Some lessons in specifying interfaces

S. Lee* & K. Shortridge (Australian Astronomical Observatory)

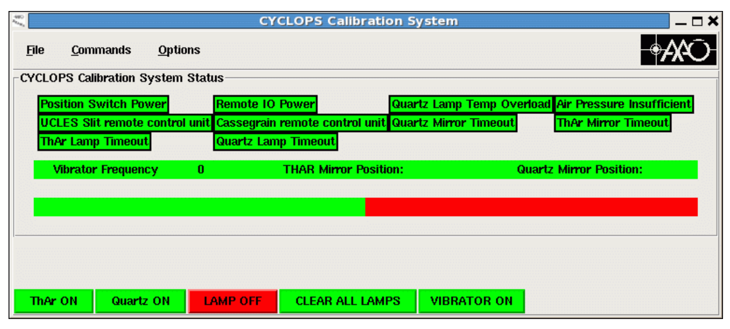
*E-mail: sl@aao.gov.au

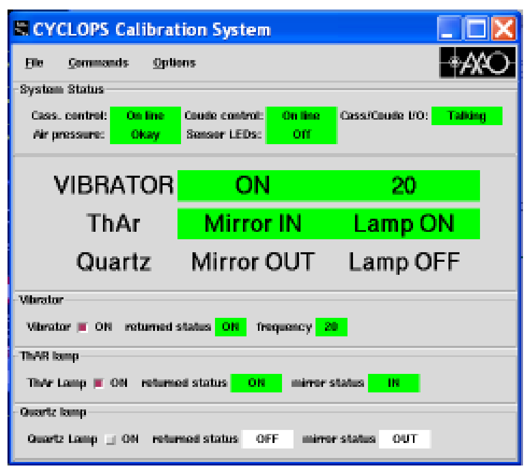
INTRODUCTION

Seen from far enough away, any equipment control system has a layered structure. At the top is the user, and at the bottom is the hardware. Just above the mechanical and optical hardware comes the control electronics, and between that and the user are layers of software: at a minimum a GUI, the bulk of the control software, the operating system, and the drivers for the hardware. Usually, these are built by different teams, and this makes the definition of the interfaces between them particularly important.

Too often, interfaces for a layer start with a functional requirement from the layer above: "we need to be able to enter a telescope position and then move to it" or "we need to be able to read the switch settings". The implementor of the layer then produces something that does this, but in a way that is most convenient given the internal structure of the layer. The result will provide the function required, but all too often in a way that is inconvenient for the layer above. So the software group will write a GUI that matches the internal layout of the code, but requires the telescope operator to select positions using multiple menus when all they really wanted were two entry boxes next to each other *(see the example at right for an interface designed solely by the software group compared to one designed by the user)*. The hardware group will deliver encoders with a driver that reads the switches, but assumes a polling model, leading the software group to realise (too late) that they really needed an interrupt when a switch setting changes.

A recent AAO project provides a couple of examples of the results of reversing this process. Through some fortuitous project timing we were able to have both the user interface and the hardware driver interfaces specified in complete detail by their eventual users rather than their implementors, and this proved extremely successful.





The interface on the left was designed by the software group, while the one on the right was a re-design by the user. They contain the same information, just displayed differently.

THE AAT TCS

For the replacement telescope control system (TCS) for the Anglo-Australian Telescope (AAT) we already had a good idea of what was needed, as well as how the telescope hardware worked, but replacing the 1970s user interface (UI) was a blank sheet. Despite its vintage the old menu-driven UI was surprisingly efficient, slowed down only by the 1970s computer hardware. What had evolved was a number of additional functions added to the system which ran from a separate command-line terminal, and these had to be integrated into the new TCS and to work as efficiently as the rest of the system.

Due to the long lead time in designing the hardware to allow the new TCS interfaces to co-exist with the old TCS (intended to allow parallel running of the new system with the old system and minimise telescope down-time, and including confirming the existing documentation and hardware of the old TCS - which had 35 years of modifications to verify) there was time available to explore what the new UI would look like.

Importantly, the telescope had experienced operators who not only had specific ideas about possible designs for the new GUI, but also had the software expertise to code up a trial GUI in Tcl/Tk that the software development team could treat as a detailed specification. Separately, the decision to write a detailed hardware simulator[1] for the telescope to allow early testing of the software meant that this user-designed prototype GUI could be tested realistically as the system evolved.

