The software design of the Gemini 8m telescopes

Stephen Wampler

Gemini 8m Telescopes Project, 950 N. Cherry Ave, Tucson, AZ 85726

ABSTRACT

The design of the software for the Gemini 8m Telescopes is nearly complete. Great care has been taken to develop a system with the flexibility to support astronomy into the next century without disassociating itself from the current methods of observing. The goal has been to design a system that supports the complexities involved in high performance telescope operation while providing an interface that is easy to operate in all modes from classical observing to modern queue-scheduled approaches. The resulting design has been crafted to augment the skills of observers and system operators and has produced a development strategy intended to encourage the interaction of scientists and developers.

Keywords: telescope software design, design processes, Gemini software

1. INTRODUCTION

The Gemini 8m Telescopes Project software design is nearly complete, with the implementation of the software now in progress at a number of sites. The project has been faced with a number of challenges that are now familiar to software developers of large telescopes - tight schedule, fixed (and limited) budget, and stringent science requirements[1]. Additionally, the Gemini Project has a distributed development environment, with software development teams spread across several continents.

This report describes the process used in to develop the design and presents an overview of the design[2]. The development process was specifically designed to address the above challenges and constraints. The design itself addresses the goals established for the software to provide an efficient, flexible system that can meet the needs of a state of the art observatory facility both now and in the future. The report concludes with a summary of the current status of the software development process.

2. SOFTWARE GOALS

The software design process is a flow that originates from a series of goals[3] that identify the fundamental characteristics of the observatory software and terminates in the working software installed and in use on site at Mauna Kea and Cerro Pachon. Software requirements are produced from both the software goals and the overall Gemini science requirements. The software design addresses these requirements as well as the original goals. While the goals are embodied in the requirements, it is important that the goals remain prominent in the design process to ensure that the design as a whole does indeed satisfy the goals. If the design had focused entirely on the requirements, it is less likely that the result would present a consistent and well-integrated solution to the goals.

Some of the major goals for Gemini software are:

(1) *Consistency and usefulness in the user interfaces*

The Gemini 8m Telescopes Project is managed by the Association for Research in Astronomy, for the National Science Foundation, under an international partnership agreement.

(2) *Flexibility in the high level software*

(3) *Efficiency in the low level software*

There are, of course, a number of other goals including *reliability*, *extensibility*, and *maintainability*. However, the first three goals are the fundamental precepts of the design.

In the end, the purpose of an observatory is to provide astronomers with tools for acquiring science data of the best possible quality. In this context, then, the role of observatory software is to provide the connection between astronomers and the observatory systems. With the sophistication of modern large telescopes, coupled with the large costs to build and operate these telescopes, the software must provide an interface that is intuitive, easy to use, and complete in the sense that it must be possible to access, monitor, and control all components of the observation system that affect the quality of the science data. These are, in some sense, conflicting tasks. Ideally, the software system should allow astronomers to think in terms of the science they want to do instead of in terms of the systems they must control.

At the same time, both technologies and science advance. New instruments are developed, new methods of using existing instruments are thought of, and new scientific questions are posed that require new ways of controlling observing systems. Observatory software should be flexible enough to adapt to these changing conditions. The cost of throwing out an entire software system and replacing it is prohibitive.

Finally, the anticipated demand on modern large telescopes and the cost of operating these observatories place a premium on the efficient use of the system. Mechanisms should operate independently and concurrently as much as possible, and the software must support this behavior. Additionally, policies such as service, preplanned, and queue-based observing must be supported by the software in order for these policies to be effective.

3. SOFTWARE CHALLENGES

The Gemini project faces many of the same challenges found in most large telescope construction projects:

(1) *Fixed (and limited) funding*

(2) *Tight schedule*

(3) *Stringent science requirements*

In addition, the Gemini project is further constrained by having a

(4) *Distributed development environment*

Having a distributed development environment does have advantages - for example, Gemini has been able to use some of the best people available in the astronomical communities of the partner countries. However, the problems inherent in designing and implementing software across multiple time zones, countries, continents, and organizations cannot be minimized. The design process has been developed, from its inception, to work as well as possible in such an environment.

4. DEVELOPMENT STRATEGIES

The core of the Gemini software design was developed by a small team of developers at the International Gemini Project Office in Tucson (IGPO), Arizona USA. This approach allowed the design team to work closely together to develop a design ‘overview’ that retained consistency and concentrated on the overall goals of the software. Once the design overview was developed, the design was split into *work packages* that could then be distributed to organizations in the partner countries. This allows the work to proceed in parallel.

