Choosing a control system for CCAT

D. L. Terrett,[a] Patrick Wallace,[a] Alan Bridger,[b] Dennis Kelly[b]

[a] Space Science Department, Rutherford Appleton Laboratory,
Harwell Science and Innovation Campus, Didcot, Oxfordshire OX11 0QX, UK;
[b] UK Astronomy Technology Centre, Royal Observatory, Edinburgh, Blackford Hill,
Edinburgh EH9 3HJ, UK3

Choosing a control system for CCAT

D. L. Terrett*[a], Patrick Wallace[a], Alan Bridger[b], Dennis Kelly[b]
[a]Space Science Department, Rutherford Appleton Laboratory, Harwell Science and Innovation Campus, Didcot, Oxfordshire OX11 0QX, UK;
[b]UK Astronomy Technology Centre, Royal Observatory, Edinburgh, Blackford Hill, Edinburgh EH9 3HJ, UK

ABSTRACT

The Cornell Caltech Atacama Telescope[1] is a 25m aperture sub-millimeter wavelength telescope to be built in northern Chile at an altitude of 5600m. Like any modern telescope, CCAT will require a powerful and comprehensive control system; writing one from scratch is not affordable, so the CCAT TCS must be based, at least in part, on existing software. This paper describes how the search for a suitable system (or systems) was carried out, looks at the criteria used to judge the feasibility of various approaches to developing the new system, and suggests the further studies needed to validate the choices. Although the purpose of the study was to find a control system for a specific telescope with its own particular technical requirements, many of the factors considered, such as maintainability, the ability to adapt to new requirements in the future and so on, are of concern to all telescopes. Consequently, the processes used to select the system for CCAT are relevant to other projects faced with the same decision, even if the conclusions turn out to be different.

Keywords: Telescope control, software environments, sub-millimeter, CCAT

1. INTRODUCTION

The amount of re-use of software from another project can lie anywhere between two extremes: taking the entire control system and using it without change, or writing a new control system using components of the infrastructure software from an existing system or systems.

Adopting an entire system without any change can be achieved only by restricting the choice of hardware to that used by the telescope for which the system was originally written, and accepting the inevitable compromises. This is unlikely to be a realistic option for CCAT. However, there are still advantages in taking an existing telescope as the starting point for implementing a new system, adapting and replacing parts as necessary to meet the new requirements. In particular, when you start with a system that already works, even if it doesn't yet meet all the functional and performance requirements, software is less likely to be on the critical path for hardware testing and commissioning. Where new software is needed, it can often be developed on hardware that is already calibrated and known to function correctly.

Even when the control system is implemented essentially from scratch using existing infrastructure software from another observatory, it is natural to use existing control system applications as a starting point for the new system. However, broadly speaking, the closer to the first option the implementation can be made; the lower will be the software cost during the construction phase.

Another form of software reuse is to develop new software in collaboration with another project which is at a similar stage in its life-cycle. The feasibility of this sort of collaboration depends as much on political factors, such as funding sources and schedules, as technical issues. At this time, the only upcoming large telescope projects that have announced details of their implementation plans for control software are ATST, LSST and the Australian SKA Pathfinder.

*david.terrett@stfc.ac.uk; phone +44 1235 446478

Software and Cyberinfrastructure for Astronomy, edited by Nicole M. Radziwill, Alan Bridger,
Proc. of SPIE Vol. 7740, 774028 · © 2010 SPIE · CCC code: 0277-786X/10/$18 · doi: 10.1117/12.856371

2. CONTROL SYSTEM ARCHITECTURES

In the past two decades, some evolutionary convergence seems to have taken place in the overall software systems of major optical/IR observatories: ESO, Gemini, UKIRT/JCMT being examples of observatories that use, more or less, the same top level architecture. Existing plans for extremely large telescopes (the TMT and the European ELT at least) seem to be heading in the same direction.

2.1 Top Level Control

This top level architecture is based around the concept of *principal systems*, each with clearly defined responsibilities. The principal systems originally identified were:

- The *telescope control* system (TCS), which handles everything to do with the control and monitoring of the telescope and its subsystems (for instance field rotators, auto-guiders and AO systems).
- The *instrument control* system (ICS), which handles everything to do with the control and monitoring of the instrument and its subsystems (typically a cryostat and/or mechanism control system, and a detector controller system).
- The *data handling* system (DHS), sometimes known as the Data Management System, which handles everything to do with collecting and storing data, both data from the detectors and the associated metadata.
- The *observatory control* system (OCS), which coordinates the activities of the other principal systems. Typically this will have a "sequencer" subsystem for sequencing the operation of the other principal systems, and some sort of interface to the operator. It may also contain software for scheduling observations.

