InstrumentModels for Single Dish Calibration

A Taxonomy of Astronomical Instrumentation

1992 August 13

R. O. Redman and R. Payne

1 Classifying Astronomical Instrumentation for Calibration

An enormous variety of devices have been built to collect astronomical data. To calibrate the data from one of these devices AIPS++ must associate with it at least one InstrumentModel describing how the data was converted from radiation into numbers. Since the only common themes linking all of these machines are the radiation entering from the sky and the numbers exitting to the disk, it is with these that we should start when designing classes of InstrumentModels.

The electromagnetic radiation field from the sky may be characterized for most purposes by the intensity

$$I(\nu, \vec{n}, t, P) \tag{1}$$

where n^ is a unit vector pointing towards a direction on the celestial sphere (cap Omega) and P is a discrete index to the four Stokes polarization states. Note that n^ is actually a two-dimensional quantity, which is most often designated by right ascension and declination. The independent variables in expression (1) will be refered to as the radiative coordinates. This document will attempt to classify instruments which measure the intensity at a set of points in radiative coordinate space. It is believed that the same collection of InstrumentModels should be applicable to an even wider selection of instruments; this will be discussed in a later memo.

For virtually all devices, the output of each channel will be separable in the independent radiative variables frequency, time, direction, and polarization to a good approximation. Thus the signal output from each channel N will be a (probably nonlinear) function of the flux into the channel which in turn may be written as an integral of the detector response over I:

$$\begin{aligned} S_N &= G(F_N) \\ F_N &= \sum_P W_{NP} \int_{t_0}^{t_1} dt \iint_\Omega d\omega\, b(\vec{n}) \int_0^\infty d\nu\, p_N(\nu)\, I(\nu, \vec{n}, t, P) \end{aligned} \tag{2}$$

where p(nu) is the sensitivity profile of the channel in frequency, b(n^) is the beamshape, and W_NP is the weighting of each polarization state in the final signal. For many instruments, the signal S_N is directly digitized. We may classify these devices as "first order" systems, because each output number will come from a single detector element. First order systems may be distinguished from second order systems which combine the signals from two detectors to produce their numerical output. Examples of second order systems are interferometers and some forms of polarimeters. In principle, third order (closure phase) and fourth order (closure amplitude) devices could exist as well, but in practice the higher order quantities are usually generated in software from the second order numbers.

For second order detectors it will often be useful to describe the correlator quite seperately from the individual detectors, giving rise to distinct ReceptorModel and CorrelatorModel objects. This

distinction is rarely useful for first order systems where the detector often digitizes its own signal or immediately passes its signal to a backend which will do the job for it. Since interferometers and polarimeters lie outside the authors' experience, they will not be discussed further in this memo which will concentrate on first order systems.

An instrument will normally output its data as a set of arrays of samples spaced along one or more of the continuous coordinate axes, with the other radiation properties being held constant for all of the samples in each array. The entire set of arrays constitutes one "Yeg" from the device, but for our immediate purposes we will concentrate on a TelescopeComponent describing a single array within the Yeg, building more complex devices from aggregations of simple components. The cleanest distinction between devices is the dimensionality of this array. Photometric time series and spectra are examples of one dimensional arrays. Electronic cameras such as the SCUBA submillimeter bolometer array or a 2048x2048 CCD are examples of two dimensional arrays. Each of these devices needs TelescopeComponents to describe how to calibrate the intensity and the position of each array element along its coordinate axes. The non-varying properties for each array should be labelled in the final calibrated Yeg; it is the responsibility of the coordinating TelescopeModel to supply these values if they are not already part of the raw Yeg.

Quite largely, the calibration of a Yeg using a TelescopeModel can be described without any detailed knowledge of the contents of the Yeg. This clean seperation of Yeg from TelescopeModel is broken in one critical aspect. The model chosen to represent the data, an array of numbers representing the flux in a set of channels, does not carry any information about the radiation properties of each channel. Calibration involves two distinct processes: conversion of the raw signal into true estimates of the intensity, and the addition of information (i.e. a TelescopeModel) describing the radiation properties of each channel. In principle, this TelescopeModel can be added nondestructively to the Yeg. It will be used in the presentation of the contents of the Yeg, but will act only indirectly on the data itself. By contrast, the whole point of the flux calibration is to modify the data itself so that it represents the intensity of radiation as accurately as possible. Rather than overwrite the raw data (**VERY** dangerous!) a new Yeg should be generated to hold the calibrated data plus the TelescopeModel. Since not all observations require an intensity calibration, the new Yeg should be generated by the system before the intensity calibration is attempted.

