AIPS++ User Specifications: BIMA Version

1 Introduction

The Berkeley-Illinois-Maryland Association (BIMA) operates the Berkeley-Illinois-Maryland Array (also BIMA), which will be a 9 (or perhaps 12) antenna synthesis array at Hat Creek, California. BIMA is used at millimeter wavelengths, primarily for spectral-line observations. Several years ago BIMA began the development of a new software system, the Multichannel Interferometer Reduction with Image Analysis and Display system, or MIRIAD; approximately 10 man-years of work has been devoted to MIRIAD, and the system is now being used by BIMA astronomers for the complete processing of BIMA data from the telescope to publication. MIRIAD is installed on a wide range of computing hardware, including a Cray 2, a Convex C2, a Multiflow, VMS VAXes, and Sun workstations. The core MIRIAD package provides for on-line inspection of data at Hat Creek, complete data processing from editing and calibration to inversion, deconvolution, and limited analysis, display, and graphics. Two auxiliary packages which work with MIRIAD data sets are MXV (MIRIAD X Visualizer) and WIP. MXV provides X Window visualization of data cubes, with extremely flexible capabilities for display of data cubes and subsets of data cubes in raster and vector form. WIP is an interactive X Window graphics package based on PGPLOT which allows flexible generation of publication quality plots.

It is important to understand the reasons why BIMA, with the usual extremely limited funding of a University-based group, decided to devote a significant fraction of resources to the development of a competitor for the NRAO AIPS system, since those motivations form the basis for the BIMA aips++ user specifications. When BIMA began the development of MIRIAD 5 years ago, we felt that AIPS was inadequate for 4 major reasons. 1) AIPS was not designed originally to handle spectral-line data; BIMA is used primarily in spectral-line mode with a very flexible correlator which generates multiple spectral-line data sets simultaneously. Efficient processing and flexible and powerful display and analysis tasks for spectral-line data sets are crucially important for BIMA. 2) AIPS did not run efficiently on the highest performance computing systems, such as Cray supercomputers, and had fundamental design features (such as extensive use of disk scratch files) which made it virtually impossible to port AIPS efficiently to supercomputers. This limitation significantly impacted the science that can be done with the VLA and would have had an equally detrimental impact on BIMA science. 3) The programmer interface for AIPS was awkward, discouraging astronomers who are casual programmers from adding special-purpose tasks. As a University based operation, BIMA places very strong emphasis on student training, including development of new ideas for data reduction and analysis that require new programming. 4) A powerful, user friendly, flexible user interface was an important feature in the MIRIAD objectives, including support for window-based workstations.

Since BIMA plans to remain on the forefront of interferometer software development, and since the aips++project offers offortunities beyond those available through our solo support of MIRIAD, BIMA will be a participant in the aips++ project.

2 BIMA Specific

It is the intention of BIMA that aips++ eventually replace MIRIAD as the data processing system for the array. We therefore feel that it is extremely important that the 4 areas mentioned above be stressed in the aips++ system. In addition to several (perhaps obvious) BIMA-specific requirements for aips++ data structures and tasks, we therefore here elaborate on the 4 areas mentioned in the Introduction.

- BIMA data structures and tasks. The design of data structures must make it possible for data from the BIMA on-line control computer to go directly into aips++ (perhaps with an interface program specific to the task, but without leaving BIMA out-on-a-limb with a non-aips++ interface). We need the ability to define data items peculiar to BIMA and the ability to add them in the future. Aips++ must have all tasks necessary to carry BIMA data through from the telescope, to editing and calibration, map making, deconvolution, analysis, comparison with theory, display, and publication. The BIMA-specific tasks, particularly calibration, will be written by BIMA itself, but the aips++ system must be designed to facilitate such programming and must never allow changes which cut off an aips++ consortium telescope such as BIMA in a way that requires extensive re-programming.