Gillies et al.[2] contains an alternative realization of this concept in the context of defining the software systems for the TMT. It adds two additional principal systems; a *facility control* system (for the enclosure) and a *wavefront control* system . However, the basic paradigm remains the same, and demonstrates the advantages of this design: there is a clear separation of responsibilities, which promotes the definition of clear interfaces between the principal systems, and the latter may be built and tested essentially independently of each other, especially with the provision of simulators for the other principal systems.

The implementation of the DHS varies: it usually acts as a server, responding to requests to store pixel data and/or metadata, but in some designs will also take responsibility by asking for metadata from other systems at appropriate times.

The main problem with this architecture is that any real-time (or near real-time) interactions between the telescope and the instrument usually must be handled in a way that circumvents the main design: the TCS and ICS must communicate with each other, or be coordinated by some external means such as a timing bus. A typical example of this type of interaction is synchronizing data taking with the rapid motion of a nutating M2.

However, many radio observatories, particularly but not exclusively interferometers, have for many years been built with a somewhat different architecture, with the separation between the instrument and the telescope not so readily apparent. There is typically a “control” system (sometimes called the “monitor and control” system) and just one other system – the *correlator*. The correlator is responsible for the near-time processing of the signal data and writing signal data to the archive. Most other responsibilities – sequencing, instrument control (i.e. receiver setup), etc. – are carried out by the “control” system. In contrast to the usual optical/IR design, there is no separate system for the instruments. This is probably for two reasons: firstly there is only one type of instrument, not a varied selection, and secondly there is often a tighter coupling between a telescope system and receivers, with motion of the telescope being synchronized with the acquisition of data. This second reason also applies to single dish radio telescopes, where complex “jitter” patterns executed by a subreflector must be coordinated with data taking.

In both designs there is a need for a data archive system. In most modern systems the archive takes responsibility for storing everything; pixel data, metadata, configuration data, monitor data and even logs. It is therefore almost part of the framework software, supplying an underlying service, rather than a principal system in its own right.

The CCAT requirements to support different instrumentation, and in particular visitor instruments, would point towards the "principal systems" architecture. Whichever architecture is chosen, some key points emerge from considering the approaches:

1. The need to support both a heterogeneous instrument suite and visitor instruments is a key driver.
2. The collection of metadata and association with the "pixel" data is very important (both what and how).
3. The nature of real-time interactions between the telescope (or any of its subsystems) and instruments affects the high-level architecture, and is a known deficiency of the most common design; thus these interactions must be considered carefully.

2.2 Telescope Control Systems

The typical TCS consists of a supervisor system, usually running on a non-real-time operating system, which controls a number of subsystems and provides the interface to the rest of the observatory. The subsystems are the true real-time systems and interface to the telescope hardware. They are usually capable of being run stand-alone for engineering and maintenance purposes.

The areas in which a common approach is lacking are:

- How to control the enclosure environment (air conditioning, vent gates etc.). Does the logic needed to maintain the environment during the day ready for night time operations reside in the supervisor, or in the enclosure control system so that the TCS as a whole can be shut down?
- How the weather monitoring fits into the overall architecture.
- Where in the system the astrometric calculations needed to point the telescope and enclosure, position auto-guiders, etc. are done. Does this take place in the supervisor, in a dedicated subsystem or distributed among the subsystems controlling the relevant hardware?

2.3 Instrument Control Systems

Where an observatory supports the use of more than one instrument, the characteristics of the control system usually depend on how the instrument was developed. They can be divided roughly into three categories: facility, visitor, inherited.

A *facility* instrument (also sometimes called a common user instrument) is one that has been built specifically for the observatory and is intended to be available to all of the user community. It will be well integrated with the rest of the observatory software and will probably make use of the same common software as the rest of the control system.

The interface to the rest of the observing system typically consists of:

- A *command* interface that allows the observing system to tell the instrument how to set itself up and to take observations. It also enables the observing system to tell when the instrument has completed its latest instruction.
- A *synchronization* interface to allow the observing system to coordinate real-time operations between subsystems, for example the telescope recording its positions and the instrument taking data frames while the telescope is performing a scan pattern. This could be implemented either in software or hardware or a mixture of both.
- A *data management* interface for transferring the detector data and associated metadata to the data handling system.