There are several key aspects of this process. First, the central team did not attempt to provide a full, detailed design for the software. The size of the effort precluded a small team from completing such a design in a reasonable amount of time. Further, it would have left the project with a design developed entirely by one group, yet implemented by other groups. These other groups include very capable people with specific knowledge that matches well with many of the tasks required of the software. It makes sense to involve these people in the detailed design of these parts of the system. This also provides an intellectual challenge to the development groups which is often beneficial psychologically. People take a more active interest in system they help design, and can catch and correct problems more readily.

Second, by concentrating the design overview in a small central team while distributing the find design and development to a wider group there is a group that identifies and understands the fundamental goals and philosophies of the software design. This group can then devote more effort to keeping the entire project moving towards these goals. This is a crucial role in any large distributed development environment. No matter how much effort is put into keeping communications strong, individual groups within a distributed system have a natural tendency to concentrate on their individual tasks. This often results in these groups drifting from overall goals and concepts. While a formal review process allows for detecting and correcting such drifts, it is difficult to use reviews effectively without impacting schedules. Either you have a few reviews, which means development efforts can drift quite a bit before being detected, or you have many reviews, where the very frequency of the reviews begins to interfere with actually producing results. A central team that is continually watching the development process can help keep the distributed development moving on a consistent path.

Another strategy adopted as part of the design process is to use pre-existing software where possible. Further, commercially provided software is preferred, as it moves the maintenance of this software onto the suppliers. This means the development teams can concentrate on building the software design rather than constructing the infrastructure on which the design is built. This is an important point. One goal of this process is to allow developers to work at as high a level of development as possible, concentrating on the design instead of the support infrastructure. This helps keep the design focused on the goals and helps maintain consistency.

Where commercial software is not available or appropriate, community-based software with a wide support base is also preferred over developing the software locally.

A significant portion of the Gemini software design is based upon infrastructure software that was obtained externally as either commercial software or as community-based software. The EPICS (Experimental Physics and Industrial Control System) package provides the infrastructure needed to communicate with and control most of the real-time systems while higher level communication and control is provided using Neo - Sun's CORBA implementation. Other community software is used to package and transfer science data (both images and header information) throughout the system. Gemini uses the IMP message protocol and SDS structures developed at the Anglo-Australian Observatory for data transfers. Image display is based upon the RTD (Real Time Display) widget developed as part of the European Southern Observatory's VLT project.

5. DESIGN STRATEGIES

A fundamental principal of the design is that the major components of the system operate as autonomously as possible. These major components cooperate to provide the functionality required for observatory control. This approach allows greater independence of the development teams and also helps increase the modularity of the system. The intent is to construct a software environment where each component has a well defined role to perform and is able to do so without the need for detailed control by an external system. Commands that pass between systems should be high-level and reflect the tasks that need to be completed to acquire high quality science data.

The general method for directing major components is through configurations. A configuration contains all the information that is needed to change the state of a component. This information is represented as a set of attribute-value pairs, where the attribute identifies some key characteristic of the component that the value applies to. Data that are typically represented as attribute-value pairs include target position information, exposure times, chopping positions and frequency, filters, and so on.

Issuing a series of commands to set target position, move filters into position, etc., would impose a sequence of events onto each component from some other component. This requires that the 'commanding' system understand the implementation of the 'commanded' system well enough to issue these commands in an appropriate sequence, which complicates the interface between the two components. In the Gemini software design, the commanding system would provide a new configuration to the commanded system and then direct it to match that configuration. Now the responsibility for performing actions in an appropriate sequence is entirely within the commanded system. This enables this system to perform actions in parallel or to serialize events as needed.

The risk in adopting such a strategy is the loss of system flexibility - there may be special circumstances where there is a need for one system to direct other systems using more fine-grained control. This is expected to happen during commissioning and when there is a need to control the telescope or instrument in some novel, unexpected manner. The design supports fine-grained control by imposing no restriction on the amount of information contained in a configuration. Smaller configurations allow finer control of the sequence of actions in a commanded system, providing greater control. Larger configurations allow individual components more autonomy of operation, allowing components greater control internally.

