeting.

Simple prototypes of the calibration and imaging models should be quite

straightforward to construct, and can be developed independently of a number of other systems such as the user interface. We suggest that imaging of a simulated data base using Direct Fourier Inversion and a simple CLEAN algorithm should be a first goal, followed perhaps by rudimentary self-calibration. This prototyping will identify and highlight any shortcomings in our model of calibration and imaging.