AIPS++: A New Astronomical Imaging Package

Ray P. Norris

Australia Telescope National Facility, CSIRO Radiophysics Laboratory, PO Box 76, Epping, NSW 2121, Australia.

ABSTRACT. In this paper I describe a new software package ("AIPS++") being written by a consortium of seven astronomical institutions spread over four continents. I start by describing the background to the project, followed by a summary detailing what AIPS++ is and why it is being written in this way. Section 3 describes the challenge of running a globally distributed project spread over four continents. Finally I describe the current status and an estimated completion date.

1. Introduction

Some ten years ago, NRAO developed the AIPS software for use with their newly completed VLA. Since that time, AIPS, which consists of some 600 000 lines of code, has been installed at over 300 sites worldwide. It has been developed and modified by various astronomical institutions to cope with a range of instruments and techniques which were unthought of at the time of its writing. However, AIPS is now showing its age, and is increasingly unable to cope with new computer technologies, new imaging techniques, and new instruments. The continuous additions and modification made throughout its lifetime are now making the code difficult to maintain. It has become clear that AIPS can no longer be modified piecemeal, but that instead we need to replace it by a new generation of software, embracing modern software techniques while building on the experience provided by AIPS.

Australia Telescope National Facility, CSIRO, Australia
Berkeley-Illinois-Maryland Array, USA
Herzberg Institute of Astrophysics, Canada
National Radio Astronomy Observatory, USA
Netherlands Foundation for Research in Astronomy, Netherlands
Nuffield Radio Astronomy Labs., Jodrell Bank, UK
Tata Institute for Fundamental Research, India

Table 1: The members of the AIPS++ Consortium

This new system is to be called AIPS++ (Astronomical Information Processing System). Because of the sophistication demanded by users and techniques, it was felt that no one institution had the resources to build this system. A consortium was therefore formed of seven institutions, listed in Table 1, each of which runs major radio-astronomical facilities.

Each institution provides between one and seven programmers, making a total team of about 18 programmers at present, although this number will increase as we start writing higher-level applications. An essential part of this simultaneous development over geographically dispersed sites is that code needs to be kept in step at all sites. This is achieved by running an automatic job overnight which distributes code and documentation via Internet, and ensures that no site is more than a few hours out of step with the others.

2. What will AIPS++ be?

2.1. SCOPE

The goal of the AIPS++ project is to provide an image processing and data reduction system for all mainstream radio-astronomy applications throughout the world. This will include both aperture-synthesis and single-dish applications, but will not initially target highly specialised applications such as pulsar search algorithms. Single-dish and aperture-synthesis software have traditionally been viewed as two separate packages. However, as newer aperture-synthesis instruments routinely provide auto-correlation data as well as cross-correlation data, and as single dishes acquire multi-beam feeds, the traditional distinctions between single-dish and multi-element radio-astronomical techniques are diminishing. It is therefore now appropriate to combine these into one software package.

An additional goal of the project is to provide a platform for optical interferometry applications, although the actual writing of those applications will be left to practising optical interferometrists. The reasoning behind this decision is that optical interferometry uses many techniques common to those of radioastronomy, and so it is natural to include optical interferometry applications within AIPS++. A further reason is to encourage cross-fertilisation between the two disciplines. In the longer term, software may be written for other disciplines or wavelengths within AIPS++, but we do not intend to target them at this stage.

A potential problem that could be caused by aiming at such a wide range of targets is that the resulting code may be very bulky. For example, a user who is interested only in single-dish applications may not want to clutter up the disk with VLBI applications. The code will therefore be divided into sections (core, single-dish, aperture-synthesis, VLBI, and so on) so that a user or institution may load only a subset of the AIPS++ package.

2.2. SPECIFICATIONS

AIPS++ is designed around a set of user specifications which have been assembled from specifications prepared by the astronomers at the consortium member institutions. These specifications are freely available by anonymous ftp (see Section 5), and we will always welcome comments on them. Broadly, AIPS++ will have the following features.

