AIPS++ NOTE 164: Layering of Tables and Math Operations in AIPS++

M.A. Holdaway 13 January 1993

I think some simple rules for the layout of astronomical methods should be defined for AIPS++. I will begin by describing what SDE does, paying attention to why SDE is written that way, and then extrapolate to AIPS++.

The Contexts

Consider the AIPS++ TelescopeModel (TM), consisting of several TelescopeComponents (TC). For the time being, we will only consider a single TC, the ReceptorGains. How should the methods be layered in this case?

Consider simple operations on Images such as adding one image to another. What will the layers of code look like?

SDE's Solution

In SDE, the code to solve the ReceptorGains would be layered like this:

Astronomical Structures

The top level code has very simple arguments largely due to the fact that the data are contained in structures which are essentially passed by name (pointer). The very top level of code performs other astronomical level function calls and/or converts the data structures into arrays and passes them to the next lower level of code.

There are two reasons why the top level of code mainly converts from structures to arrays:

* In SDE, the arrays in the data structures can be accessed by an index, but not directly by an array, making it clumsy to deal with. However, the index can be converted into an array by passing a dummy array with that index down to the next deeper level of code.
* It is conceptually elegant to split up the different levels of thinking about the problem into different levels of code: one level deals with the data structure, the next level deals with the lower level details.

The first reason is not applicable to AIPS++.

At this level, the data have astronomical meaning.

Data Arrays

The middle level subroutine takes the data in the arrays and creates matrices which will be the victims of pure mathematical operations. For example, if the solution interval consists of several data intergration times, and some data is not present, we need to further average the data and accumulate the weights considering the missing data and then pass the arrays containing the cleaned up data to the next lower level.

At this level, the numbers still have physical meaning.

Mathematical

The bottom level code is pure mathematics. This level will be made up of linear matrix methods, nonlinear solvers, maximum entropy engines and the like.

There is no physical meaning to the numbers at this level, no units or "measures" are applicable here.

The arguments passed into top level routines are very simple, usually consisting of just a few astronomical data structires. This makes the high level routines easy to use without knowing much about them. The next lower level routine is passed scalar and array arguments, and usually requires a somewhat deeper understanding of what is going on in order to succesfully use the sunroutine. The interface to the lowest level may be simple or complex, depending upon the operations and structures which are involved.

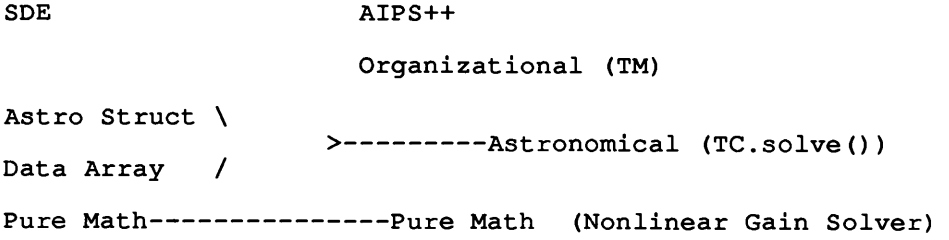
In image processing, SDE has three levels of code: Image level, Array level, and Pixel level. The Image level is concerned with astronomical concepts such as coordinates and pixel units, the Array level is concerned with array conformance, and the pixel level operations do the actual work, treating N dimensional arrays as if they were vectors and performing the mathematical operations pixel by pixel.

AIPS++ Solution

The situation in AIPS++ is a bit different than in SDE for two reasons:

* because C++ is more powerful than the Fortran with a C DataBase used in SDE, some division between layers may not be required in AIPS++.
* because AIPS++ is more ambitious than SDE, and because it will be a production environment, we require a higher level than SDE's top.

The SDE and AIPS++ layer structure for the ReceptorGains compare like this:

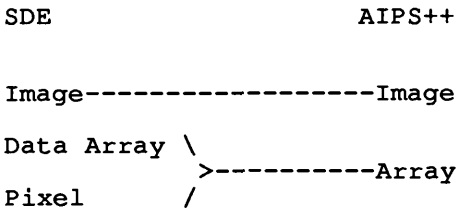


The organizational level represents going through the TelescopeModel to perform the solve. This allows the TM to record its own history and the order in which the TelescopeComponents have been solved.

The interfaces to the top level methods are based on astronomical objects and are therefore quite simple. At the bottom level, one will usually be passing arrays and scalars to the mathematical methods. In the particular case of the ReceptorGains, control parameters and matrices of complex and floating point numbers need to be passed in, and a vector of complex numbers needs to be passed out. It is possible to create a special purpose object to encapsulate the inputs and outputs, but the utility of this is unclear. One doesn't have to turn everything into an object.

One could argue for a four level scheme in AIPS++ in which a distinction was made between the astronomical structures level and the data array level; actually, one would pass TableVectors to the data array level code. This may clarify the logic of the code and force a common structure upon developers, and would insulate the functional astronomical code from the organizational database code (remember, Tables might not be with us forever). On the other hand, separating out the middle level astronomical code from the database code would result in many more methods in each astronomical class as well as more complicated interfaces to these middle level methods. It is unclear if a three or four layer approach is best, but this should be decided pretty soon.

The SDE and AIPS++ layer structure for Image operations compare like this:



An AIPS++ Image has an array. The Image level image processing code will be very similar to SDE's. However, SDE's Array and Pixel level routines can be combined into a single level of code, as in ArrayMath. This is possible because of the use of templating and the because iterating through an array pointer is kosher in C++, but is seen as ugly in Fortran.