Note 163: The AIPS++ Environment

Last changed on $Date: 2003/07/16 03:49:05 $

editor: M. A. Holdaway

This note takes a look at the requirements that are to be placed on the processing environment. The first chapter looks at what the user will see (subject to change), the second chapter looks at what application programmers want to see, and the third (currently empty) chapter presents an overall picture of the requirements on the processing environment from which a processing environment design may be generated. The ideas and text are mainly due to Mark Holdaway, Sanjay Bhatnagar, Terri Bottomly, and Bob Garwood.

1 The AIPS++ User's Environment

This chapter explores the different processes which will be visible to the AIPS++ user in both windowing and non-windowing environments and makes explicit the requirements these processes will place on the underlying operating system.

1.1 The Catalog: The first thing I would want to see is a catalog of my data.

What will a catalog look like? First, the catalog will look different from the user's and the software engineer's points of view. From the user's view, the catalog is made up of logical objects, not of files. The first requirement is that the catalog be hierarchical to improve organization. You can name a given node whatever you want. You must be able to dispay what KIND of object something is: a MeasurementSet, a TelescopeModel, a Postscript file, a Table, what type of file. You need to be able to either highlight all objects of a certain kind or display only objects of a certain kind. Since we have an object oriented system, there is no reason to stop displaying the objects when we get down to the top level objects: the constituent objects could be displayed just as the project level organizational hierarchy is displayed. The catalog display should be configurable: we need to be able to control the "depth" with which we are viewing the hierarchical catalog, we need to be able to go up and down the hierarchy, the window should be scrollable, the window should allow multiple views of the catalog, and the display should allow the user to see associations to a given object. The catalog will also provide a handle to the data objects: one should be able to type the "path name" as the input to a program, or one should be able to "click and drag" one of the objects into another AIPS++ window instead.

Most of the functionality addressed above should also be possible in an emacs window from a dumb terminal. Since emacs has very nice multiple buffer capabilities, it provides an excellent "dumb terminal" interface to AIPS++.

1.2 Table Browser: I'd like to see my data now, please.

Select one of the data objects: the gain Table underneath a ReceptorGains TelescopeComponent (antenna based gain table). Click on the "options" button on the menu at the base of the catalog window, up comes a list of options, you select "Table Browser", and another window (probably formerly iconized) pops up. The Table Browser will first show you a list of the columns in the Table with a default list of columns to be displayed. You can change that list if you wish, and then load the Table into a scrolling window which allows you to see the data values. SubTables can also be displayed.

Sometimes, it will be desirable to take 2-D array data which is being displayed by the graphics tool and display the pixel values using the Table Browser.

1.3 Graphics Tool: I'd like to see a plot of that data.

The Table browser will need to have a clear interface with the graphics tool. In the Table browser, you select out data from antenna~N, select on column "time" to be the independent variable and "antenna gain" to be the dependent variable, possibly a few more dependent variables, and issue the "plot" command from the menu of options on the Table browser, and up pops the Graphics Tool (probably formerly iconized), and the vectors are plotted as a function of time as different colors. You have the power now to change the plot: should the values be plotted on different graphs, on the same graph, linear, or linear-log, change the scales, create and move labels, change the display to black and white, plot the vectors with different line styles or different symbols, and make a hard copy. Now you decide there are some problems with the gains, so you flag a few gains from the graphics window (all calibration tables will have a "flag" column), and immediately the Table Browser display indicates that those few gain entries are now flagged. A Flash Box pops up with the message "Table has been altered", with possible click boxes "discard edits" and "save edits". You do not need to respond immediately, other graphics or Table Browser operations may continue until you finally decide to save or discard.

The same graphics tool can also display continuous-color plots.

The graphics tool is also accessible from the command line and from executing tasks.

Some of the line drawing capabilities of the graphics tool should be accessible from dumb terminals.

A variety of outputs should be available from the graphics tool, to be used as inputs for other tasks. The inputs/outputs block model which we discuss here may provide part of this functionality, but it is not a clear match.

