Subject: Comments on Brian's note
Date: Mon, 9 Mar 92 09:28:04 EST

The Green Bank document glosses over many things -- it was not self-contained. A large amount of verbal explanation needed to go along with it. It was felt that an incomplete document, produced quickly, would be more useful than a slower appearing, definitive document -- especially seeing the ideas were still under much revision.

Brian's note brings up some valid misgivings about the type safety of the system proposed at Green Bank, and the utility of manipulating data at the "Yeg" level.

Persistence Issues

I suspect Brian is correct in saying that we will rarely manipulate data at the Yeg level, but rather at the IntYeg (interferometer Yeg), etc level. A Yeg is close to a abstract base class. Its base methods would probably consist of little more than data selection and persistance, as well as various "administrative" methods (history files, etc).

While an IntYeg (for example) may well be a better place to start, this does not solve the type-safety problem. There will be widely differing telescopes with different calibration models and (to a lesser extent) imaging models. We cannot prespecify all the possibilities, and for the sake of flexibility, administrative sanity and autonomy, individual sites must be free to develop new telescope models without interfering with the generic aips++ code.

It appears that a true persistent store is is not easily achievable. We cannot easily store the methods associated with a class. Assume we have two tasks, one solving for the calibration corrections, and the other applying them. If the solver produces a WSRT_IntTelescopeModel, how does the corrector know to instantiate a WSRT_IntTelescopeModel when it gets the objects from the persistent store? What if the corrector was compiled before the invention of the WSRT_TelescopeModel? Similarlyly if some smart researcher decides to write a special telescope model for his/her particular observation, how are all the aips++ tasks going to instantiate the correct type of telescope model? Dynamic linking is one possibility. Another is to have a telescope model as an independent (server) process (the act of instantiating a telescope model from the persistent store would invoke a server process, whose name would be in the persistent store, and pass this server some handle, into the persistent store, of its data members). Of course the interprocess communication overhead would be a disadvantage of this approach. A hybrid approach would be for the code which instantiates the telescope model, etc, to know how to instantiate directly the ''standard'' models, and to only invoke a server process when needed. We could probably partially automate the steps needed to "install" a new ''standard'' model. Shareable libraries will be a great help here.

Smart Yeg vs Dumb Yeg

One issue that we flip-flopped over at Green Bank several times was how smart was a Yeg. How much did it know about itself? Part of the problem was that the calibration process can potentially tell a Yeg that it was mistaken about itself all along. Similarly how much does a telescope know about the data that it produced? Is it so smart, that the data itself becomes redundant! Where does monitor data go? There is fuzziness about what goes where. There is no clear cut distinction between what belongs with a telescope, and what belongs with the data.

As a concrete example, a pointing offset would probably be stored with a Yeg in a mosaiced experiment, whereas it may well be stored with the telescope information in a single pointing experiment.

However we believed that separation of the telescope and the Yegs was useful. Both have useful existences in their own right. Additionally we wanted a base data class (Yegs) for the sake of the database system, which was independent of the data stored in a telescope.

Where I have talked about "telescope", I have meant some data and methods which describe how a particular telescope was set up and how it functioned for a particular observation. I do not mean the Green Bank "Telescope" class. This information, in a real sense, is a model. We are

not storing the real telescope! As a model, it was convenient to store this information in an object of the same base class as the object used in the calibration process -- a TelescopeModel object. The information, then, which described the setup of an observation was thus stored in the Default-TelescopeModel.

What to do about the fuzziness of where to store information about an observation -- does it go into the Yegs or the Default-TelescopeModel. The Green Bank approach was for the Yegs and the Default-TelescopeModel to be in close cooperation. A Yeg may well delegate responsibilities for some operations to its associated Default-TelescopeModel, and consequently a Yeg needs to know how to find the Default-TelescopeModel. A Yeg and its associated Default-TelescopeModel should be firmly tied together. The Yegs pointer to the Telescope class, and in turn the Telescope class's pointer to the Default-TelescopeModel were the Green Bank solution to this strong tie. I am not sure that this is the best way to make this tie, but the tie must be done at a low level (because the Yeg and Default-TelescopeModel delegate to each other).

In the Green Bank approach, the Telescope class was merely a means of tying together the Yegs, the Default-TelescopeModel, and to associate a TelescopeModel and ImagingModel. Although we did not harp on it, I think it was always assumed that prior telescope models, etc, would also be pointed to by the Telescope class. In this way, the Telescope class differs little from Brian's Associators (?).

BobS