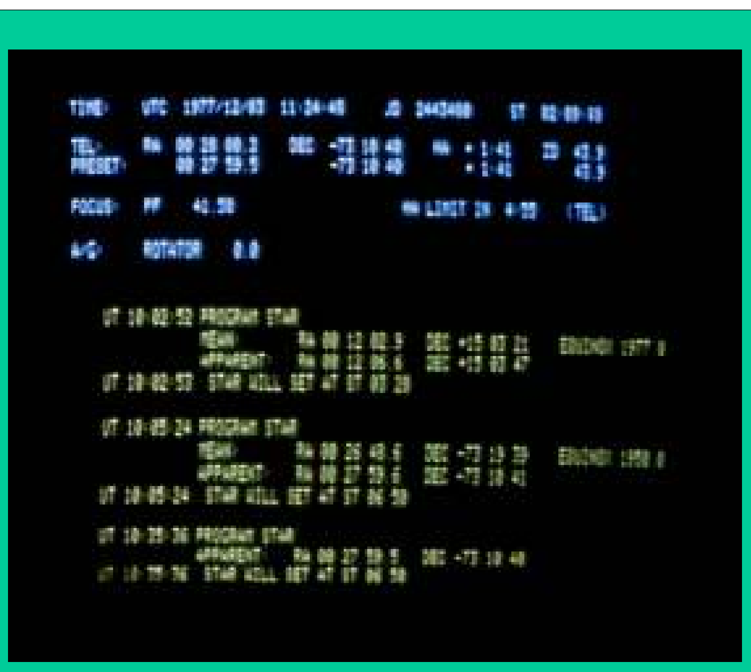
In what might be referred to as 'agile development'[2], the approach taken was to have the telescope operators and observers work with the evolving GUI in parallel with operating the telescope from the old system. This way direct comparisons could be made with the common telescope operations and deficiencies quickly spotted. The original 'specification' GUI was coded in Tcl/Tk, and with no significant code below the GUI – just enough to read and display a catalogue or acknowledge a button press – could quickly be modified to try out different options. One of the software team is also a telescope operator and so undertook the coding of the original GUI and conducted the testing cycle. Having an end user also responsible for the GUI design is an uncommon luxury, but one which ensured that GUI would actually fulfil all the requirements of the users.

It was remarkably quick to make changes this way and was far easier to iterate the GUI at the very beginning of the project than at the end. The major advantage of having the GUI developed before the remainder of the system, apart from the end users being able to have a say in the final product, was that defining the way the user specifies parameters and actions made it clear how some of the lower level code needed to be structured. This is how 'top-down' design is supposed to work, but rarely does.

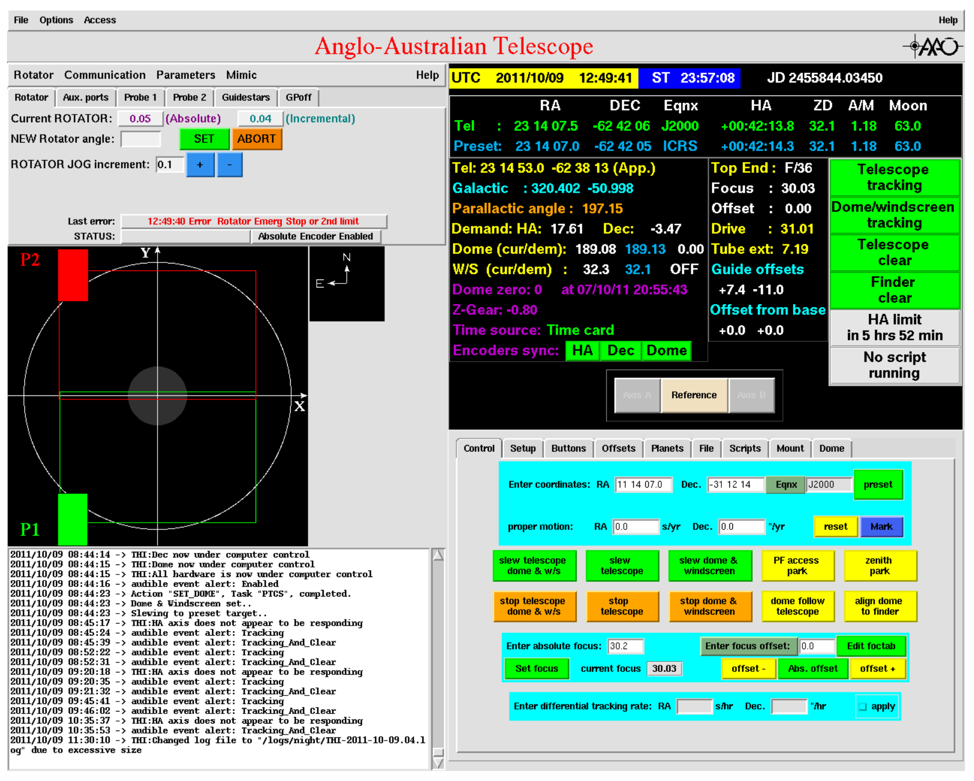
One obvious example is shown to the right; the way the sky mimic display evolved from being a small bit at the bottom right hand of the GUI to taking over an entire section. In other words, although we were close, we didn't know exactly what we really wanted until we were able to play with it..