1.4 Message Browser: I'd like to see the history on this data.

The Message Browser is a specially configured Table Browser which will not allow you to alter any records, but will allow you to add records (comments and such). The message system is highly configurable (Roberts, 1993). For example, you may want to set up the AIPS++ message handling so that you do or do not get the messages displayed in the command line window, so that you get messages logged to a personal "global" message table, to a "project" message table, and also to message tables associated to individual objects. When the AIPS++ system starts up, the default will be to have a Message Browser viewing your global message table, in which any messages generated by any processes running under your username will be logged in a scrolling window. You may select on all messages logged after a certain time, by a certain task, and by a certain object, displaying only those messages you are interested in. You may then print the messages.

But right now, you are concerned with the processing history of the gains you have been looking at. So you go back to the catalog window, selecting the ReceptorGains object, clicking and dragging the handle to the Message Browser input area, and up pops the message Table attached to the ReceptorGains TC. After you've refreshed your memory as to how you made these gains, you can jump back to the previous message table, or jump back to the default message Table.

1.5 The Command Line Window: Now I'd like to do some processing

OK, you are ready to do something. You go back to the Catalog window and click on a MeasurementSet, and up springs a list of suggested tasks to run. These can be used as suggestions: you type the name of one of these commands at the command line in the command line window; or you click on a task name, and the task name writes itself in the command line window.

1.6 Inputs/Outputs Editor/Browser: I want to set the task inputs.

Typing the name of a task in the command line window will create a new window (or invoke a new buffer if running under emacs) which contains a scrollable text file consisting of input parameter names and input parameter values. The display configuration can be altered so that you also see help for each item, the default value, and the range of valid values, if defined. The visibility of these parameters, the help, the default values, and the validity range are all customized by the particular AIPS++ package you are running (generic, single dish, interferometer, VLBI, etc). You can easily jump around and reset the parameters, dragging handles to cataloged objects if you wish.

You might also want to get the value which another program calculated to use as an input for this program. Since the inputs and outputs blocks are the same format, you can easily access the values created the last time you ran another program. The inputs/outputs editor can have multiple scrolling windows (like emacs). Once you access the value you were looking for, you can select it and then drop it down in the other inputs window. The outputs are "read only". You could also drop the value into a user-created Table under construction in the Table browser. For than matter, you could access this data from the command line and insert it into either the inputs or into the user-created Table from the command line or from a processing script.

When you are happy with your inputs, hit "save" or "save and execute".

1.7 Flash Box: Oops! I didn't know it would take that long!

And, provided you don't have the "batchmode" environment variable set, up pops a small, prominently placed Flash Box with the message "This task will take 1.7 hours" with buttons "proceed" and "abort". You decide to proceed.

1.8 Meter: How is my CLEAN doing?

Up pops a small window, about the size of a performance meter, with axes labeled: iteration, flux. A line is updated every 30 seconds or so, showing how much flux has been cleaned so far. You decide that isn't enough information: you click on the "information available" button and see that you can also plot the residual level, so you do. But you are observing 3C273, so you click on "plot properties" and switch the graph to "log-log". You can gauge the process of your clean, also viewing text messages as they go to your command line window or into your global message table which is viewed from the browser. You see that the clean is done. When you type the next command, the clean meter display will disappear unless you click on "plot properties" and make th plot stick to the screen. Alternatively, you can redisplay this plot using the graphics tool, anotate it, plot it out, or save it as an AIPS++ plot file.

1.9 Help: Help!

There are many kinds of help available. The inputs each come with their own short help summaries. Each type of object comes with a list of suggested tasks and subroutines which will operate on that type of object (configurable by AIPS++ package). More detailed help is available on each of these tasks and subroutines (subroutines are accessible from the command line). Some routines will also have on line help concerning what algorithms are used, and the accuracy and testing which has been performed on those algorithms. There will also be hypertext cookbooks which can be accessed on line, and perhaps even tutorials which will walk you through some aspect of data processing in your AIPS++ package. There must be online programmer help as well, such as high level and more detailed level diagrams of systems and subsystems. Individual help entries must be easily printable.

1.10 Script Editor: So, I want to repeat that on my next 40 sources.

You've blazed a data reduction path which works for your first source, but you don't really want to do it for the rest, so you go into the Message Browser, select on some time range, select on "commands issued" and "inputs", and enter "Script Editing Mode". The script editing mode will allow you to convert your previous steps into a single procedure. You can get data from the outputs of one task and flexibly feed them into the inputs of another task. You can delete commands which were mistakes, add extra commands you would like to perform, "save" (or "save and execute") the contents to a procedure, which immediately shows up in the Catalog, move to the command line, type the procedure, and go home to relax while your computer puts in the overtime.