- Data sets. It presumably goes without saying that the aips++ data structure should be 3-dimensional, with polarization included explicitly although not as a full 4th dimension. However, we argue here for a 4-dimensional data structure. It will often be the case that multiple spectral lines will be observed of the same object, and it will be extremely convenient to treat different cubes as the 4th dimension of the data set. The BIMA correlator can simultaneously produce up to 8 independent (not contiguous in frequency) spectral-line data sets with 256 channels each, so the 4th dimension might originally be 8 units long. During the calibration and other processing of the 8 simultaneous spectra, the same procedures can be applied once and uniformly to the entire data set. Users will want to be able to add additional cubes to the 4th dimension, including perhaps continuum images at multiple wavelengths and spectral-line cubes obtained with different telescopes. When supersets in the 4th dimension of image data are assembled, display and analysis tasks written to take advantage of this 4-dimensional structure will be very powerful and convenient. The MXV ability to handle multiple spectral-line cubes in synchronization is an example of the functionality desired. Finally, the aips++ system should be able to have the ability to deal with large flexible data sets of any image size in the 4 dimensions. Although most computer hardware could not handle such large data sets, the system should allow for it through dynamic memory allocation even for small machines. The largest machines which will be available during the lifetime of aips++ will certainly be able to handle extremely large data sets; we do not want the software written now to limit our future.

- High Performance Computing. Aips++ must be designed to run on the highest performance computer hardware available. Today, these are the vector processor supercomputers such as Crays and Convexes. However, supercomputers with one job per CPU are approaching the ends of performance gains. The future of high-performance computing is in the massively parallel processor, shared memory machines such as the Thinking Machines Corporation Connection Machine and the Intel Sigma. The fact that vector supercomputers were unknown when AIPS was designed is one force driving the NRAO to develop aips++. The fact that massively parallel processor supercomputers were unknown when MIRIAD was designed is one force driving BIMA to join in the aips++ development. The rapid improvement in workstation-class computers relative to much more expensive supercomputers means that most of the jobs that aips++ will be called on to do will be run on workstation-class computers. However, there will always be the requirement for the high-end of the performance curve of computer hardware for jobs including wide-field imaging, large-scale mosaicing, spectral-line observing, improved deconvolution and other algorithms, et cetera. To have aips++ not designed to take advantage of the obvious evolution in high-performance computing would be a mistake of the first order.

- Programmer interface. Aips++ must be designed so that experts can easily use the system to explore new algorithms and add new capabilities to the system, so that aips++ can be easily maintained, and so that casual programmers can easily add special-purpose (perhaps throwaway) tasks to the system. The use of OOP and C++ and careful attention to data structures will go a long way to satisfy the needs of experts. It must be easy and straightforward for casual programmers who know C or C++ to write programs in the aips++ system. However, many astronomers do not know C (let alone C++) and will be unwilling to learn. While experts may lament that attitude as being shortsighted, aips++ must live in the real world of astronomers dominated by FORTRAN programmers. To be successful for the casual programmer, aips++ must allow for integration of FORTRAN programs.

- User interface and user friendliness. The user interface to aips++ and how friendly the system is to both beginner and expert users is equal in importance to the functionality of the software system, for the user interface determines how the users will interact with aips++, whether accomplishment of user goals will be a pleasure or a pain, or even whether the user can do what (s)he wants to do at all. The ability for aips++ to easily and straightforwardly move from raw telescope data to publishable results will determine the success of the project. Clearly, a multifaceted user interface will be necessary. Experts will not want to wade through all of the aids that new users will find essential. MIRIAD has basically 3 interfaces: a command line interface, a menu interface similar to that of AIPS, and a GUI interface which (especially for MXV) is very intuitive. In the area of user friendliness, we should incorporate expert system/artificial intelligence into aips++, including but going well beyond having full, context-sensitive help available on-line.

