NOTE 220 – AIPS++ SYSTEM PLAN

Ger van Diepen

26 September 1999

Contents

1 Introduction

The purpose of this document is to sketch the developments needed and envisaged in the system area. It gives the dependencies of AIPS++ on its environment (e.g. compilers, standards) and how they may change in the future.

The priorities for the AIPS++ project are:

- Have a scientifically usable system for the common UNIX platforms. The documentation should be such that a user can find its way.
- Stabilize the development system. Configuring, building and releasing the system should be fully and easily supported.

- Make a plan for future releases. It has to be investigated and decided which changes need to be made and which capabilities need to be added. Changes and additions have to be weighed against the scientific priorities in the project.

2 Current system

AIPS++ can be executed on any UNIX system with X-windows. It has proven to run under SUN Solaris, HP-UX, Linux, SGI/Irix, and Digital Unix on a variety of machines. It requires little extra software. The public domain package Tcl/Tk is needed for the graphical interface provided by glish. Netscape is needed for browsing the documentation.

Developing in AIPS++ requires much more software, all of it available in the public domain area. The following software is needed:

- C++ compiler
- C compiler
- Fortran compiler
- RCS
- GNU make 3.74
- latex and latex2html
- ghostview
- LAPACK
- WCSLIB
- PGPLOT
- Perl 5.003
- Tcl/Tk 8.0

The offical project C++ compiler is egcs 1.1.2, which works fine for all supported UNIX platforms. GNU and egcs have merged again, so the newest g++ compiler is 2.95, which will be adopted in the future by the AIPS++ consortium. New compilers are often based on the EDG front-end. Ports to the KAI and SGI compilers, based on EDG, are mostly completed.

The C++ code is quite heavily templated. Templates are explicitly instantiated, because automatic instantiation led to excessive link times.
AIPS++ has some Fortran77 code, which can be compiled with g77 or a commercial Fortran compiler.

In October 1999 the first release of AIPS++ was distributed on CDs, which contain executables, libraries, documentation, and source code. It is possible to get updates via ftp.
Sites actively developing AIPS++ can obtain and update a development version via ftp.

3 Future developments

3.1 C++ standard and Compilers

C++ has now an official ANSI/ISO standard. Several new possibilities in the C++ standard are already used in AIPS++.

- Keyword explicit for one-argument constructors.
- True bool data type.
- The local scope of a variable created in a for-statement.
- True exceptions. Note that the AIPS++ code hardly uses AutoPtr to delete objects on the heap. Code should be changed to avoid memory leaks.
- Namespaces.
- Enums and statics in templated classes.

Some other options are of interest and might be used in the future.

- Better templates. They are needed when we want to implement something like Blitz++ to speed up array calculations.
- Standard Library of which STL is an important part. Since all basic classes have already been implemented in AIPS++, the Standard Library is not really needed. However, it may be better to switch to it to make it easy to use future other C++ software. It may also be expected that the classes in the Standard Library are well optimized.
- Casts are done in a better way. They should be used to get safer code.

- Covariant return types. An overloaded function in a derived class can return a derived object.

The AIPS++ consortium sites use the products purify, purecoverage, and quantify to check the code thoroughly. Unfortunately Rational only supports egcs 1.0.3 for these products. Therefore only those C++ constructs can be used that are supported by egcs 1.0.3 or can easily be emulated.

3.2 Porting to Windows NT

A port to Windows NT is underway as part of the parallellization efforts at NCSA to use AIPS++ on an NT cluster. Apart from that it is foreseen that NT will be an important platform. It raises the problems how to map the UNIX and X11 functionality to Windows NT.
Windows NT has the POSIX standard implemented, but when using it the Win32 API cannot be used anymore. This limitation is not acceptable.

AIPS++ uses several UNIX system and library calls. The most important are file handling, pipes, sockets, time, environment variables. X11 is used heavily in aipsview. It is also used in the Display Library, but the class structure is such that it should be easy to use classes dedicated to NT.

There are a few public domain and commercially available packages being able to aid in a port. Amongst them are:

- NuTCracker (commercial) supports Unix and X11R5. It can also be used for Windows95/98.
- Exceed (commercial, Hummingbird) supports X11R6 and NFS. It can also be used for Windows95/98.
- Cygwin32 (public domain, Cygnus) provides a UNIX-like API and tools (like gcc, bash). It can also be used for Windows95/98.
- UWIN (public domain, Korn AT&T) is similar to Cygwin32. Its source code is not available.
- Interix (commercial, formerly OpenNT) supports Unix, X, Motif, and NFS. It has the drawbacks that is does not support Windows95/98 and that it is not possible to mix UNIX and Win32 calls.

When doing the port it has to be decided what is the best way to go. There are a few scenario's possible.