The software that controls a facility instrument typically has three components:

- The external interface plus internal supervisor. This interprets the commands arriving, forwards information to the other subsystems, monitors the state of the subsystems and returns acknowledgements to the observing system.
- The control and monitoring subsystem records instrument temperatures and pressures and controls mechanisms.

- The detector subsystem controls the main instrument detectors, sets them up and reads them out. It has to make the data available in some efficient way to the data handling system. Normally the real-time signals are delivered by some means to the detector subsystem, possibly directly to the electronics.

A *visitor* instrument is one that was built either for another telescope or not for any one in particular and is installed on CCAT for a limited period. The users of the instrument will typically be confined to the instrument builders and their collaborators. The control system will not have been designed to interface to CCAT specifically and will probably have relatively simple interfaces to the observatory control system.

The simplest arrangement is where the instrument is completely stand-alone. This means the observing system must allow the telescope to be commanded to point and track and continue tracking in the absence of any other standard operations. The observer then accesses the stand-alone visiting computer and instructs it to take data. The observer is responsible for ensuring the observations are useful by recording metadata such as times and positions. This mode of observing is very limited. It allows only observations of a single field, and any signal modulation (sky chopping or a rotating polarizer) has to be internal to the instrument system.

The chopping-nodding technique for removing sky and telescope background, commonly used in the millimeter and sub-millimeter, requires a real-time connection between the data taking system and M2. There are conceptually two ways of achieving this, and it is likely that CCAT will have to provide both.

- M2 drives the instrument: in this case the M2 control system receives a software command from the observing system giving the required chop parameters, and it outputs a simple hardware signal showing its timing and phase. The instrument receives this in real-time and uses it to label its data.
- The instrument drives M2: in this case the M2 control system receives a software command setting-up the characteristics of the chop (size, direction, speed of movement) but transits from one position to the other only on receiving a simple hardware signal from the instrument. The instrument is then able to demand reflector movements which suit the instrument's internal logic.

An *inherited* instrument is one that already exists, was built (or at least designed) for another telescope and is being integrated into the CCAT system so that it can be operated in the same way as a facility instrument. It has to be made to conform to the other principal system interfaces to allow it to be used fully in conjunction with the higher levels of the observing system such as observation preparation and survey definition tools. Integrating the data handling and (say) temperature/pressure monitoring may or may not be desirable. For example, there may be an existing data processing pipeline designed specifically for the instrument, and it may be more cost effective to continue to use it “as is” and not to try to integrate it with the rest of the observatory systems.

2.4 Data Handling

The data handling system (DHS) might be considered to be generally outside the scope of this study. However, there are aspects that impact on the design of the control system.

The DHS is responsible for collecting, storing and managing the (binary) science pixel or signal data obtained from instruments on the telescope. This aspect is obviously very important, but is not the major concern of this study. However, another role played by the DHS is the collection and storage, *and association with the binary data,* of the metadata about an observation (the FITS header, typically). These metadata are crucial to the later calibration and interpretation of the binary data, and must be complete if any kind of automatic data processing pipeline is in use.

Assembling the metadata is often quite a complex operation. In a distributed system, the required information is usually scattered across many subsystems, and some may also be packaged with the binary data. In addition the information may be required at specific times, e.g. at the start of an "exposure" and at the end. Finally, the metadata must be stored in a structured (searchable) manner and associated with the science data. Note that the metadata might itself be complex, with tables and perhaps even further binary information. This usually leads to a lot of thought being put into the design; for instance, see Goodrich et al.[3] for a modern approach, by the ATST.

Publish-subscribe and *event-based* systems have been common aspects of solutions to this problem and, given the messages that are necessary to do this data collection, one consideration of any design solution must be its performance. The “scatter-gather” design presented in Goodrich et al.[3] is a promising solution, key elements being the broadcasting of metadata request messages and the aggregating of the responses. It is of interest that the designers felt that the core of

their solution should in fact be part of the "common services" framework, basically because the provision and collection of metadata is regarded as a service.

2.5 Software Environments

All observatory control systems run within a software environment of some description – one or more operating systems at the very least. Typically, there is also a range of libraries that implement common functions, and often a framework for constructing the individual programs that make up the complete control system. There are obvious benefits from this approach: there is less code to write and maintain and, if an existing environment is adopted, much of the required functionality is obtained at close to zero cost. However, there are also risks. Learning how best to use a large environment can be a formidable task and it may take considerable time for programmers to become productive. Also, if the environment is not a good match to the problem to be solved it can become a hindrance rather than a help.