1.11 Pipeline Canvas:

We must plan for the possible future use of graphical representations of tasks and pipelines sending information from one task to another. Such a pipeline canvas can probably be implemented on tp of the input and output block objects which have been discussed here and elsewhere. An interface between the pipeline canvas and the CLI and or message logging needs to be implemented. Basically, I don't think we've shot ourselves in the foot on this yet.

1.12 How These Windows Are Initialized

- **Catalog**: this window is initialized by the AIPS++ startup script. For a dumb terminal, it is a buffer in emacs.
- **Command Window**: this window is initialized by the AIPS++ startup script. For a dumb terminal, it is a buffer in emacs.
- **Message Browser**: this window is initialized by the AIPS++ startup script. For a dumb terminal, it is a buffer in emacs.
- **Graphics Tool**: this window is initialized by the AIPS++ startup script. Some functionality should be available for a dumb terminal; no functionality is available from a really dumb terminal.

- **Table Browser**: this window is not initialized by the AIPS++ startup, but can be initialized from other AIPS++ windows (notably, the Catalog window or the graphics tool). There may be a Table-to-ASCII translation facility which will allow limited Table Browsing in an emacs buffer.

- **Inputs/Outputs Editor/Browser**: this window is not initialized by the AIPS++ startup, but is instantiated from the

- **Flash Box**: this window will only pop up when an AIPS++ application requires additional input and will disappear after the input is obtained.

- **Meter**: this window is instantiated by individual tasks and generally will disappear after the task is complete unless the user prevents this.

- **Help**: help windows will pop up when required. Multiple help windows should be possible but not the default.

- **Script Editor**: special feature of the Message Browser.

- **Pipeline Canvas**: window is invoked when the user requests it.

1.13 Data Flow Between Subsystems

The various windows which have been mentioned above may be the tip of the icebergs of various AIPS++ subsystems. It is important to understand what communicate is required among these subsystems. Figure~1 shows all of the subsystems mentioned above, plus "printer" and "task". "Task" represents an astronomical task which is executing. Single and double arrow lines indicate when information or objects must be passed from one subsystem to another. The details of this diagram may change somewhat: information may actually pass to subsystem~A to subsystem~B through subsystem~C; or perhaps a more detailed analysis would require that some more interactions were required.

A few obvious points about Figure~1: several subsystems use the Flash Box (usually called a dialog box), many subsystems initiate and send information to the Help subsystem, and lots of subsystems send information to the Printer. Removing these three heavily (but simply) used elements results in the communications shown in Figure~2.