3 Target Machines

One defect of AIPS which should not be repeated is that its design was not compatible with *efficient* porting to the highest performance computers, such as Cray supercomputers. Today, the ability to run on large memory, vector machines is not enough. The high performance computers of the future will be massively parallel processor machines; aips++ must be designed to run efficiently on these architectures. One such machine with which BIMA and NCSA programmers have some familiarity is the Connection Machine. NCSA now operates a CM2 and expects to be operating a CM5 within a year. We should target such a machine for the high performance needs of aips++ users.

Unfortunately, C++ is presently not portable to the CM, and there is little reason for optimism that it ever will be portable. In addition, the CM has its own flavor of C, and "ANSI C" is not portable if the parallel processing capability of the CM is to be used. Doing things in powers-of-2 is critical to the CM's ability to parallelize a problem. The CM has its own flavor of C to take advantage of its parallel processor architecture. Our experience with C* ("C-star") - the CM version of C - is that anyone with experience on vector machines, as well as experience using C, will have a very short learning curve ... the language looks like standard ANSI C with features that aim to take advantage of the CM's parallel processing. If aips++

were written in C*, it would be a simple matter to convert to ANSI C (one simply disregards the attention paid to structuring the code towards parallel processing) ... to go from ANSI C to C* is an oppressive task in that the code must be redesigned to take advantage of the parallel processing. However, it should be a non-compromisable objective of the aips++ design that aips++ (at least the computationally intensive parts) run on a massively parallel machine like the CM5.

Because of the above coding complications, it may well not be appropriate to set as our target having all of aips++ run on a massively parallel processor machine. Such machines, like the CM, are ideal for such tasks as FFTs, and other array manipulation operations. Having hooks so that the most computationally intensive tasks can be transparently off loaded to a massively parallel machine which is closely coupled via a high speed data transfer link to a more convention computer, such as a high-end Sun or a Convex, is probably the best plan.

4 System Friendliness

Although all software systems probably make some attempt to be user friendly, users generally give software systems low scores in this area. Aips++ should be designed with user friendliness in mind, in spite of the fact that many of the functions described here may not be implemented initially.

4.1 User Interface

The appearance of aips++ should change as little as possible across machines (eg, a user familiar with aips++ on a Sun system can go to a Convex and see the same software). On the other hand, we don't want to go only through a meta-language which essentially attempts to hide the operating system from the user in order to operate aips++. We need to be able to have system level access to files and tasks.

Both an interactive and command-line interface should be available, the latter usable when batching tasks. Building up a command-line input and executing that is probably the easiest way to allow for different user-interface programs, like menu-systems, direct inputs or MIRIAD-like interfaces. On the other hand, the keyword prompting feature of GIPSY makes it very easy to use and may be one of the main factors contributing to its popularity. Keyword prompting and constructing a command line are not directly compatible, unless special attention is paid to the keyword-input routines from the beginning. The much improved user friendliness that keyword prompting provides makes this effort worth considering.

4.2 Expert Systems

One area which should be emphasized in order to achieve user friendliness is expert systems/artificial intelligence. Some examples of areas where expert systems code would be useful include the following:

- Input checking: Inputs should be checked by the expert system before an operation begins. The user could be warned if the inputs do not make sense to the expert system. If the task the user wants to start will take an extraordinary amout of CPU and/or disk space, the user could be warned. User frustrations, such as having an output file created when the task will bomb because the input file does not exist, could be avoided.

- Keyword defaults: Keyword defaults can be context sensitive, with expert systems code to examine the actual data stream being processed and to attempt to set the best keywords and parameters for each data stream. If the user could call up completely meaningful definitions, the values of the defaults, plus an indication of why the defaults were chosen, that degree of expert systems in aips++ would be extremely user friendly.

- Error Messages: 1) Error messages should be accompanied by useful hints. (When someone tells you what message they got from what program, you should have a good guess at what they did wrong. Those types of guesses should be displayed along with the error messages.) 2) When the program determines that you shouldn't be allowed to do what you are doing (e.g., adding two maps with different center coordinates), it should give a You Shouldn't Do That message, with perhaps an override option.