- Rewrite UNIX/X11 specific code. This may cost quite some time.

- Do the port using tools only. This may turn out to be (too) expensive when commercial software is needed.
- A mix of using tools and porting ourselves.
- Starting with tools, but slowly moving towards rewriting. NT system and library calls.

A possible porting strategy could be:

1. Handle parts of the basic library not dependent on UNIX or X11.
2. Handle UNIX-dependent library code.
3. Handle glish. Tcl/Tk has already been ported to Windows NT, so the glish graphics should not be a problem.
4. The final step is porting X11-dependent code.

It has to be decided if and which porting tools will be used. Using Win32 calls directly results in best performance, but tools make the porting easier.

As soon as code development will be done on Windows NT, a code management system (integrated with the UNIX side) and a build system should also work on it.

3.3 Code Management

The current configuration, code management, and build system has a few problems:

- Support for checking in/out of entire directories (for tables) is lacking.
- Releases are not supported.
- Partial distribution is not supported.

It has to be investigated whether current subsystems have to be replaced by other packages (public domain or commercial).

Another problem is the size of the libaips library. It is already quite large and will get twice as large once the trial code is moved to aips. It requires that the aips package is split into a few subpackages. A requirement is that they are layered and that a lower level layer is not dependent on a higher level layer. This may be hard for DO implementations which tend to use a lot of classes.

Splitting the library can be done in a simple way by having a few packages at the aips level. A more fundamental approach is to have a subpackage layer under aips. That makes it possible that other packages (nrao, dish) are split in subpackages.

An extra directory level could also be useful in the apps tree. One can think of an application group (with maybe some common source files) and in subdirectories the individual applications.

Shared libraries and dynamically loadable code are used to make dynamically configurable code possible. The glishtk part of glish is relying on shared libraries.
Some experiments have been done with using shared libraries to reduce the size of the system. So far they have not been successful, because too many seldomly used code got linked into the shared libraries. Also the use of static variables gave problems. After splitting the libraries another attempt can be done.

3.4 Documentation

The current user documentation system, based on latex and latex2html, is fragile and hard to configure. It needs to be stabilized. At the moment Kate Weatherall and Wes Young are investigating other methods like SGML-based systems..

Care should be taken that no latex constructs are used which are not (yet) supported by latex2html. Having to do very frequent latex2html updates can be a pain for some users.

It should regularly be checked if the search facility for the AIPS++ html pages is selective enough.

The code documentation system is based on the Perl script cxx2html which extracts declarations and comments (with special tags) from the header files. It is doing a very good job, but needs some changes to make nicer html output.

3.5 Release support

On a regular base AIPS++ releases are done on CD. In October 1999 the first release was done. Users can get updates via ftp. In order the reduce ftp traffic a mechanism is needed to make the updates selective, i.e. that only the parts needed are ftp-ed. It would also be nice if the dependencies are such that linking a program is based on changes in the source code and not on changes in a library. In that way a program is only linked when really needed, thus also ftp-ed when really needed.

3.6 Support of industry standards

There are several industry standards which are of use to AIPS++. It has to be decided if and how they are supported in AIPS++.

- Multi-threading gets more and more common. In AIPS++ it could be used to have non-blocking DO's and to have read-ahead and write-behind in the table system. Later it could also be used for finer parallellization, but it will require some changes to make the AIPS++ library more thread-safe.

- Standards like CORBA,DCOM,OLE,ODMG and Java are getting more important and make it easier to communicate with other software. They are not vital for AIPS++, but when possible it should be tried to support them.

4 Priorities for system

After the release in October 1999 two formal releases are foreseen until the end of 2000. The last release is meant to be used by any programmer who likes to develop code using AIPS++ classes.

Before the next release the following needs to be done:

1. Implement data proposal of Wim Brouw.

2. Improve using native exceptions. To prevent memory leaks, all variables on the heap should be put into an AutoPtr object until their pointer is taken over.

3. Split the library into some layered subpackages.

4. Build shared libraries from the lowest layers. Static variables sometimes cause problems, so we should try to get rid of them.

5. Improve the stability of the documentation system.

6. Improve cxx2html.

7. Have a better dependency analysis to link based on source code changes instead of library changes.

Before the following release:

1. Use threads in DO's to make them non-blocking. It requires that static data is thread-safe.

2. Port to Windows NT.

3. See if automatic template instantiation can be used. This is needed when heavily templated classes in Blitz++ or STL are going to be used.

4. Replace some of our classes by classes from the C++ Standard Library. Candidates are String, Complex, Map, Block. Other classes like valarray should also be looked at.

5. Use Blitz++ or techniques used in it to speed up the Array arithmetic.

6. Investigate how CORBA, DCOM, OLE, XML, ODMG, and Java can be used.