With these considerations in mind, a generic InstrumentModel is presented in Figure 1. Each of the qualifiers attached to the GenericInstrument corresponds to one of the radiation coordinates and selects a coordinate subcomponent which describes the behaviour of the instrument in that coordinate. For instance, the frequency qualifier should select a suitably parameterized sensitivity profile for each channel; suitable parameters would probably be the center frequency and channel width in this case. Calibration of the instrument would consist of setting values for all of these parameters. It is useful to note that the GenericInstrument, although it was motivated by Equation (2), is more general than the equation, and with suitable definitions for the coordinate subcompo-

nents can handle a very wide variety of instrumentation. Understanding the structure of a generic instrument also goes a long ways towards defining the contents of a Yeg for single dish astronomy; this is a large topic which will be dealt with in a later memo.

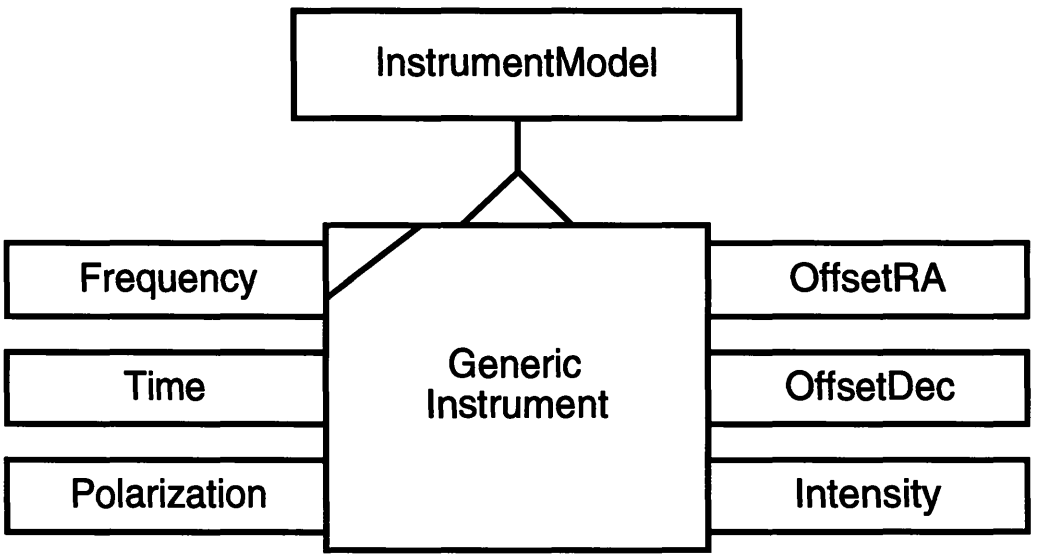

Figure 1 — The Structure of a Generic Instrument

The following sections will consider the instrument components needed to build a detailed InstrumentModel. Rather than attempt a comprehensive description of all possible models, each section will offer a representative collection of components which should handle most of the important cases and which can be easily supplemented as new cases arise. Also, the focus will be on the parameterization of the coordinate subcomponents, rather than on their functional representation in radiation coordinates. The functions are likely to be very simple in the general cases and only develop an interesting complexity for particular instances, which may safely be left to later developers. Never-the-less, it should always be born in mind that a "center frequency" or a "coordinate offset" actually refer to parameters in a sensitivity profile and a beam shape model respectively.

2 Mathematical Classes to Describe Instruments

To describe a coordinate axis for one of the arrays, it is necessary to provide a function which accepts a channel number and returns an axis value. The most direct means of doing this in a TelescopeComponent is to provide the component with an operation() which takes a channel number (a two-dimensional channel number if the array is two-dimensional) and returns the value of the axis at that channel. Figure 2 describes a family of TelescopeComponents called FuncOfChan which would provide the commonly needed functional forms and can be extended easily to include new functions. Only the one-dimensional family is shown in detail in the figure. The two-dimensional

analogues are easy to imagine, and it would be extremely easy to add a third dimension to the scheme if it should prove necessary.