There is quite a wide range of environments in use in observatories today, each with its own strengths and weaknesses. Some focus on a particular area, such as the control of hardware devices, whereas others attempt to provide a framework for every aspect of the control system.

One critical function that all environments provide is communications between processes running on different computers – for the obvious reason that partners in any data exchange have to be compatible, so some standardization is essential. The style of communication falls into one of two camps:

Remote procedure or *command/response*: one process sends a request or command to another process and receives one or more replies. The exchange is strictly bilateral and every reply is firmly tied to a particular request. This style is natural for applications involving sequences of operations but is unwieldy for tasks such as monitoring and logging.

Publish/subscribe or *distributed database*: the sending process writes data without regard for who is going to read it, and the receivers read the data they are interested in, either when notified that something has changed or at some periodic interval. This style is typically capable of higher bandwidths and lower latencies than the remote procedure style, and is particularly suited to monitoring and logging activities. However, because there is no synchronized two-way exchange of messages between processes, care is needed to avoid race conditions when implementing sequences of operations. Race conditions, where the outcome depends on the detailed timing of processes on different computers, are particularly difficult faults to detect and diagnose, and the system may appear to work almost all of the time and fail only very occasionally or when some seemingly innocuous change is made.

Some publish/subscribe implementations implement a quality-of-service concept where, if messages are not delivered within a specified time (or not at all), both publishers and subscribers are notified of the failure. This enables a deterministic messaging system to be implemented on top of a publish/subscribe protocol. Some, but not all, existing environments support both mechanisms.

3. CCAT REQUIREMENTS

The requirements for a telescope control system are well known and understood, and there is nothing about CCAT that requires functionality that hasn't already been implemented for some existing telescope. (The same cannot be said, necessarily, for performance.) However, the combination of features required for CCAT is probably unique, and so the chances of finding an existing control system that satisfies all the requirements are negligible, but there may be systems that match closely enough to be a useful starting point for implementing the CCAT system.

The significant characteristics, as far as the control system is concerned, are:

- Segmented primary with active control of segment positions.
- Chopping (or nutating or wobbling) secondary.
- Enclosure (Calotte design).
- Scanning with the telescope mount.
- An auxiliary optical or IR guiding telescope.

The CCAT control system will have to support a range of observing modes:

- A "survey mode" using facility instruments, with the telescope operated by CCAT observatory staff from the local operations facility (some tens of kilometers from the telescope itself but on the same private network). It is envisaged that, eventually, this mode will be completely robotic, with the operator intervening only if a fault or some other unexpected event occurs.
- Queue scheduled (or "service") observing using facility instruments, where observations are done by CCAT staff, either from the local operations facility or from some more remote location (possibly using public networks). In this mode several instruments may be in use, with the observer selecting which program to execute as atmospheric conditions change.
- Classical observing, where visitors operate the telescope (with assistance from CCAT observatory staff).
- Commissioning and other engineering activities, such as calibrating the active optics and telescope pointing.

Observatory staff can be expected to have an in-depth knowledge of the system, understand the implications of the decisions they make while operating it and be able to diagnose and fix problems from detailed, but perhaps somewhat cryptic, error reports. On the other hand, a visiting observer will want a telescope that is simple to operate, does not present the user with baffling choices and gives easy-to-understand error messages when something goes wrong. (The same is not necessarily the case for the instrument control system, where the astronomer may need the option of being able to fine tune the behavior.)

Service observing is typically highly automated, with the observer's role being to select which observation to do next and to intervene only when something goes wrong. This requires a control system that lends itself to the generation of pre-programmed observing sequences that can be made reliable in operation and able to produce data of consistent quality. These will be written by software engineers with an in-depth knowledge of the system and who are happy to use heavyweight software engineering techniques where necessary. A visiting observer, on the other hand, needs a system that can be programmed quickly and easily. He or she wants to get a result that works with the minimum of effort but doesn't need something that necessarily handles errors gracefully or is completely automatic. However, even a completely service-observing-oriented observatory needs a system that can be operated hands-on some of the time, for commissioning new instruments and developing new observing modes.

The extent to which remote (i.e. from outside the CCAT internal networks) operation is required is another important consideration. This could either be for science observing, or, if the observatory is to rely on technical support from its partner institutions, engineering. For example, some communications protocols perform poorly when faced with the long round-trip times inherent in intercontinental links and others require firewall configurations that may be unacceptable to some institutions.