It became apparent that this technique of having the higher level tasks defining the lower level procedure calls made the software effort smoother. By starting early with a user-specified GUI, and by testing it with a very full simulation of the hardware, most of the potential issues with the GUI were solved long before any of the new hardware interfaces to the telescope became available. It also became apparent that left to themselves, with limited contact with the users, the software team would have written something significantly different and less satisfactory (but easier to implement).

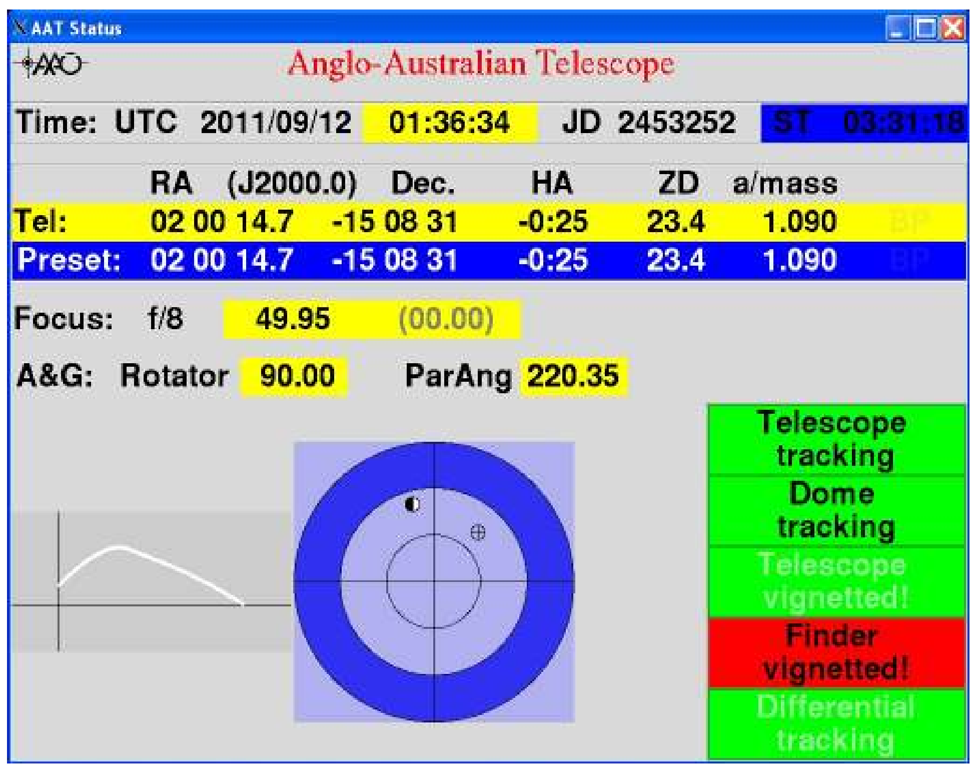
At the same time, the hardware interfaces were being designed and the driver interfaces specified. In this case, we were again lucky. Since software is usually the last thing finished in a project, the software group is still busy finishing off the last project when the hardware group designs the interfaces for the next, and so has insufficient time to notice potential problems with the proposed hardware interfaces. By chance, there was software effort available to think about the interfaces for TCS before they were fully specified, and the initial driver specification was written by the software team. After some iteration – not everything specified by software was possible with the actual hardware – what evolved fitted very nicely into the software design. A hardware-driven interface design might have been easier to implement, but would probably have been harder to use asynchronously, and might not have made available the precise timing information needed by the software.