- Expert-defined tasks: Although all tasks and operations should be modular and broken down into their basic functions, there should be available expert-defined tasks (made up of the primitive tools) which may be used for the major paths through the data processing system without users having to assemble them for themselves. Moreover, these expert-defined tasks should incorporate as large a degree as possible of expert systems knowledge. At the extreme, it should be possible to have a "SHOW MAP" button which for a particular instrument takes raw telescope data through to maps that the astronomer can look at without (or with minimal) astronomer intervention. Such a system will be beneficial for at least 3 reasons. For real-time or near real-time observing (of time variable phenomena or mosaicing or mapping of unknown objects), immediate feedback can save valuable telescope time and maximize scientific productivity. For beginners, an expert system may produce better and more reliable results than the beginner choosing among options (s)he really does not initially understand. For experts, the quick and easy results can provide valuable information to guide fine-tuned data processing.

4.3 Simplicity

Simplicity for both users, programmers, and system managers is essential if aips++ is to be a friendly system. For users, much of this involves hiding complexity. Included among simplicity issues are the following:

- Aips++ should have the ability for users to use aips++ on their own local computer accounts, rather than forcing usage to be through a single, "special" logon for everyone to use. A minimum of effort should be needed to plug a user into aips++.

- It is important to be able to easily get data into and out of aips++ format, so aips++ can be used even when other packages are still needed to complete data reduction, analysis, and publication.

- The idea of 'pipes' could be employed to make a large selection of basic tasks. A 'task' could be something like 'fit a single profile'. Or 'return a particular pointer (to a z-profile etc) to selected data from a particular dataset'. These 'tasks' could then be strung together to treat a full dataset. A standardized 'macro' to loop over pixels or profiles or planes could be provided. And 'macros' for standard processes would be there too. However, users could very easily do something new with their data by just writing a small 'task' to treat e.g. a profile or series of profiles, without worrying about the whole infrastructure of header-updating and reading/writing. Seeing 'tasks' as part of a 'pipe' may provide a comparatively easy way to implement tiling of contour plots, or gray scale and contour overlays etc. It certainly would promote re-usability.

- The internal catalog system of AIPS should be abandoned. We would like to have the ability to control where the data structures exist, not to have them in a general data location or with opaque names and extensions. A hierarchical data set (similar to the MIRIAD data set) which is directory based would provide users with a neatly organized structure than can be identified in commands, saved with any backup technique, compressed to save disk space, and provide users with the freedom to decide where their directory structure should reside. The AIPS internal catalog system introduces needless complexity. For example, the current AIPS has drawbacks in deleting files which use of the basic operating system would avoid: the term ZAP (instead of for example DELETE introduces confusion and ambiguity. Typing ZAP 20 does not delete file 20, but deletes the file of the last "getn", a surprise to new users. A file in the "catalog" cannot be deleted if it has been deleted outside of AIPS (the entry remains in the catalog although there is no file).

- Users should have no need of knowing how their data is stored, i.e. as ra-dec-vel or vel-dec-ra cubes or any other way. The input-specification should suffice to tell the code how to read and write the data. So, making a ra-vel contour plot should be possible from a ra-dec-vel cube just as easily as making a ra-dec contour plot.

- Useful Headers: The header information should be detailed and self-explanatory. Use of cryptic words in headers should be avoided.

- Source code should be readily available to users, not only for users intending to program but also for users who want to ascertain exactly what a given task is doing.

- It should be possible to stop tasks without quitting, in order to change parameters or keywords in mid-stream. An example might be changing the number of CLEAN iterations.