What the user sees

- It will contain (as a minimum) most of the present functionality of AIPS
- It will take advantage of new computing technologies such as graphical user interfaces, distributed computing, and massively parallel machines
- It will have a variety of user-friendly interfaces, including text input, a graphical user interface, and a dataflow interface (such as that found in Khoros and AVS).
- It will have on-line documentation as well as hard-copy documentation
- It will be easy for users to write new tasks
- It will be possible for users to enter algorithms (as in IDL and PV-Wave) at the user interface. For example, a user might write ImageA=ImageB/5 + SQRT(ImageC).
- It will be distributed freely (using a gnu-style license) via anonymous ftp
- Regional and international user groups will be set up to encourage dissemination of information and feedback to consortium members

What the user doesn't see

- It will use object oriented techniques to maximise maintainability and adaptability, and be written in C++
- It will use the X window system for display purposes
- It will be portable to most Unix (strictly, Posix-compliant) systems
- It will adhere to industry standards (Posix, X, C++, etc)
- Code will be kept simple by taking advantage of operating system features
- Wheels will be borrowed rather than re-invented where possible (e.g. Khoros, InterViews)

2.3. WHO WILL WRITE THE APPLICATIONS?

Fig 1. shows the three levels of the AIPS++ software. At the centre is the core of AIPS++, containing efficient low-level software, written by computer science specialists. This core includes libraries, data structures, and the interface to the operating system. Surrounding the core are the "official applications", which constitute the bulk of the code seen by most users, and which are distributed by the AIPS++ regional centres. These applications include all the essential astronomical and imaging tasks, such as tasks to edit, image, and display data, and conform strictly to standards of coding, documentation, and reliability.

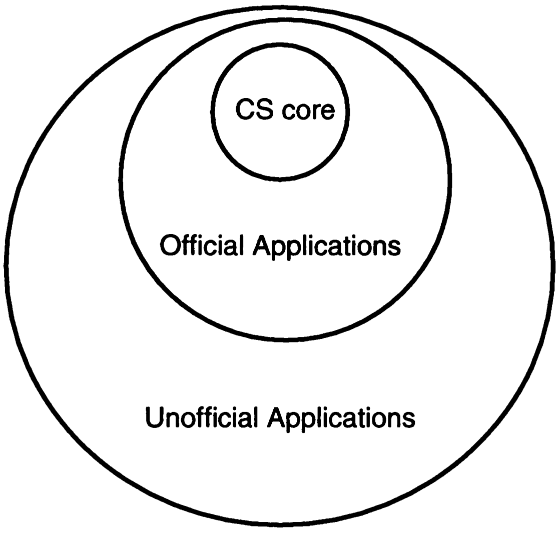

Fig 1: The three layers of AIPS++ software

Finally, there exists a layer of "unofficial applications". These are applications freely written by users without having to conform to the standards of coding which are enforced for the officially distributed code. Use of these unofficial applications will be strictly at the user's own risk. However, such tasks which prove to be particularly useful or of wide appeal may be adopted by the AIPS++ centre, recoded to the AIPS++ standards, and incorporated into the official layer.

2.4. OBJECT-ORIENTED PROGRAMMING AND C++

AIPS++ will be written in the object-oriented language C++. It is not my intention here to give a tutorial on object-oriented programming or C++, but rather to give the casual reader some flavour of what is involved in object-oriented techniques.

Traditional programming (function-oriented programming, or FOP) has relied on a functional approach, where emphasis is given to the functions (corresponding to verbs or subroutines) rather than the objects (nouns, common blocks, and data structures), which tend to be added as an afterthought. In object-oriented programming (or OOP), on the other hand, the design is built around the nouns rather than the verbs. FOP will generally produce a number of subroutines within which are embedded the objects (data structures). In OOP, a number of objects (or classes) are produced, within which are embedded instructions (the methods) which tell how to perform operations on them.

In principle there is no particular reason why this should offer any benefit, but in practice it is found that object-oriented code is less complex, and carries fewer connections between modules. In FOP the number of connections between modules tends to increase as the factorial of the number of modules, whereas in OOP there are far fewer interconnections. While FOP is appropriate for small or medium software projects, in large software projects it tends to lead to complex code which is difficult to maintain and adapt.

As well as this primary difference, there are a number of other elements of OOP as follows.