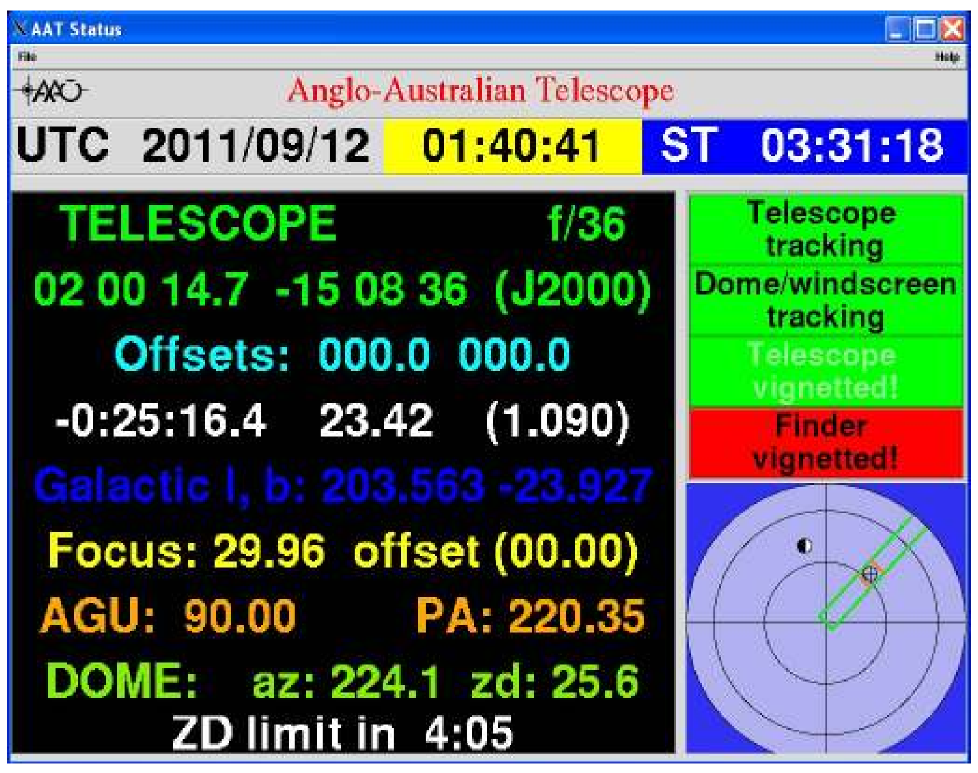


TCS information display circa 1970s. The evolution from this to the first attempt at the new status display (shown directly below) is obvious.