- There should be simple system-wide subroutines to read/write/select image data. These routines should return a profile/plane/cube to the calling task (or pointers to them). They should be part of the definition of the 'image' class, so that the same code is used everywhere. Possibly, one could have a pointer to "profile number n in z of the dataset", so that the calling task does not have to take care of the read/write. By doing this, each task will have easy, optimized access to the data it needs. And loops over profiles (for continuum subtraction or profile fitting) or planes would become trivial. All possible orientations of profiles or planes should be accessible (i.e. x-profiles, z-y planes etc). Memory management should be delegated to the subroutines, not to the application programmer.

- The standard interface (unrelated to the software per se) should be *standard*; the X11 standard should be supported.

- The ability to debug programs via a symbolic debugger is essential, so that for example a new support person without experience in aips++ can track down problems and can follow program flow. A good prototyping environment is necessary, not just desirable.

- A single file format should be used by all machines. Data can be ftp'd from a Sun to a Convex (and among any other machines on which aips++ is supported) and used directly.

- Version upgrades should not be a major undertaking, but should be simple, direct, and straightforward. "Root" privileges should not be required to install aips++. There should be a minimum of structure. We don't want a rats-nest where the search for anything becomes a quest in its own right. Small changes in structure should not force re-loading the entire system (ie, the effect of local structures should be local). Stability in the package is very important.

- It should be possible to load a customized system. In MIRIAD, for example, you don't need to load the whole system; if all you want to do is make models and plot them, then only two tasks (imgen and implot) are all you need. In aips++ single dish processing does not need all the interferometer and vlbi software, and single- dish users should not have to fill up their disks with unwanted code.

5 User Programmability

Both experienced programmers and astronomers with limited programming background must be able to program in the system. At the expert level this means those with C++ and aips++ knowledge; obviously, aips++ must be designed for experts to add, modify, and maintain code without having to have intimate knowledge of all of aips++. Thus, modularity is very important. At the amateur level, an interface that is reasonably straightforward so that astronomers can write their own tasks for their own purposes without gaining an in-depth knowledge of aips++ or C++ is essential. This might take the form of a statement interpreter for the simplest things. However, as FORTRAN is still a "standard" for technical computer users, it should not be discarded in favor of C for this purpose. Astronomers will wish to add modules in FORTRAN. (Moreover, FORTRAN may be necessary for implementation on high performance computers.) Aips++ must interface with FORTRAN programs at some level.

How to support external programming capabilities (i.e., the possibility to string together tasks and use while, repeat or for loops and ifs) is a major problem. To use the C-shell has the disadvantage that it only works under unix systems, and may not even work the same under different flavors. One solution is to make very easily available a function that executes a task, so that one can make a C-PROGRAM to string together tasks. That way ALL capabilities of C are available. Inclusion of a visual programming language, such as the cantata program in the Khoros system, would be extremely useful. (Cantata is a graphically expressed, data flow-oriented language. The user builds a cantata application program by connecting processing nodes to form a data flow graph; nodes represent modules of several hundred lines of code. A dynamic execution scheduler allows the user to interactively execute the entire flow graph across a heterogeneous computer network. There are other, similar systems; Khoros and cantata are are mentioned here because they available free via anonymous FTP from the University of New Mexico.)

6 Documentation and Processing History

Aips++ should have strong astronomer assisted user documentation from the very early stages. Astronomer assisted means that an astronomer not associated with writing the code should have the ultimate responsibility for the documentation. There should also be a documentation czar to insure that the documentation be uniform throughout the project. A beginners "cookbook" should be available that starts out much more simply than the current AIPS cookbook. The endless possibilities should be discussed in a different manual. A possible solution is 3 levels for the users, each self-contained: beginner, user, and advanced user/programmer. Documentation should include a listing of all parameters or keywords with complete, meaningful definitions that are neither so short that the user must guess the meaning or so long as to discourage reading them.

It is essential to have both a good on-line help facility and manual documentation, sufficient both for experts and for new users to effectively use aips++ without hand-holding.