Another concept embodied in the design is that status information is provided by a message bus. That is, status information from a component is made available automatically and interested client applications can attach to the message bus and subscribe to specific status information as needed. This eliminates the need for establishing application-to-application connections for status information and provides a great deal of flexibility in monitoring status. It becomes easy to reconfigure a client to monitor different status items and to initiate this monitoring, as the source of the status information operates independently of the actual monitoring.

One of the risks involved in distributed development is that it easy for user-interfaces to reflect the distributed development. Pieces of the interface may vary widely in look-and-feel from one component to another. While standards can help ameliorate this problem, they cannot eliminate it without becoming overbearing and too restrictive - the resulting interfaces are often less than pleasing and rarely innovative. In the Gemini project, all user interfaces used during the routine operation are the responsibility of a single team. This team is free to develop user interfaces that are both consistent and of a high quality - appropriate for use at high altitude observatories. However, the team does not develop the user interfaces in isolation. Communication between the user interface team and all other system development teams is expected and planned for, as are the use of prototypes to get user feedback early in the interface development process.

6. AN OVERVIEW OF THE DESIGN

A high-level view of the Gemini control system is shown in figure 1.

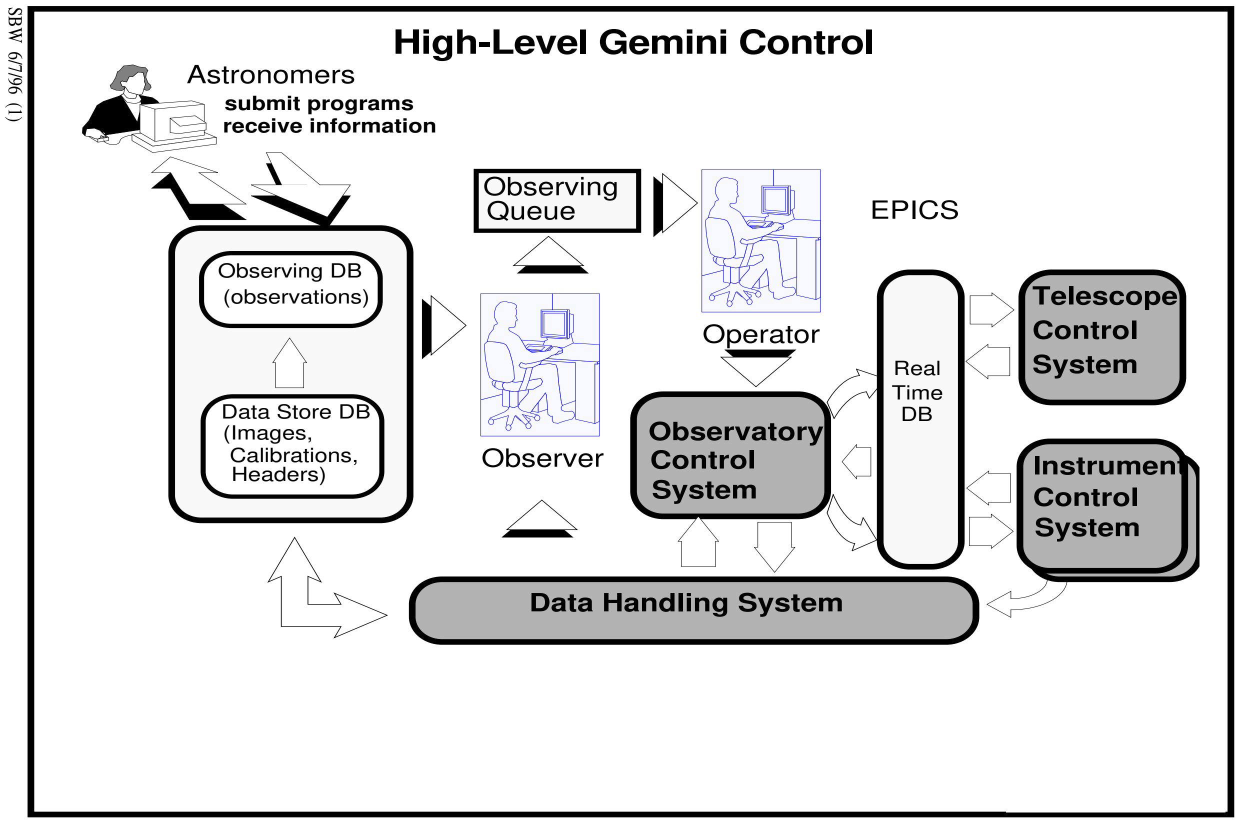

Figure 1 - A high-level view of the Gemini control system

Interaction with the astronomer, whether on-site or remote, is through an *Observing Tool* application that provides the astronomer with the ability to create and interact with a *science program*. The astronomer uses the observing tool while planning observations and configures the observatory systems through a series of consoles presented by the observing tool. These consoles can be configured and dropped into the science program for queue and service observing or they can be configured and acted on immediately by the system during interactive observing. Each set of consoles that are required to fully configure the system for a given task comprise an *observation* within the science program. For example, if there are several targets that must be observed as part of the observing, then there will be several observations within the science program.

Science programs are kept in an *observing database* within the control system. The on-site staff observer is able to examine the individual observations within the science programs contained in the observing database to select and schedule the most appropriate observations for the current conditions.

As observations are performed by the system, the science programs in the observing database are adjusted to show any changes to parameters that were made while they were executed (for example, if the exposure time was shortened because better than expected conditions allowed the data to be collected more rapidly). Any data that is collected during the observing process is also available through the science program as it becomes available. The science program thus changes from a plan on how to perform observations into a record of the observing process. An astronomer can use the observing tool to examine the progress of a science program at any time.

Figure 2 shows the observing tool displaying a partially completed science program. (This is a prototype version of the observing tool and is missing some of the functionality required in the final version.) The observer has opened up the science console and is reviewing the parameters that were actually used with the (now completed) first observation. The second observation has been scheduled, but not yet started.