Encapsulation. Each module is self-contained, so that it can be modified without affecting the rest of the package. This also enhances the reliability because each module can be thoroughly tested on its own. Maintenance is also improved by the consequently limited scope of variables.

Inheritance. Objects can inherit properties of a more general class. For example, for the class of animals we can define a number of functions (eating, breathing, etc.) which can be inherited by another class (mammals) without having to rewrite the code. A further class (cows) could then be defined which would inherit the properties of animals and mammals, but would also have its own special functions defined (mooing). This approach reduces the volume of code by eliminating redundancy, and consequently reduces the number of bugs.

Polymorphism. Polymorphism is the ability of a function to choose the appropriate method depending on what it's operating upon. For example a "Draw" function might be defined to act on a class of "Picture" objects, and it will then be defined for each type of object ("Circle", "ContourMap", etc) inherited from Picture. When called, Draw will automatically pick the right function depending on the object, and new types of Picture can be added without changing the applications which use Draw. Thus the writer of an application who wants to draw an object on the screen doesn't have to worry about the internal details of the objects being drawn. This results in less complex and more understandable applications.

Design Tools. A feature of OOP is that the balance between design time and coding time is shifted far more towards design than in FOP, so that as much as half the time dedicated to the project may be spent in design rather than construction. This is very frustrating to those of us who expect to sit down and write code on day one of a project, but will hopefully result in a much better design.

Finally, a disadvantage for many astronomers brought up on Fortran will be their unfamiliarity with object-oriented programming. However, undergraduates now in university are being taught C and C++ rather than Fortran, and so to the next generation of graduate students it will be the Fortran rather than the C++ which will be strange and unfamiliar. A Fortran interface will be provided to write tasks in Fortran, but this will clearly be less powerful than the C++ interface.

3. Project structure

3.1. PROJECT MANAGEMENT

Management takes on an even more important role in a project spread over seven institutions than it does in other projects, and we have worked hard to represent interests of all the institutions while trying to avoid "software designed by a committee".

The management structure is shown in Fig. 2. Policy decisions are taken by a steering committee consisting of a representative from each institution, but the day-to-day management is by a team consisting of a project manager, project scientist, and a project computer scientist, resident at the Project Office (currently at NRAO, Charlottesville, Virginia). Programmers at the individual sites report to the project manager, and are allocated "software contracts" which remove any ambiguity over what is expected from them and when it should be delivered.

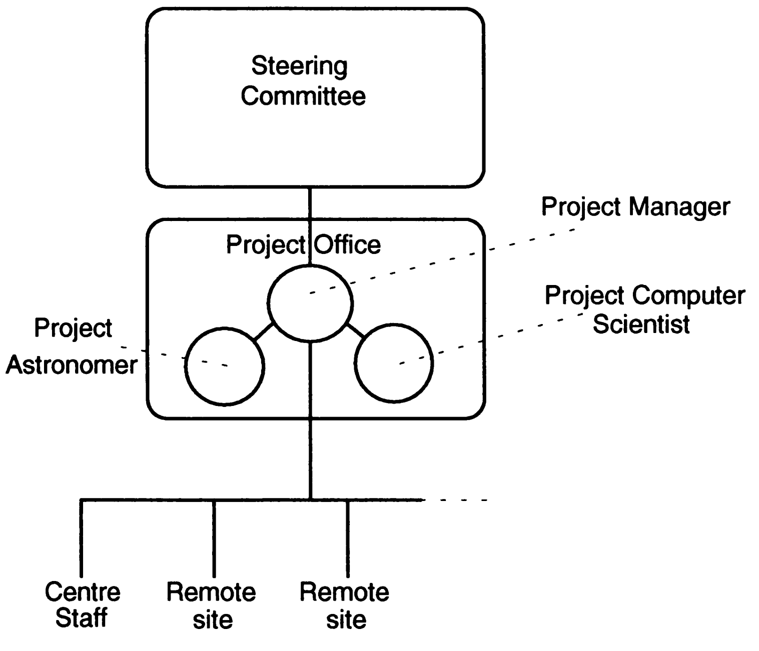

Figure 2. Project Management Structure

3.2. PROJECT COMMUNICATIONS

Good communications are obviously important for this project, and take place at several levels. It should be stressed that we rely heavily on Internet for all our electronic communications, and it is doubtful whether a project like this would have been feasible a few years ago before the near-universal electronic access that we now enjoy.

