1 Introduction

Although I have used casa (and previously, AIPS++) on occasions for a number of years, in the last couple of months I begun spending much more time working with it and on it. One of the reasons for this is that we have begun to receive interferometric testing data from OSF, and use of casa (or at least, casa-core) is necessary to read the resulting Measurement Set (MS) data format.

The second reason for this pick up in activity is that one of the primary objectives of our EU-FP6 funded work in Cambridge is to provide a WVR phase correction algorithms and tools which are integrated with the ALMA software, and in particular with CASA. We are now at a stage with this work where we need to start detailed planning of this integration which requires a very good understanding of CASA.

Inevitably during this initial stages of work I came across some areas which I think could be done better and/or suggestions for future improvements. Obviously this is by no means a detailed study, just items noticed during the normal course of work, so no doubt some of the points below are misplaced and perhaps some have been resolved long ago but I’m not aware of these resolutions. Similarly, this is by no means an exhaustive review, just an ad-hoc list. Anyway, below are the opinions & suggestions, I hope they will be useful.

2 Short term/relatively easy

2.1 An open mailing list for casa

There should be an open mailing list to which all users and developers of CASA can subscribe if they wish, and which will be publicly archived and accessible through standard gateways such as GMane. There should be a separate list primarily targeted toward development topics, but again fully open. Such a mailing list is the backbone of most successful, reliable, open-source projects (e.g., Linux kernel, Emacs, GSL, etc.).

I am still surprised that there doesn’t already exist such a list for CASA. I know that one concern which has perhaps influenced this decision is that it would divert too much of developers time on continuously supporting CASA and would therefore drastically slow down the development. I think this concern would be quite misplaced in relation to the mailing list, since the primary purpose of such a list would be for the users and
external developers to support each other, rather than expecting the core developers to spend much time on this list.

Further more, any time that core developers of casa did choose to spend replying to messages on such a list would be time much more efficiently spent than privately replying to individual messages. For example, I have exchanged a few emails in the last two months with Darrell Schiebel and Brian Glendenning, and I think it would have been useful if all of those were in the public domain. This would be automatically accomplished by an appropriate mailing list.

I should note that I think (and my impression is I’m not the only one!) that public mailing lists and Wiki’s complement rather than replace each other.

2.2 An open bug-tracking system for casa

All users and external developers of casa should be able to freely access information such as:

- List of known defects affecting each release
- Progress on tackling known defects
- Plans for implementation of new functionality

The (perhaps obvious) reasons for this is that having this information means users/external developers do not have to waste time reporting/investigating already known defects, finding work-arounds for defects just about to be fixed or waiting for a new function that is still many moths away.

All of this information is already collected in the casa JIRA system, it just need to be made publicly available. Again, such an open bug tracking system are standard on open-source projects and work fine on very large projects (e.g., the original ‘bugzilla’ https://bugzilla.mozilla.org/, or the Ubuntu bug-tracker at https://bugs.launchpad.net/ubuntu/). The software required for supporting such a public bug-tracker is freely available, and even free hosting is available (e.g., Google code, or Launchpad https://bugs.launchpad.net).

2.3 Casa-core revision control

Currently, revision control of casa-core is hosted in two separate repositories:

- Google code: http://code.google.com/p/casacore/source/browse/
- NRAO: https://svn.cv.nrao.edu/svn/casa/casacore/alma-evla/

Unfortunately these repositories are independent of each other, they contain different information, and there is significant loss of information when commits applied to one are moved to the other. It would obviously aid long-term maintainability and usability of casa-core if only one repository was used by everybody.

It has been put to me in the past that this arrangement is beneficial because it allows one camp to review the changes by the other camp before putting them into their repository. Such a review process however should be handled through the appropriate use of branches, as is standard practice in the software engineering community.
2.4 A SCons based build system for \texttt{casa}

The casa-core build system is based on the SCons tool. My experiences with this system have been generally very positive and I think it has a number of advantages compared to older systems. I have been particularly impressed by the ability to accurately and efficiently do parallel builds, distributing the compilation on all of the cores present in the computer.

The build system of the remainder of \texttt{casa} is however based on a myriad of Makefile’s, but without the use of the standard tools such as autoconf/automake. The results is a non-standard configuration and build system which is difficult for newcomers to understand.

I think transferring the remainder of \texttt{casa} to the SCons build system would improve the accessibility of casa to new developers, make it easier to support a variety of platforms, and increase developer productivity. It would also mean that all of \texttt{casa} is built in a uniform way.

I recognise some that some effort needs to be expanded for this switch to the SCons to be made. I am, however, working on a prototype SCons based scripts for casa and can keep the project informed about this (via the mailing list of course!) if there is interest.