The way in which the telescope construction is organized has significant ramifications for the choice of software. If all the software that is specific to CCAT is implemented in-house, the project will have a sizable team of engineers, who will become experts in whatever software systems and tools have been adopted. Skills and experience gained writing one subsystem will be transferred to the next, and new and better ways of doing things will be retrofitted to existing systems to improve their maintainability in the long term. Also, the same engineers who built the system will probably be available during commissioning, and even on into observatory operations.

However, if the software for each subsystem is to be provided by the manufacturers of the subsystem hardware, the situation is quite different. The engineers writing the software will be working in isolation and, if they are obliged to use infrastructure software mandated by CCAT, will probably be using the software for the first time. This results in *every* subsystem being a "first try" at using the software, with no opportunity for learning from mistakes and correcting bad design decisions. Some of these problems can be mitigated by the project providing extensive training and support (at not inconsiderable cost) but this can run into conflict with the barriers imposed by commercial nature of the contracts between the project and its suppliers. The seriousness of these issues depends a lot on the nature of the suppliers – they will typically be less of an issue with academic or research institutions, particularly those affiliated to the project partners in some way.

There are other issues that are not so easily addressed. The knowledge of the chosen software system gained by a commercial manufacturer may have little or no value to the company once the contract is completed. This means that there is no incentive to re-work parts of the system when it is realized that, with hindsight, there is a better way of doing something. Also, there is no opportunity for lessons learned on one subsystem to be applied retrospectively to another.

The alternative is to allow each manufacturer to write the software in whatever way (within reason) he chooses; leaving the project's own engineers to integrate it into system has a whole. Although this requires more effort from the in-house software team, there will be compensating savings because there is no longer a need to support external users of the software. This undoubtedly means that subsystems can be purchased for less, but has major implications for the long-term maintenance of the system. It also probably means that more effort will be required from the manufacturer during the commissioning and verification stages of the project.

It is often said that the long-term software maintenance costs for an installation with the expected lifetime of CCAT will dwarf the initial development costs. This is certainly true if we include the integration of new instruments, upgrades to enhance performance and the changes needed when obsolete hardware is refurbished or replaced. Judging whether a system is likely to be easy or difficult to maintain requires some knowledge of who will be doing the maintenance. Will there be a group of full-time professional software engineers or will the system have to be maintained as a part-time activity by astronomers? Other models are possible; perhaps the maintenance can be outsourced to another organization that uses the same basic software (some link with ALMA being the possible scenario here).

Another factor is how often new instruments have to be integrated and commissioned. For CCAT this is likely to be once every 2 to 3 years, but this is hard to predict over the long term. Some breakthrough in detector technology could result in a flood of new instruments.

Though graphical user interfaces for engineering are useful – enabling the configuring and testing of systems in an engineer-friendly manner – there is little doubt that during integration and testing, and then during commissioning, the ability to be able to control systems and subsystems from a command line, and to execute scripts that perform integrated actions, especially for automating tests, is invaluable. The more open question is whether or not this command line or scripting interface has a life beyond the testing and commissioning phases, and therefore how sophisticated it needs to be and how much effort should go into its construction. We see three possibilities:

1. The command line interface is built just for the integration, testing, and commissioning phase, and is abandoned thereafter. In this case it may be a "cheap and cheerful" facility.
2. The command line interface is retained throughout the lifetime of the observatory, seeing continued use as an engineering interface and when new systems (e.g. new instrumentation) are added. This would imply more thought and effort going into its design and implementation.
3. The command line interface eventually forms an integral part of the whole observing software system, as well as retaining its engineering purpose. This is the approach taken by ALMA, where, although observing is entirely driven from GUIs and *scheduling blocks*, underlying these is a Python script that actually coordinates the systems to take data. This does imply even more design and implementation effort, as the scripts themselves need to interact with the scheduling block to access the information stored therein. There are, however, advantages to be gained:
    - Non-standard observing may be performed by writing special scripts that may still be used within the normal observing system.
    - The engineering scripting interfaces and the observing interfaces are kept aligned – with a potential saving in software maintenance.

Though bespoke scripting languages have been used in the past, most observatories now use an existing language. Python is the current favorite, and is easily picked up by reasonably software-literate astronomers and engineers, but there are other choices that could be considered. The consequence of requiring a scriptable command interface is of course that the framework chosen must either come with such a feature, or be easily interfaced to an existing scripting language.