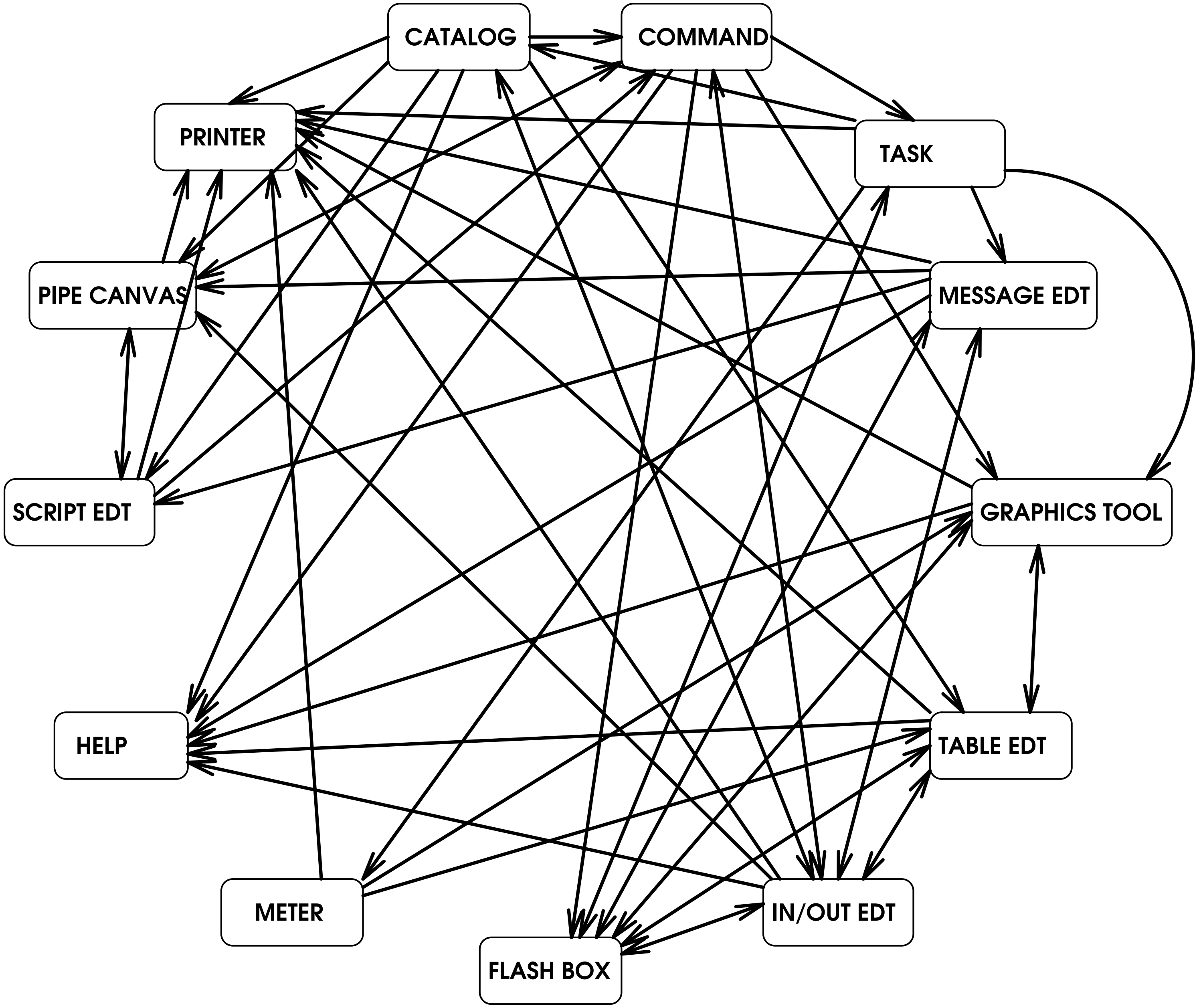

Figure 1 – Subsystems and Dataflow

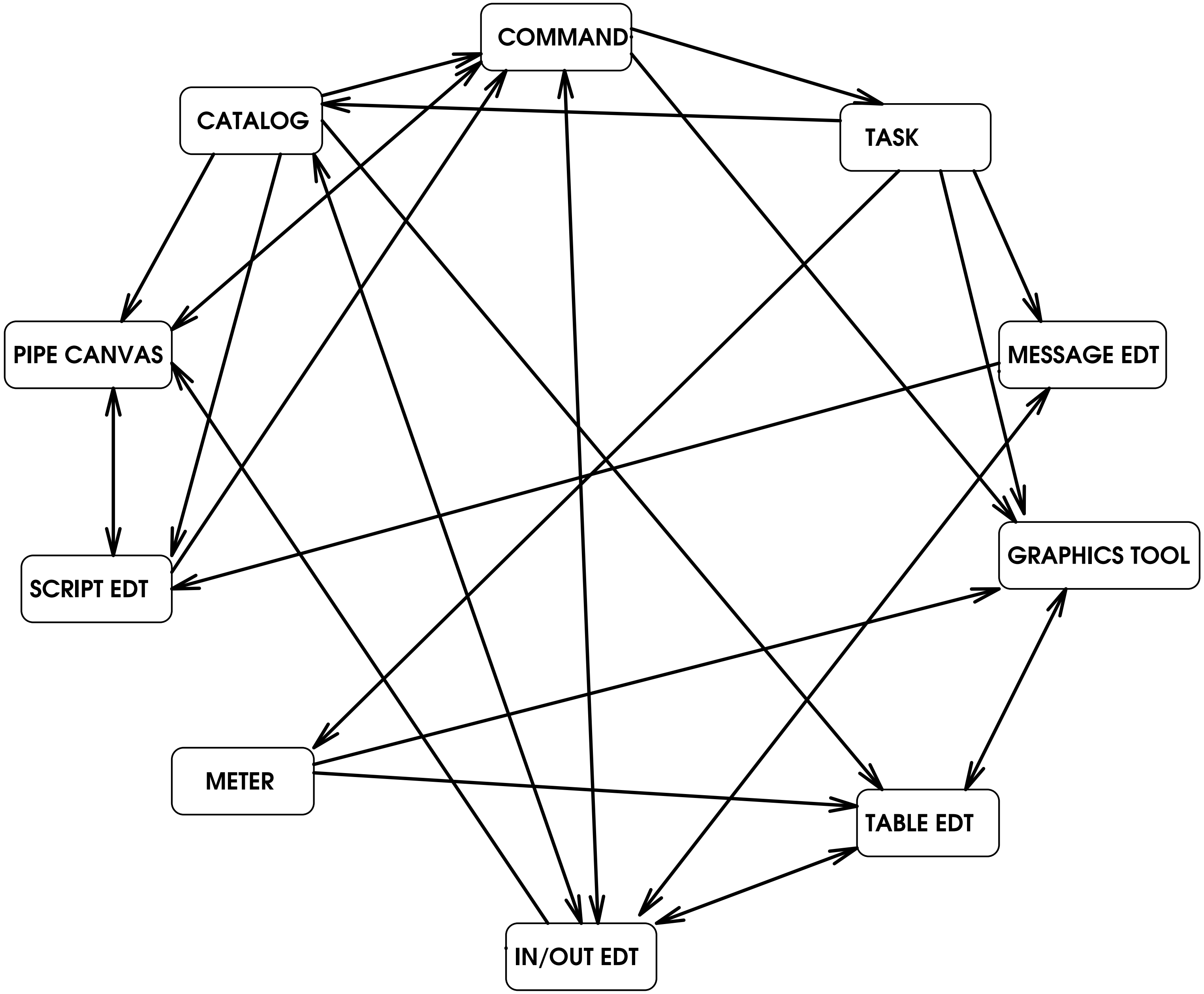

Figure 2 – Subsystems and Dataflow

1.14 Other Issues

- AIPS++ will eventually need to run on multiple machines and parallel machines. My feeling is that we should largely rely upon utilities such as PVM.
- multiple AIPS must run on the same machine.
- AIPS being run on multiple machines may have their display on a single machine. Which windows will need to be doubled?
- some data files will be sharable among different processes under the same AIPS, some will be sharable among many different AIPS on different machines.
- many different aspects of AIPS++ will be configurable under the control of simple ASCII files. These files need to be well documented with good online help. Also, each user should have their own copies of these files. What files are required? Where should they go? Which should typically be read only?

2 The AIPS++ Programmer's Environment

This chapter is produced from the perspective of an astronomical application programmer and user of the aips++ system (Sanjay Bhatnagar). This note, at some places, goes a level deeper into the aips++ acrhitecture than the previous chapter.

The User Interface (UI) is the top most layer of the aips++ system. The application layer makes the next level of layer and it interacts with the UI and the Processing Environment (PE). Tables for storage and mathematical operations underly the application layer (see Note~164 for a picture of some of the processing layers).

. Since the UIs of aips++ are going to "plug compatible", there has to be a fixed protocol between the application layer and the UI so that the applications, as long as they stick to the protocol, never come to know what UI is being used by the user. This protocol also needs to make sure that applications are able to serve all the needs of the user (see Chapter~1).

The application program will potentially have two types of data that the user may like to have a look at:

- Application configuration data
- the "scientific" data

It may also have a service which can give reports about itself (how long it will take to complete the jobs as determined by the configuration data, etc.). The application programmer will have a service available where s/he can "register" all the objects in memory that the outside world can look at. This will include all the configuration data and all the data that the application is going to produce and work on (Images, Visibilities, etc.). The scientific data referred above, will be dynamic in the sense that as the application runs, this data will be changing. Having registered it with the UI, the user from outside can "attach" a tool to the UI to look at the registered data of the application. The UI will talk to the application to get access to this data. What all internal "scientific" data will the aips++ application make visible to the outside world by registering it with the UI, is a matter of policy which needs to be chalked out.

There should be only one contact point between the outside world and the application and the transaction between the two should be managed by the UI (or the Object Broker (OB)). The application should listen to requests from the OB asynchronously and serve the requests. These requests may originate from various different tool that the user may use to use the PE - TableBrowsers, PlotTools, HistoryTool, Meters, HelpTool, etc.

3 A God's Eye View of the AIPS++ Processing Environment

We're still waiting on the mountain.