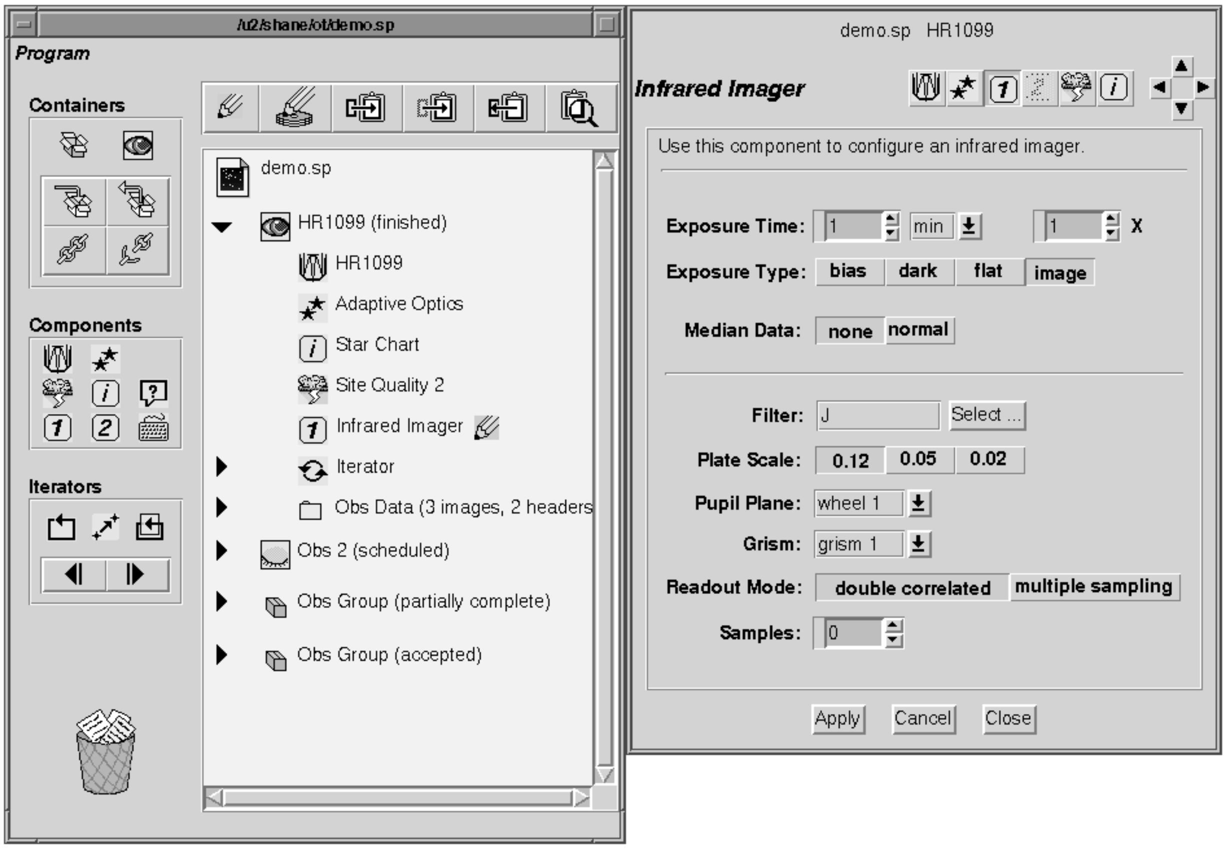

Figure 2 - Prototype observing tool and sample science console

As the on-site observer selects observations from the pool of available science programs, the observations enter the *nightly plan*. This nightly plan serves as the queue of observations to be performed each night. The observer can construct several nightly plans and the system can switch between them to meet changing conditions. The system operator allocates the observatory resources to the observations in the observing queue as the night progresses.

The control system software is organized into four *Principal Systems* each with a specific role in producing high-quality science data.

1. The *Observatory Control System* (OCS) is responsible for supporting the on-site observer and system operator in their tasks and coordinates the activities of the other principal systems by routing the appropriate portions of the configurations in each observation, waiting for all the systems to match these system configurations, and then directing the systems to acquire and process the data.
2. The *Telescope Control System* (TCS) handles the observatory subsystems such as the telescope and enclosure. The TCS must acquire and track the target while keeping the image quality delivered to the instrument as high as possible.
3. Each *Instrument Control Systems* (ICS) handles one of the instruments mounted at Cassegrain focus. Each ICS sets up the instrument and the detector, controls the exposure and detector readout. Each ICS supplies the science data and associated header information to the DHS for processing.
4. Finally, the *Data Handling System* (DHS) accepts data from instruments and other sources, archives the data, and processes and displays the data for quality control decisions by the observer and operator.

Communications with the TCS and ICS systems are through the EPICS system discussed earlier. EPICS presents these systems as part of a *real-time* database. Applications can direct the actions of the systems by setting values into the database and can monitor status information by subscribing to information contained within the database.

7. A VERTICAL SLICE THROUGH THE DESIGN

Figure three presents the software design from a slightly different viewpoint to show the control levels within the system and the control and data flows between the various systems and subsystems.