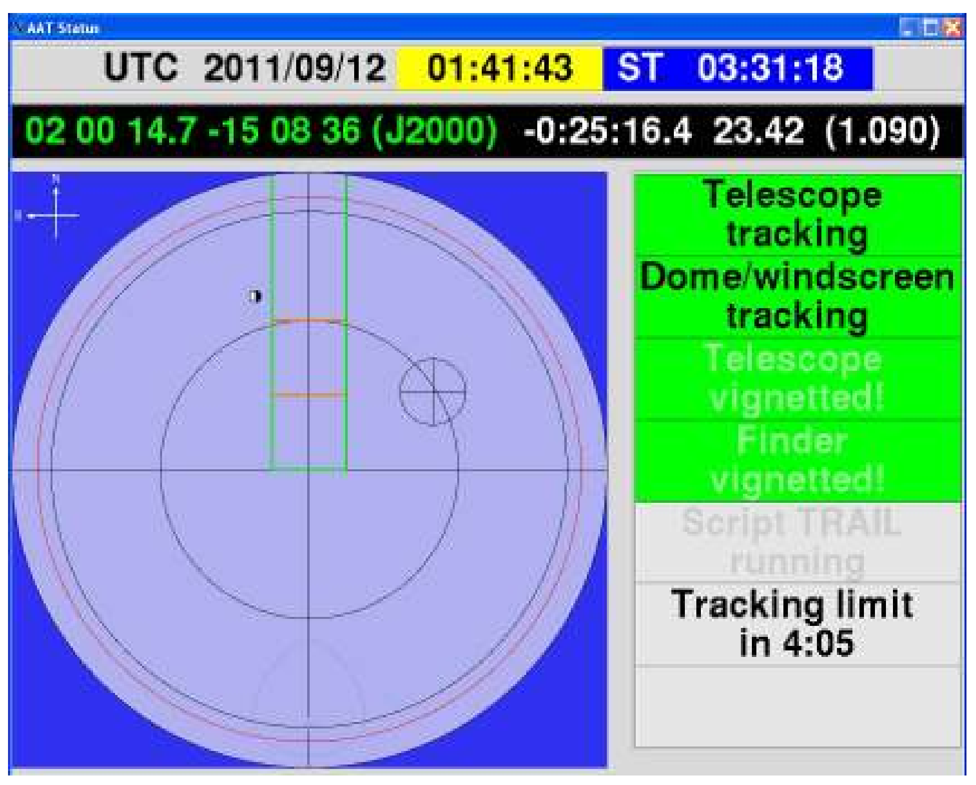


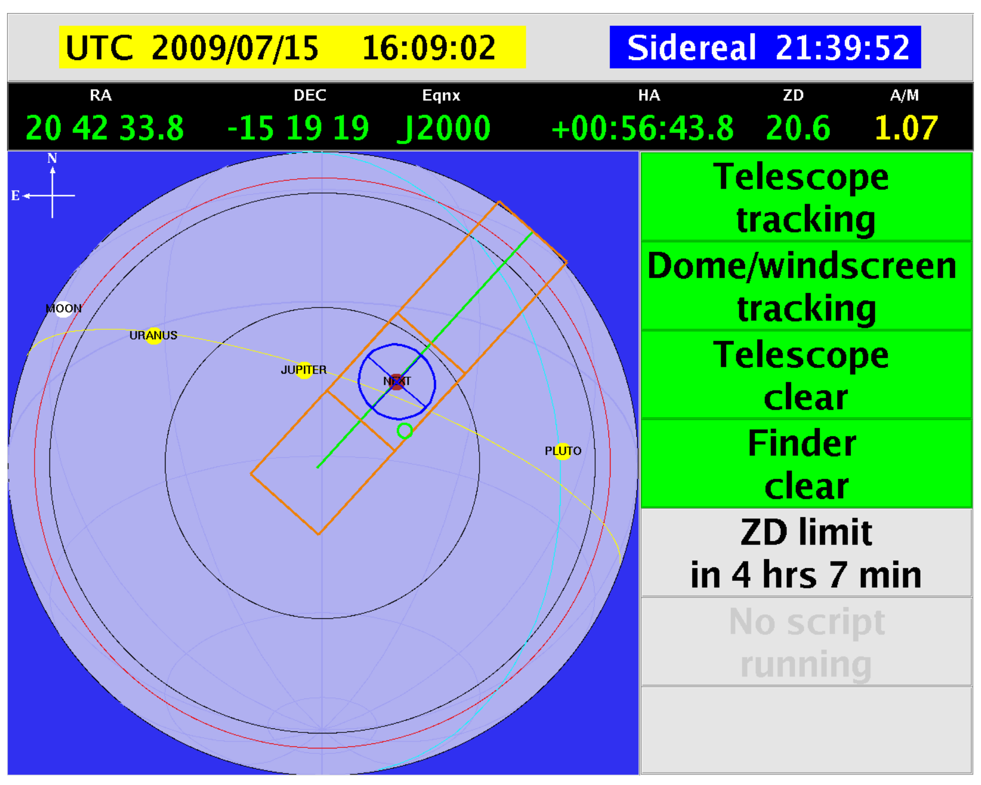
The new TCS control GUI, designed to help define the interfaces to the lower level code.


AAT Status
Anglo-Australian Telescope
Time: UTC 2011/09/12 01:36:34 JD 2453252 ST 03:31:18
RA (J2000.0) Dec. HA ZD a/mass
Tel: 02 00 14.7 -15 08 31 -0:25 23.4 1.090
Preset: 02 00 14.7 -15 08 31 -0:25 23.4 1.090
Focus: f/8 49.95 (00.00)
A&G: Rotator 90.00 ParAng 220.35
Telescope tracking
Dome tracking
Telescope vignetted!
Finder vignetted!
Differential tracking



Evolution of the status display for the AAT TCS. Bottom-right is the final product, while the other panels show the design evolving after user feedback. Note how the small mimic showing the telescope position on the sky grew from a small section with lots of text-based information surrounding it into becoming the dominant conveyor of information with only the minimum of additional text used.





CONCLUSION

In summary, we were fortunate to be able to run the TCS project the way we did, but the success we achieved suggests that other projects would benefit by deliberately trying to specify their interfaces in this 'user- rather than implementor- designed' way.

REFERENCES

[1] Shortridge, K., & Vuong, M., "Faking it for pleasure and profit: the use of hardware simulation at AAO", Proc. SPIE, 7740, ed. N. Radziwill, & A. Bridger,p. 774008 (2010).
[2] http://en.wikipedia.org/wiki/Agile_software_development.