In addition to documentation of the software, there must be documentation of what the user has done to the data during processing. There should be a history/log file for each project, with user flexibility concerning how it is to be used and configured. Every non-trivial action taken by the user or by the software should be listed in this history/log. Further, the user should be able to write comments into the log at any time. The history/log should be available as an ascii file for printing or use in the text processor of user choice. There should be a filter of commands so that the processing may be re-run automatically and completely from the history/log file, so that the user may essentially un-do processing and start over.

7 Tasks

The heart of aips++ will be the tasks that actually do the work. We attempt to list here an incomplete list of tasks which are needed.

1. Data access. Simple applications to get at ANY part of the data structure. For example, it should be possible to grab any x vs. y in the data stream easily and transparently, and to write intensities as functions of x or y in an ascii file. We need easy and transparent high level programs to have low level access.

2. Single-antenna tasks. We do not attempt a complete list here, since this is largely outside the BIMA arena. However, we will use single-antennas for obtaining zero and short spacing data to include with the array data. A task specifically to add arbitrary single dish data to data from an arbitrary array is needed. Given a set of single dish observations (a single-dish mapping), it should be possible to construct a map and then a corresponding visibility dataset. The process of adding the single dish visibility data to the array visibility data should allow checking to see if the calibration is compatible (i.e., compare amplitudes). Options on how to combine the data sets should include using single-dish visibilities inside a given uv-distance and array visibilities beyond it, as well as allowing a "transition region" where visibilities are averaged. Primary beam effects must be appropriately treated. A task for single-dish beam-switching deconvolution is needed. Single antenna and interferometer raw data should have compatible formats so that the two can easily be combined. However, although the data files should have a general format for both kinds of data, it should not be *necessary* to carry around a lot of unnecessary headers and blank files for one sort of data or the other.

3. Calibration. Although calibration of telescope data is peculiar to each telescope, a majority of the functionality will be common. Attention should be given to making as much of the calibration code as possible common, so that there is as little as possible site-specific code. A useful task would cross-check calibration of different data sets of the same object (different in time of observation, but with overlap of uv-coverage, otherwise the same observing parameters). This could also check for variability. The task would look for discrepancies in amplitude (and possibly phase) at the same uv-distance.

4. Imaging. Continuum data are to be seen as 2-dimensional data sets otherwise structured the same as line data sets (although continuum data often needs an additional dimension too, e.g. when needing frequency for Faraday rotation data). Possibly everything should be viewed as mosaiced data. For single-fields some values would then be either missing or set to 1. This allows the exact same task to be used in all cases and thus decreases duplication and increases re-usability. In a sense it means adding an extra dimension to datacubes. But programs like 'primary beam correction' may be the same for the mosaiced and non-mosaiced case.

    - direct FFT - weighted, gridded, tapered imaging and AP computation
    - subtracting continuum in the uv plane (includes subtracting point-sources in the uv plane)
    - deconvolution: 3-D deconvolutions. CLEAN with options selecting variants. Extended source CLEANS (such as SDI clean, multi-resolution CLEAN). MEM.
    - mosaicing (3-D). Should be simply and easily accomplished for an arbitrary instrument.
    - primary beam correction (removal) / generation / fitting.

5. Map-manipulation. Rectangular box or curvilinear region selection should be built in automatically. One can e.g. carry a blanking array along in the C-class, which can be set by the selection mechanism.

    - interactive image data editing
    - image calculus (includes determination of continuum image/moments etc)
    - continuum subtraction (in several, flexible ways)
    - statistics (between data sets, in a subcube, as function of a coordinate, histogramming)
    - profile fitting (and everything that that includes, like gaussians, rotation measure, electron temperature, ...)
    - source fitting (e.g. 2-D gaussians) and source-list production
    - smoothing (1-D, 2-D, 3-D, gaussian, boxcar, hanning)
    - reprojecting (from equatorial to galactic etc, from gnomonic to orthonormal etc)