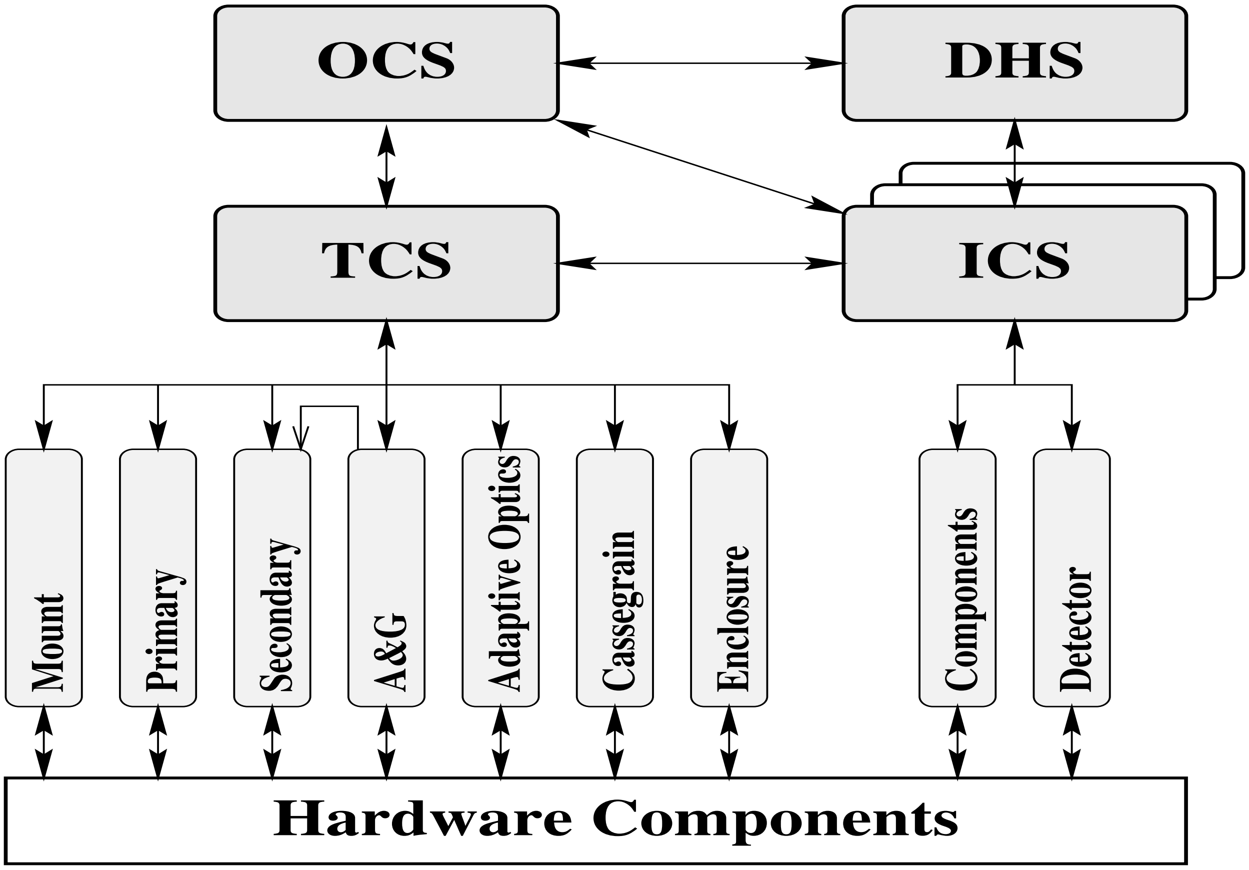

Figure 3 - A vertical slice through the software design

Here, the four principal systems are arranged in a loose hierarchy, with user interactions through the Observatory Control System and the Data Handling System. (Engineering staff can, of course, access the systems at any level in the hierarchy.)

The subsystems of the Telescope Control System and an Instrument Control System are also shown. The top level of the TCS is responsible for maintaining the pointing and optical models and for sequencing the activities of the TCS subsystems. Normally, the TCS subsystems do not communicate directly with each other. The exception is that fast tip/tilt/focus information must be transmitted by the Acquisition and Guidance system directly to the Secondary Mirror Control System. Even here, however, the TCS establishes the parameters for the control that results from this information flow.

To illustrate the distributed nature of the development, the following list shows the institutions responsible for each of the systems shown above. Not all the instrument development sites are given here, but they are equally well distributed.

System	Institution
OCS	Gemini Project Office, Tucson, USA
DHS	Herzberg Institute of Astrophysics, Canada
TCS	Royal Greenwich Observatory and Rutherford Appleton Laboratories, UK
ICS	Various sites, primary site at Royal Observatory Edinburgh, UK
Mount	RGO
Primary	RGO
Secondary	ROE, and commercial vendor
A&G	ROE, and commercial vendor
Adaptive Optics	HIA and ROE

Cassegrain	RGO