4. CONTROL SYSTEMS SURVEYED

The obvious place to look for suitable control systems is telescopes operating at similar wavelengths, where the functional requirements are likely to be similar, and a total of 16 millimeter/sub-millimeter telescope were considered when selecting the shortlist for further study. We also included six optical/IR telescopes, because despite the different wavelength regime there is still considerable overlap between their capabilities and the CCAT requirements. We

included six longer wavelength radio telescopes, because they use similar observing modes (e.g. scanning) to those expected to be required for CCAT.

Preliminary investigation consisted of inspecting published material available on the web and making some informal contacts. At this stage a number of systems were identified as not justifying further examination because they were clearly unsuitable for some obvious reason.

The observing modes used on radio interferometers are rather different from those for single dish telescopes, and although the basic control of the antenna is similar their most challenging problems are not shared with single dish telescopes. Therefore, we felt that detailed examination of the array systems should not be a high priority.

A total of eight systems were selected for further examination; five were operational telescopes, two were projects currently in the design phase and one a telescope under construction.

5. SELECTION CRITERIA

The shortlisted systems were looked at in more detail; in order to answer the questions below as far as is possible in the time available. They were roughly categorized as issues mainly affecting the construction phase, those pertaining to the cost of long-term maintenance, and others, but recognizing that they are all interrelated. Also, they are not necessarily relevant to all the systems selected, particularly those where it is the software infrastructure that is being considered rather than a complete control system. The aim was to reduce the number of candidate systems to no more than three, for further study. The questions that were asked were the following:

1. Organizational
    - Are there any contractual or political reasons why collaboration is not possible?
    - If the answer to this question appears to be "yes", then it is not worth proceeding unless there is a reasonable prospect of being able to overcome whatever the difficulties are.
    - Can the "donor" institution provide any support for the external use of their system (and what would it cost)?
    - Would the donor institution want to place any restrictions on the use of their code (for example, on distribution to subcontractors or instrument builders)?
2. Financial
    - Would any license fees have to be paid to the donor institution or to third parties?
    - What commercial software would have to be bought by CCAT and what would it cost?
    - Would licenses have to be supplied to sub-contractors?
3. System properties
    - What computing hardware does the system run on?
    - What I/O devices are supported and are all of them commercially available?
    - What operating system does it run on?
    - What external written software does it depend on and where does it come from?
    - What support is available for these products?
    - Are they widely used in astronomy or elsewhere?
    - What programming languages are used?
    - What other software development tools are used?
    - How "large" is the system? e.g. expressed in number of lines of source code or pages of documentation.
    - How much effort is currently devoted to software maintenance?

- What styles of user interface (GUI, command line, etc.) does it support?
- Does the system support remote access, and what functions are available to a remote user? How are security and authorization issues addressed?
- How does the system handle the export and archiving of science data?

4. Implementation issues
   - Which of the CCAT functional requirements are not covered by the existing system?
   - How much adaptation would be required to make the system meet CCAT's requirements?
5. Maintenance
   - What level of support would the resulting system require (both numbers of engineers and required expertise)?
   - Can the project offer any support for an external use of their system, and if so, at what cost?
   - Is support available from a third party?
   - Will the maintenance engineers need any unusual skills (e.g. in exotic programming languages or tools)? Are any of the required skills likely to become unobtainable in the foreseeable future?
   - What will be the ongoing cost of commercial software maintenance?
   - What would be the impact on the ease of future upgrades and refurbishments?
   - Would it be easy or difficult to replace a subsystem with a new implementation?
   - How easy would it be to port the code to a new operating system?
6. Instrument integration
   - What interfaces does a facility instrument have to implement?
   - What interfaces are available for visitor instruments?
   - What needs to be done to an existing instrument to integrate it into the observatory control system? (SCUBA-2 provided a good example when evaluating how an inherited system would fit into candidate frameworks.)
7. Miscellaneous
   - Does the project have any expertise or experience that would be of particular interest to CCAT?
   - If the system as a whole is not adopted, are there parts that are particularly successful and might be used in conjunction with software from elsewhere?

It was recognized that many of these questions would not have an answer that could be easily quantified, which made comparing different systems difficult, nor was it obvious how much weight to assign to each aspect.

Another way of assessing a system is to imagine a few scenarios where some change to the software is required at short notice, estimating how much effort would be required and who would be capable of making the required changes. For example:

- During commissioning, a disturbance is observed that is suspected to be a function of the temperature of some part of the telescope structure, so a sensor is installed. What has to be done to record the temperature and correlate it with other data being collected by the control system? Can it be done by a hardware engineer or does he or she need assistance from a software engineer?
- The effect being investigated in the previous example is confirmed and, to correct it, the temperature needs to become an input into an existing control algorithm. What has to be done to achieve this?
- A new scan pattern is required. How is this implemented, and who is able to do it? What needs to be done to integrate the new pattern into normal observatory operations?

