Subject: Delayed reaction to Green Bank
Date: Sat, 7 Mar 92 18:50:26 EST

1. Introduction

I have become increasingly convinced after discussion with group members (particularly Dave Shone and Bob Sault), and visiting speaker James Coggins, that the "Green Bank" approach is somewhat unsuitable for implementation in C++. My thoughts are also influenced by what I perceive as common problems the "UVW" and "Imaging" groups have run into in our current prototyping effort, although this prototypes is small enough so that conclusions from it should be cautiously drawn.

I believe that we have been seduced by the promise of polymorphism; we hope that from the "outside" we can tell a dataset to calibrate and image (deconvolve, ...) itself. Moreover we hope that from the outside we don't even have to know what kind of dataset we have or what sort of telescope it came from or exactly what kind of imaging (or other) model we want to apply. We obviously realize that we will have to code this information inside the class, but we hope to present the application programmer with bland, generic, datasets -- uncontaminated by any knowledge of it's structure or originating instruments.

I believe that this approach could be quite successful in smalltalk with it's very late binding of methods to types. I believe however that this approach is somewhat flawed for C++; I believe that the Green Bank approach if implemented directly will lead to routine violations of the C++ typing system. While we have to be pragmatic rather than dogmatic, I think that that the alternatives need to be carefully explored before designing this into the system.

I believe that what I propose here will leave the class hierarchies relatively unchanged from Green Bank; it mostly affects the (implicit and explicit) connections between them.

2. Summary

I propose that we need to do two things:

1) Have several (or many) explicit interfaces to the calibration (etc) system that depend more explicitly on type than discussed in the Green Bank approach and
2) Create a new type of object (called an associator here, although a better term must exist) to replace the generic pointers (between, for example, the yegset and the telescope).

3. Typed Calibration and Image Tools

I believe that we should not attempt to treat all data alike. I believe that our potential datatypes (Interferometer, Single Dish, Photon list, ...) are divergent enough that the intersection of all their properties (which ideally is all one would want represented in the base "Yeg" type) is too small to be interesting. Or, at least, I have yet to see a list of what these common properties are. My "Field" idea most likely has the same problem. By all means let's define generic Yeg objects, but my suspicion is that the base classes may only contain computational rather than astronomical methods (related to storage, identification, history, etc).

I think our problem essentially breaks down as follows:

- Data
- Telescopes (model telescopes)
- Algorithms which connect (and modify) the above (calibration,imaging,...)

Data types and telescopes are relatively constant. Algorithms are many, variable, and diverse. Moreover, the "casual" astronomer/programmer we are trying to reach is far more likely to want to implement a new algorithm than to create a new data type or to model a telescope. Also, different algorithms are going to take very different types of argument and produce very different types of results. All these points I believe strongly argue for decoupling the algorithms from the data and the telescopes. In Coggins' terminology, they are enzymes (as he also points out, algorithm is a noun).

Thus we may have an antenna-based gain calibration enzyme that takes an interferometer dataset, an interferometer dataset, and some parameters and does its thing. We will probably also have a WSRT specific calibration enzyme that takes only WSRT data and a model of the WSRT telescope and parameters and does something unique. I would hope that we could do most of what we need to do with polymorphism through the interferometer (SD, photon list, ...) interfaces. That is, I believe that the useful layer of generality starts one step below the yeg, and that it should (and probably must) be possible to treat all interferometers largely alike. It will certainly be a lot of work to get this right, and if we don't we will have a profusion of things that work only with the more instrument-specific classes. Writing the Yeg class will be easy, writing the layer below that will be very hard and will require the hard work (I believe this is part of Bill Cotton's argument).

So, to sum up:

1) Decouple Data, telescopes and algorithms
2) Commonality at the "Yeg" level is a chimera, work at achieving commonality one step below that level

4. Associators

Of course in the above I've so far neglected one very important point: Data, telescopes, and the products of algorithms (images, deconvolved images, statistics, ...) have associations with one another that have to be maintained. (The association of objects and the actual algorithms I consider to be history which I don't discuss here). These associations are after all the reason the various pointers and back-pointers were arrived at in Green Bank.

The associator essentially plays the role of an object database (or of the AIPS1 cataloging of extension files). The user or programmer explicitly associates various types of objects with one another. I also assume we will need a versioning system where (for example) multiple gain tables may be associated with a visibility dataset.

I suggest that in general these associations should not be automatically be made by the enzymes, rather that they should only be associated after the user (programmer) has had a chance to inspect the new object. An unassociated object should essentially be considered "uncataloged." (Although it would still have its history associated with it). It is tempting to have the root of this "association tree" embedded in the "project" class identified but not developed in Green Bank.

The exterior interface of the associator class should be typesafe. It should supply upcasts but not downcasts to association queries (on the outside, the inside is apt to be ugly). That is, we should be able to ask for a WSRT dataset as an interferometer dataset, but not as a DRAO dataset.

The mental model I have in my head is essentially a hierarchical database of objects of different type (with multiple versioning at a given node), with matching allowed on class upcasts. More powerful ideas must certainly be possible; however since I'm essentially uneducated about databases (let alone OO databases) I won't attempt to describe such a scheme.

5. Disclaimer

Although these thoughts have been strongly influenced by interesting discussions, they are solely my own and do not reflect any position of NRAO or the aips++ project.

Brian Glendenning, March 7/1992