6. Plotting. Large amount of control on details is essential.

    - simple direct printing of data
    - scatter plots
    - x-y plots (e.g. correlate to datacubes) x/y can be pixels from 2 cubes, profiles from 1 or 2 cubes etc.
    - profiles and tiling of profiles (includes single-profile plotting)
    - ruled-surface plots
    - wedge plots (intensity=area of wedge, velocity=angle, dispersion=width)
    - image slicing perpendicular to curved (or straight) tracks, with scaling, stretching, and/or squeezing non-linearly in order to combine and compare various ones
    - images

- contour plots (include tiled plots)
- gray scale plots
- overlays of contours on gray scales (colored contours), where the various plots can be from different files (e.g. MPLOT at DRAO).
- superposition of coordinate grids
- profiles superimposed on with lines pointing to position in images

7. Publication plots. It is important to have high quality hardcopy. The parameters should be flexible, but have straightforward defaults for the usual plots.

   - A MONGO or WIP type facility would be useful, to flexibly plot ascii data.
   - Useful header information should appear by default on plots, including the data printed.
   - The plots should be available as .dvi files that can be incorporated into, for example, TeX files of articles submitted electronically to a journal.

8. Image display.

   - Simple display of an image, with interactive palette control (annotated palette), many different standard palettes should be available
   - cursor measurement of position and intensity
   - Tiling
   - Movie
   - Intensity-hue display of multiple images
   - Rendered surface display
   - Interactive rotation of either real datacube (slow) or parametrized datacube (like from a cube with results of gaussian fitting, using one plane to define intensities, second (velocities) to define z-position (very fast).

9. Comparison with artificial data. The ability to generate test data with known "results" when that data is processed through aips++ is essential (this is also a strong support-oriented criterion). Also, comparison of observed molecular intensities (in data cubes) with theoretical ones determined from detailed simulations or simple modeling with radiative transfer calculations is important for interpretation of observations and should be possible in aips++. This should include the possibility to create a model from some data set(s) as well as option to use an existing model as input. The model should be re-gridded or whatever is needed if it is not compatible with the array size or orientation of the dataset. The ability to "observe" the model with the instrumental signature imposed is important for the comparison. The full range of aips++ tasks would then be available for comparison of theory and observations. Use of the 4th dimension of the image data sets for model and data might be useful. The ability of users to add rather complex programs themselves to aips++, such as radiative transfer programs, would be very useful. Consideration should be given to system support of standard comparison tasks, such as LVG radiative transfer programs.

10. Archiving. An archiving subsystem, suitable for the storage and retrieval of data over the Internet, would be an important addition to aips++. Whether this should directly be a part of aips++, with code to access archives, or a completely separate system with the ability to input easily to aips++, is unclear. However, the advantage of the archive browse and retrieval software being a part of aips++ is that all the visualization and other tools necessary for an archive tool would be available without duplication and with the standard, familiar interface.

8 Maintenance

Once aips++ replaces AIPS, MIRIAD, and other systems, there remains the problem of maintenaning and further enhancing the system. The aips++ consortium is only committed to the point of developing aips++, not maintaining it. Long-term maintenance of aips++ appears to be an issue which has not yet been faced; this issue must be addressed now and a satisfactory agreement for the long-term support of aips++ put in place before organizations like BIMA can fully commit to having aips++ replace their own software systems. The aips++ consortium must commit itself for the lifetime of the code to upgrade and maintain the *full* system as released to the general user community (including all of the inclusions for local instruments of the members of the aips++ consortium), with appropriate safe guards against critical organizations dropping out of the consortium. As the largest and lead partner in the consortium, NRAO should recognize and accept a special responsibility in the area of maintaining aips++.

NRAO should be site of updates and should keep a current version of the entire system available for anonymous ftp. In addition, up to date patch files that can be retrieved as needed should be maintained in a similar location. A standard mechanism for reporting bugs (and receiving responses) should be established, probably via email.