- A new observing mode is proposed that involves displacing some M1 segments coordinated with data collection by an instrument. How well does the system architecture handle such an unanticipated new function?

6. BASELINE SELECTION

As we looked at the short listed systems in more detail, it became clear that all were capable of forming the basis of a successful control system but that no existing control system matched CCAT's requirements closely enough to amount to an off-the-shelf solution. Conducting a detailed examination of all nine systems would be costly and, in any case, not guaranteed to select the best; instead, we recommended picking the most promising candidate and implementing prototypes for selected parts of the CCAT control system to see if it was adequate.

We decided that the ALMA common software (ACS)[4] should be evaluated first, for the following reasons:

- The likely availability of engineers experienced in the use and maintenance of ACS and willing to work in San Pedro (either as CCAT employees or under some sort of contract arrangement with ALMA).
- Much of the sub-millimeter astronomy community (both scientists and engineers) will be familiar with ACS or telescopes using it (such as ALMA, APEX and the IRAM 30m).
- ALMA is developing a queuing and scheduling system that could possibly be used by CCAT with relatively little modification. (ALMA has a "total power" mode that is not very different from basic single dish operation.)
- The ACS is supported by a large organization (ESO) with assured long-term funding.

However, we also recognized that the advantages of using LabVIEW as the interface to hardware devices were sufficient that the option of using LabVIEW as a layer between ACS and the hardware should also be explored. Interfacing ACS with LabVIEW has already been the subject of some experiments carried out by ESO[5] (in the context of looking at control system options for the European ELT), and similar interfacing issues are central to planning the LSST. The reasons for recommending LabVIEW be included in the prototyping phase were:

- It supports a wide range of hardware devices including high performance processing units such as field programmable gate arrays.
- It allows engineers who are not software specialists to install and test hardware devices.
- LabVIEW is an attractive option for instrument builders, many of whom are likely to be using it already, and the interface offered to visitor instruments could be LabVIEW rather than ACS.

While evaluating the shortlist we noted that the system in use on the HHSMT and Kitt Peak 12m was cited several times as being particularly successful and held in high regard by users. Although not a complete solution for CCAT, the reasons for its success needed to be understood and taken into account in the design.

7. PRELIMINARY ACS EVALUATION

In order to learn more about ACS and get some hands-on experience, the package was downloaded from the ESO web site and installed on a Linux PC. A simple but realistic prototype application (a pointing kernel driving simulated altitude and azimuth axes) was written as a way of getting some familiarity with the system.

ACS is open source and distributed under the GNU LGPL license, so is free of any licensing costs. It relies on a considerable number of software products from third parties, most of which are common currency in the Linux and open-source world and are included in most Linux distributions. The significant exceptions are:

- The Adaptive Computing Environment (ACE).
- TAO – a CORBA Object request broker from the same consortium of universities that produced ACE.
- JacORB – a CORBA ORB implemented in Java.
- OmniOrb – a CORBA ORB for C++ and Python.

All are available at no cost, but commercial support is available if desired. They are integrated into the ACS distribution, and so they do not have to be obtained and installed separately.

Developers are not required to use any particular software development tools but there is support, in the form of documentation and templates, for the Eclipse IDE (also an open-source product). Applications can be written in C++, Java or Python, but the functionality available from each language is not the same. It is clearly the intention that the low-level (i.e. close to the hardware) parts of the system should be written in C++; the higher level observatory control functions should be in Java and Python, used as the language for observing scripts and the like.

The ACS makes no attempt to disguise the fact that the underlying interprocess communications are based on CORBA. The developer has to define interfaces using the CORBA interface definition language (IDL) and use C++, Java or Python classes generated from that IDL. But ACS does hide much of the complexity of CORBA, and provides a framework for configuring and managing the life-cycles of the CORBA objects that the developer creates.

Informal discussions with a number of people working on ALMA alerted us to some technical issues with ACS that need to be investigated. However, something that is a problem for ALMA is not necessarily a problem for CCAT; any of the following might be true:

- An ACS feature has been used in a way not intended by the ACS designers and a better alternative exists.
- The problem is associated with the scale of ALMA (particularly the large number of antennas) and would not be significant for a single dish telescope.
- A satisfactory technical solution to the problem has now been found.