Enclosure	HIA

8. CURRENT STATUS

At this time, the design is nearly complete, with most systems past their preliminary design reviews. The one exception, the Data Handling System, is scheduled to complete its PDR in a few months. Most of the infrastructure is in place, with IMP, EPICS, and Neo working well to provide the majority of this infrastructure. The TCS pointing algorithms are complete and implemented.

While most of the major interfaces are now complete, there are still quite a few details on the specific interfaces that are being worked on at this time. These will be finalized as the detailed design phases continue.

Several systems have produced prototype versions of portions of their software. As these prototypes are being developed, the IGPO is combining these into a prototype simulation system to test the interfaces, the functionality, and the integration process. The currently implemented prototypes and connections in the simulation system are shown in figure 4.

9. ACKNOWLEDGEMENTS

The overall design of the Gemini control system was done by Steven Beard, Kim Gillies, Peregrine McGehee, and the author under the direction of Rick McGonegal at the International Gemini Project Office in Tucson, AZ, USA. Steven Beard was on loan from the Royal Observatory Edinburgh. Other significant contributors to the software design include Andrew Johnson, Robert Laing, and Christopher Mayer of the Royal Greenwich Observatory, Pat Wallace of Rutherford Appleton Laboratories. Many others have now contributed to the detailed designs.