An important facility not indicated in Figure 2 is an inverse function mapping the function back into channel number. It is easy to see how this might be done for the simpler FuncOfChan classes, although issues of existance and uniqueness are nontrivial even here. It is not at all easy to see how to invert a ComplexFuncOfChan, and it would probably be necessary to invert two FuncOfChan2D's simultaneously to find a unique solution. This issue is too important to ignore, but too complex to address properly in a concept paper such as this. Further work will be needed to decide when inverse functions are necessary and how to provide them in the more difficult cases.

The most common of the FuncOfChan will surely be Linear. The IF of most radio astronomical spectrometers, the sample times in most time sequences, and the pixel positions in Reticon and CCD arrays are all linear functions of channel number. Corresponding to a Linear generalized position is a Constant generalized spacing. For the three examples given above, the Constant spacing would be the channel spacing, the sample time, and the pixel size respectively.

Imperfections in optical devices such as acousto-optic spectrometers and CCD cameras often introduce nonlinearities in the coordinate axes. The class Polynomial is one possible representation of a nonlinear axis. Other possible classes, not illustrated, might include spline functions and trigonometric polynomials.

For some devices, the sampling of the coordinate axis may be too scattered to be represented by any simple, functional form. In this case, a LookUp table may be the appropriate description. For example, a spectrometer used to probe atmospheric lines at millimeter wavelengths provided 128 channels ranging in width from 10 KHz up to 32 MHz. A custom data reduction program was written for this device, when a simple LookUp table could have allowed the data to be reduced with a standard package.

Some properties, such as the weight assigned to each channel, can assume a finite (usually small) number of real values. For these quantities, a PiecewiseConstant may be an appropriate representation, being significantly smaller and faster to process than a LookUp table.

In some cases an expression may be known for the function which is most conveniently written as a `double function(int)`, with or without internal parameters. In order that these cases may be handled in the same way as any other FuncOfChan, the class FuncOfInt provides the necessary conversion. C++ functions which want a FuncOfChan may be overloaded using FuncOfInt to extend their utility to ordinary functions:

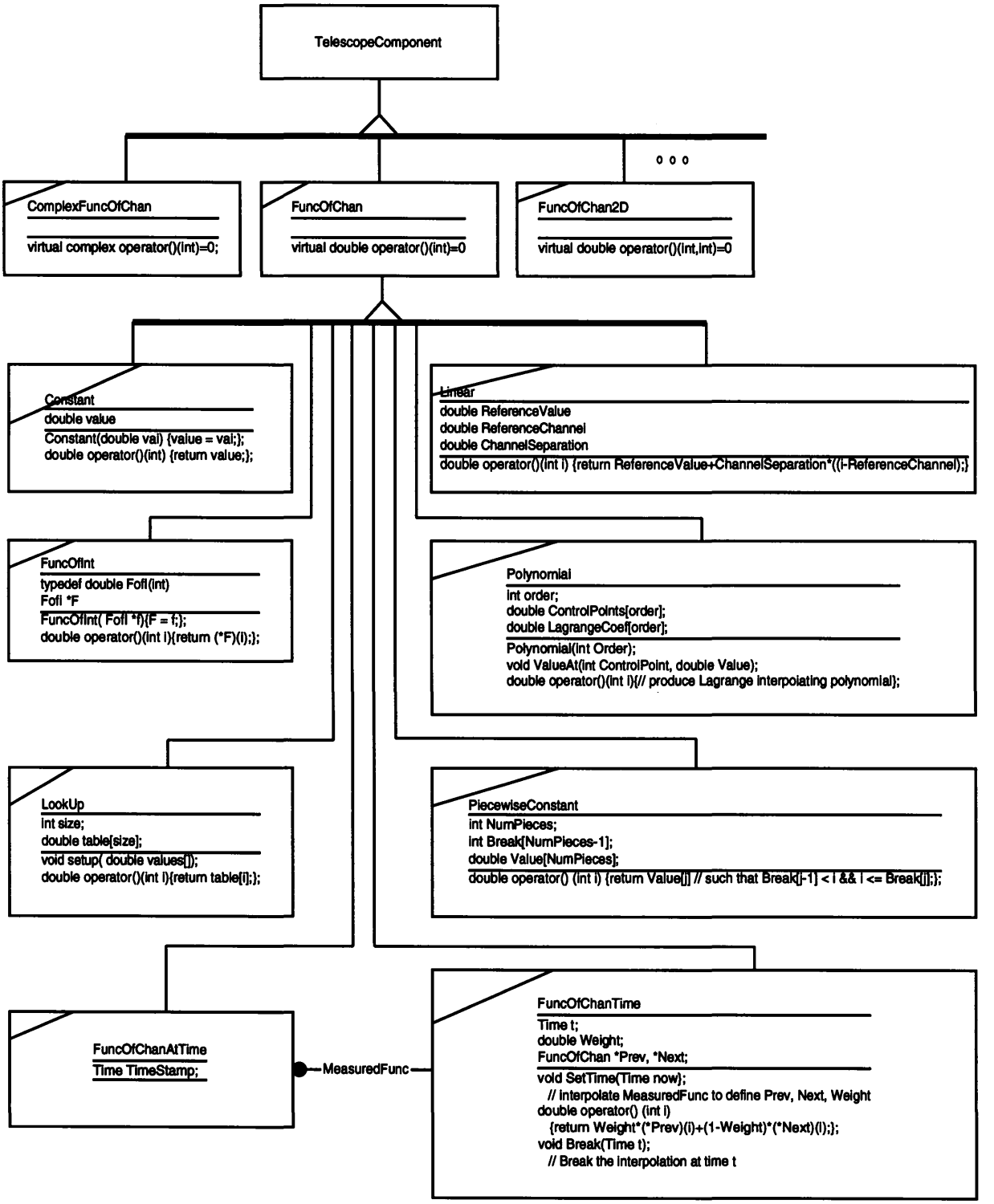

Figure 2 — Mathematical Representations for Coordinate Axes

```
double dosomething( FuncOfChan &f);
double dosomething( double f(int)) {return dosomething(FuncOfInt(f));};
```

A very useful class not shown in Figure 2 would be a dynamically defined function, whose definition could be provided as an ASCII string either from the keyboard or perhaps as an entry in the header of the YegSet. This string would be parsed into tokens and semi-compiled by the class constructor so that it could be executed rapidly. The necessary tools to define such a function should be available almost for free from the user interface. This kind of function could replace all of the simpler classes in Figure 2, and would allow the YegSet to be nearly self-documenting, since the functions defining the behaviour of the coordinates could all be written explicitly in the header of the YegSet.

Many properties of a TelescopeModel are slowly varying functions of time which may affect the signal in each channel in arbitrary ways. The most common calibration process will be to measure these properties at particular times and to estimate the corrections for the remaining data by interpolating in time between the measured values. For times before the first calibration measurement or after the last, the value at the nearest calibration should be used, i.e. constant extrapolation rather than linear extrapolation. As shown in Figure 2, the class FuncOfChanTime provides this functionality. It is intended that FuncOfChanTime should store the measured FuncOfChans in a time-ordered list, and that the pointers Prev and Next should be used to store the most recently accessed place in the list. This should allow rapid access to the required values for most purposes. Note that FuncOfChanTime is intended to be accessed potentially many times by channel for each change in time. If no measurements have been entered into a FuncOfChanTime, it should return a programmer specified value entered through the constructor; this should be a 0 for additive functions or 1 for multiplicative functions. Finally, it should be possible to insert a "break" into the interpolation, i.e. at a specified time the interpolation makes a discontinuous jump and on either side of the break is constant and equal to the next measured value.

3 Flux Density Calibration

Virtually all astronomical observations are intended to measure the flux of radiation into the detectors on the telescope. This is one of the few universal properties of all astronomical detectors; every top level TelescopeModel will have an InstrumentModel defining the flux calibration, even if subsequent processing reduces the data to a nonlinear property of the flux, such as the closure phase or a polarization position angle.