During the prototyping phase, particular attention needs to be paid to the following:

- *Remote access:* The network protocols used by CORBA are blocked by most typical firewall configurations, so remote access cannot be provided simply by running ACS GUI applications from the remote site. Running the GUI application locally (to CCAT) with the display on a remote X display will work, but the performance is said to be poor (according to the ACS FAQ). This is an issue for ALMA as well, and there is currently work in progress investigating the use of Web front-ends to provide remote access to ACS applications.
- *LabVIEW integration:* ESO have done some experiments[3] with ACS and LabVIEW which demonstrate that it is at least possible to combine the two. Further work is needed to determine whether the technique described in that paper, or something else, is satisfactory. The LSST project has also been investigating techniques for interfacing LabVIEW to other middleware systems.
- *The logging system:* There has been some criticism of the logging system from within the ALMA project. The reasons for this need to be understood and a strategy for dealing with any deficiencies developed.
- *System startup performance*: This is currently a significant issue with the ALMA system and the project is actively seeking a solution. CCAT needs to be aware of this danger, which is unfortunately difficult to test at the prototyping stage. Performance once the system is running is not currently a problem, and CCAT will be a much less demanding application than ALMA in this respect.
- *The configuration database:* The system configuration database is a set of XML files and has been criticized for being too inflexible for ALMA's needs. We suspect that this is because ALMA has a complex subsystem (an antenna) duplicated many times but with small differences for each one. CCAT will not have this particular characteristic so probably will not suffer from this problem.
- *GUI programming:* There appears not to be a full consensus on the best way to implement graphical user interfaces within the ALMA project, and the support for GUIs that came with ACS is now officially deprecated. The main approach now appears to be that GUIs are written in Java as ACS clients – they are not normally full ACS components. Though not an issue that could disqualify ACS, this needs to be investigated further.

8. PROTOTYPE IMPLEMENTATION

We recommended that the choice of ACS plus LabVIEW for the CCAT control system be validated by constructing a prototype that implements a representative set of control system functions. Such a prototype will:

- Demonstrate, as far as possible, that a control system for CCAT can be built using the chosen technologies.
- Test technical solutions to the issues listed in Section 7.
- Select the best implementation strategy in cases where more than one choice exists.
- Define software engineering standards and guidelines (e.g. naming conventions, directory layouts, configuration control, etc.) to provide a solid platform for the development of the full system.
- Enable the CCAT project to evaluate the design by experimenting with actual software rather than having to rely on written descriptions.
- Provide a number of well engineered and well documented applications that can be used as templates or examples during the development of the actual control system.

To achieve these aims, the prototype should be as well engineered as the final system. It will, of course, be incomplete and (probably) not have any hardware to control and, during development, parts will be somewhat experimental and some of it will ultimately be thrown away. However, if successful, large parts of the prototype can be expected to be used in the real control system.

Exactly what should be implemented in the prototype will depend somewhat on what is learned as it is developed but the following would be a reasonable initial plan:

- A mount control system with simulated axes
- An M1 control system with simulated actuators and edge sensors
- A "weather" server
- A simple pointing kernel
- An operator interface
- A simple simulated instrument
- A logging and alarm system

The hardware simulations would be written in LabVIEW and the rest using ACS.

ACKNOWLEDGEMENTS

We would like to thank Gérman Schumacher and Thomas Folkers for their time and hospitality. The study was funded by the UK Science and Technology Facilities Council.

REFERENCES

[1] Radford, S. J. E., Giovanelli, R., Sebring, T. A. and Zmuidzinas, J., "The Cornell Caltech Atacama Telescope (CCAT)," 33rd International Conference on Infrared, Millimeter, and Terahertz Waves (2008).
[2] Gillies, K., Dunn, J. and Silva, D., "Defining Common Software for the Thirty Meter Telescope," Proc. SPIE 6274 (2006).
[3] Goodrich, B. D., Wampler, S. B. and Hubbard, J. R., "Gathering headers in a distributed environment," Proc. SPIE 7019 (2008).
[4] Chiozzi, G., Caproni, A., Jeram, B., Sommer, H., Wang, V., Plesko, M., Sekoranja, M., Zagar, K., Fugate, D. W., Harrington, S., Di Marcantonio, P. and Cirami, R., "Application development using the ALMA common software," Proc. SPIE 6274 (2006).
[5] Chiozzi, G., "Private communication"