2.5 Distributed version control for \texttt{casa}

Casa and casa-core currently use the Subversion revision control system. This system is far superior to the system used by the remainder of the ALMA project, i.e., the CVS system, which is now obsolete and inappropriate for either ALMA or casa. Nevertheless, Subversion does have some shortcomings:

- A single centralised server which is a single-point of failure, and introduces network latency/speed restriction for remote users
- Users must have write access to make their own changes
- Users must be connected to the network to view the history of the project, record the changes they are making or switch between branches
- Poor intrinsic support for merges (perhaps this has been resolved?)

For these reasons, Subversion has not been adopted by the many of the largest open source projects like the Linux kernel.

All of these issues have been very satisfactorily resolved by the new generation of third-generation “distributed” revision control systems. The three main 3rd generation systems available today are: \texttt{git}, \texttt{hg} and \texttt{bzr}. All of these tools are available freely under open-source license and are used in very large projects.

I’ve been using \texttt{bzr} for the last three years with very satisfactory results. In particular it is very easy to setup and the existing Subversion repositories and tools can be trivially converted to \texttt{bzr}.

Adopting a distributed revision control system would have the following main benefits:

- Much easier use of feature and release branches, which appears to be a somewhat weak point current development cycle
- Easier for external developers to track progress and contribute
3 Longer term/More difficult

3.1 Interfacing between C++ and Python

I recognise that there were many different drivers when deciding on the mechanism for the interfacing between the portions of \texttt{casa} that are implemented in C++ and Python. I think however the current design is spectacularly complex and unusual, and I think it would be useful to review in the medium term if this is still the optimum way to proceed.

In particular some of the flaws of the current system are that:

- It relies on a complex third-party package named CCMtools, which is no longer developed
- The binding between the C++ code and the \texttt{casa} “tools” tries to solve two challenges at once (or, at least this is my understanding):
  1. Interfacing between C++ and Python
  2. Distributed computation by using a network-aware communication mechanism (CORBA)

It is not at all clear to me however that implementing distributed computing at the granularity of the “tools” is useful, so it may be that this second driver is not relevant going forward

- The mechanism of partially generating the Python code from XML description seems rather cumbersome, it involves a number of extra tools and technologies (XSL) which are quite unusual in this area. I think it should have been quite easy to simply stick to Python (it has excellent “higher-order” programming facilities) and make the code much easier to understand and develop further

3.2 HDF5 support

I am intrigued by the initial support for the HDF5 format in \texttt{casa-core}. In my experience, I’ve found HDF5 to be an efficient format, and of course it has the advantage of being a fully open, cross-discipline, standard and is relatively widely used. Using HDF5 has the great advantage that it is possible to instantly and independently “get to the data” without having to build on top of \texttt{casa}. Another very useful aspect of HDF5 is the excellent PyTABLES package (http://www.pytables.org/moin), which makes access to HDF5 very easy.

In fact (I have to admit), in my current prototype environment I write out visibilities collected at the OSF to a HDF5 file using a simple C++ program on top of \texttt{casa-core} and then interactively analyse these data using Python/PyTABLES/numpy.

So for me, a very interesting future possibility would be if it were possible to write the Measurement Set in HDF5 format instead of the current custom-designed format that makes inter-operation quite difficult. Would that be possible?

3.3 Updating C++ frameworks

(I know this is tall order but I think it is essential if \texttt{casa} is really going to hang around for the expected lifetimes of EVLA and ALMA.)

Many of the C++ frameworks within \texttt{casa} have been designed and implemented specifically for \texttt{casa}. This includes, for example:
• Data structures such as sequences/arrays of objects, matrices of numbers
• Random number generation
• So-called “smart” pointers

In the beginning of the AIPS++ project there was little choice but to do so. In the intervening years however, standard designs and implementations of these framework have become available and widely used (most notably, the STL and the BOOST libraries). There would be a number advantages if casa were to use some of these standard solutions:

• New developers would find it much easier to contribute since the syntax and semantics of generic objects such as vectors, matrices, strings, etc, would be the same as in other C++ applications
• It would be easier to interface to other open-source libraries
• Developers who worked on casa would find it easier to subsequently work on other scientific projects or in industry

If this transition to standard libraries is not implemented, it will become progressively harder to recruit people to work on the casa project, since the framework it uses has so significantly diverged from what is commonly used on new projects these days.

This would be quite a big project, but one which I think would be very worthwhile. It should however be done slowly and in carefully selected stages to maximise the benefit and ensure that the standard libraries chosen to build on are indeed going to be maintained and supported in the long term.