Much of the actual work is done at the individual institutions, except that several programmers spent a few months at Charlottesville in early 1992, where the initial problem analysis and design phases took place. Throughout the lifetime of the project, occasional exchanges of programmers between institutions will continue. Code and documentation written by programmers at any one institution is automatically transferred overnight to all other institutions, so that the code at each institution is never more than a few hours out of step with that at any other institution.

Most of the business of the steering committee (and of most of the AIPS++ team) is by electronic mail, with only three or four face-to-face meetings each year, which are rotated around the participating institutions. In addition, we have recently started monthly telephone conferences, which are surprisingly successful for a multi-continent project like this, although some members may respond rather sleepily to the early-morning brightness of others.

4. Current status of the Project

The construction of AIPS++ started on 1 January 1992, and a system will be completed by the end of 1994 which will largely replace the main functionality of AIPS.

1 Jan 92	Project started
Q1&2 92	Programmers assemble at Charlottesville for "indoctrination"
Q3&4 92	Design basic math classes, tools, kernel Construct of code/documentation distribution system Select of CASE tools
Q1&2 93	Finalise kernel design Complete database system, prototype user interface, on-line help, simple imaging tools. Construct prototype to take FITS files, FFT and image them
Mid 93	Start construction of major applications Construct VLBA applications
End 93	First major subsystems available for "friendly" users
Mid 94	Operational single-dish system
1994	Design and construct more applications
1 Jan 1995	Release AIPS++ to general users Cease AIPS support

Table 2. The AIPS++ Timetable

Thereafter, additional applications will continue to be added to AIPS++ over its estimated ten to fifteen-year lifetime. Table 2 gives a brief summary of the milestones in the development cycle of AIPS++.

5. Conclusion

AIPS++ is an ambitious project, tackling not only an unfamiliar software methodology, but also the challenge of managing a large project distributed over seven astronomical institutions spread over four continents. We do not expect the path to completion to be smooth, but signs so far give us cause for optimism that we will achieve our goal. If AIPS++ is the success that we hope, then it will be interesting to see the extent to which this type of collaborative project becomes more common in the astronomical community. Such collaborative projects not only have the potential to reach goals that are beyond the scope of any one institution, but also have the potential to achieve cross-fertilisation between both disciplines and institutions.

For further information on the AIPS++ project, all code and documentation is available by anonymous ftp from aips2.cv.nrao.edu. Of particular interest is the project book, stored as /pub/aips++/docs/Pbook.ps.

Discussion:

Armstrong:
What hooks for optical aperture seynthesis will be part of AIPS++? Will it be able to deal with data consisting of closure phases and squared visibility amplitudes?
Norris:
That is certainly our intention, although right now the emphasis is on radio-astronomical applications. We would welcome input on what needs to be done in such areas as the imaging model.

Bedding:
How confident are you that AIPS++ will be able to adapt to and take advantage of future hardware advances?
Norris:
My (personal) estimate of a lifetime of 10-15 years for AIPS++ is based on past experience which says that 10-15 years is the time it takes for hardware to evolve sufficiently to make existing software obsolete. So, we are building AIPS++ to cope with the hardware which either exists now or we can guess will be available soon, but I don't have enough confidence in my crystal ball to predict it will be able to cope with the hardware available in 10 to 15 years time.

ten Brummelaar:
Will data abstraction, operator overlaying and other aspects of OOPS be available to the user via the script language?
Norris:
We are currently discussing this. Many users will want to adopt a functional approach in the script language, and we will support this. However, some users will certainly want to access the power of OOP and we will investigate ways to make this possible.

Marson:
Given that C++ programmers spend as much time designing the code as writing it, will it still be possible to whip up a quick and dirty program in AIPS++?
Norris:
Yes – one of the main aims of AIPS++ is to make it easy for users to write their own tasks, and incorporate their own experimental algorithms. Users who wish to write their own tasks will find a suite of tools available, and in those tools resides the investment of time spent in the design, so users will find it fairly easy to 'whip up a quick and dirty program'.