The Gemini 8-M Telescopes Project is managed by the Association of Universities for Research in Astronomy, for the National Science Foundation and the Gemini Board, under an international partnership agreement.

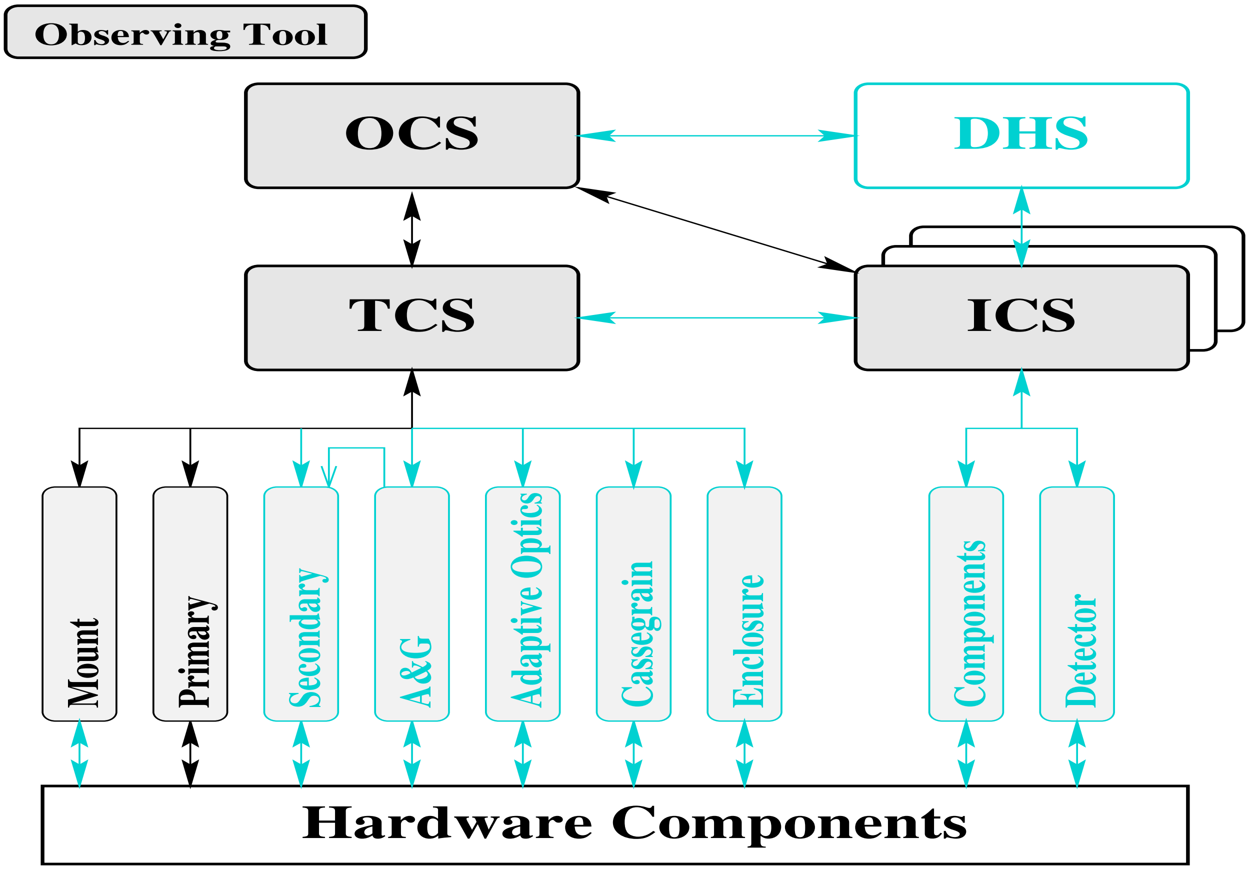

Figure 4 - Prototype systems and interfaces

10. REFERENCES

1. R. McGonegal and S. Wampler, "Managing an Internationally Distributed Software Project", *Proceedings of the 1995 International Conference on Accelerator and Large Experimental Physics Control Systems*, October, 1995.
2. S. Wampler, et al, "Software Design Description", *Controls Group, Gemini 8m Telescopes Project*, 1994.
3. S. Wampler, "Goals and Requirements for Software and Controls", *Controls Group, Gemini 8m Telescopes Project*, August, 1993.