GBT MEMO 89

GBT Monitor and Control Design Rationale

Mark Clark and Rick Fisher
September 16, 1992

This note is an attempt to translate the basic design concepts and rationale of the GBT Monitor and Control system (GBT M&C Requirements Analysis Model, Aug. 3, 1992 [draft]) into not-so-technical English and to highlight the assumptions that have driven the design so far. The Monitor and Control group has been assigned the task of building a computer system for coordinating the actions of all subsystems of the GBT under observer or telescope operator control. This group also has the task of displaying all system monitor information and putting the astronomical data into a format acceptable to a number of data analysis packages.

Three questions need to be answered at this stage of the project. What is the Monitor and Control group promising to deliver at specified milestones in the project? Are these milestone realistic and frequent enough to measure slippage when corrective action is still possible? Is the present design workable and somewhere near the best that we can do with the current technology?

A list of milestones with dates and one-sentence descriptions is attached to this document. Most include a demonstration of a specific system function to members of the steering committee or other interested persons. The milestones closely reflect completion of various components of the system. The dates on this list are those that must be met in order that the monitor and control system be ready when the telescope is. We are now discussing whether the current manpower is adequate to meet this schedule. The rest of this document addresses the third question above.

Assuming that the control system does what the astronomer wants, the first important system requirement is reliability in the sense that it does exactly what it is told, and it continues to operate predictably under all conditions. The second requirement is that the control software must accommodate additions and changes to the telescope hardware with a minimum of disruption to normal operation. We are asking for a control system that is more reliable than those that exist at most other telescopes, although the telescope is more complex. The GBT Monitor and Control design is aimed directly at taming the complexity.

The control and monitor software tasks are dictated by the requirements specified by the user, which, in turn, largely reflect the requirements of the hardware to be controlled. Not all software objects have hardware counterparts, but nearly every substantial piece of hardware is represented by a software object. However, a hardware block diagram usually shows only the signal path which has simple output/input connections between objects. A software diagram is concerned more with the flow of control and monitor information. Also, the dependencies between objects are much more involved. We want to maintain independence between system objects but, at the same time, avoid duplication of design, coding and testing effort. This requires that we look for similarities between objects, and it requires a fair degree of object abstraction from the control and monitor points of view. The aim is to allow any object or piece of hardware to be removed, replaced, or changed internally with very little or no effect on the rest of the system.

The following assumptions, derived mainly from experience, have played a large part in the design:

First, it is feasible and desirable to distribute the control software among a number of computers on a network and among a number of relatively autonomous processes on each computer. The current design is far enough along to convince us that modern networking software makes this quite feasible for GBT Monitor and Control. Here we address mainly the desirability of distributed processing in this design.

Second, time-critical software functions can and must be isolated to small and very well tested pieces of the system. Control servo and time-critical software is some of the most difficult code to debug and verify, and it the most prone to failure when the processor on which it runs is busy or receives an unforeseen combination of interrupts. This is often the cause of unexplained system crashes or strange and difficult-to-diagnose system operation. The only synchronization mechanisms between time-critical functions in the M&C design are absolute time and common high-level command software. The operation of any time-critical function must not depend critically on the operation of any other.

Third, most of the monitor and control software needs only to be fast enough to provide an imperceptible or completely acceptable delay between command and action or response. This means that very few independent processors are required to isolate time-critical functions. It also means that most of the software can run on standard operating systems, e.g. UNIX, and network protocols that make no guarantees about response time but are easily fast enough on average.

Fourth, software units (functions, routines, classes, etc., which we have been calling modules) must be small, independently verifiable, and require a minimum of communication with other software units. This is an old software maxim that is often violated when deadlines approach. This is implementation issue, and it has a strong impact on system reliability. Even the highest level software modules, which tie the whole system together, may be small and testable because they communicate with relatively few modules at the next lower level in very well defined and restricted ways. Modern software languages, including C++, provide many of the tools needed to implement modularity, and they encourage, if not enforce, minimal communication between modules. The distribution of software to a number of computers and processes is entirely compatible with module isolation. Modules at the medium and large software scale are exactly the same as objects in the design.

Fifth, new modules may be added at any level, an entirely new back-end, for example, with minimal disturbance to the system. Any software module may be removed without affecting the operation of the system except to remove the services supplied by that module. This will be an effective system test for robustness. Any module may be internally modified, optimized, or simulated with no affect on the rest of the system as long as the communication rules for this module are not changed. All of these rules must be enforced to help solve to old problem of mysterious system operation or failure when supposedly innocuous changes are made to the software.

Finally, safety issues must be handled at the lowest software level possible. Interlocks that involve software of any kind must survive a major computer system failure which means that they must not depend on the network or the computer operating system. This is an extreme extension of the isolation of time-critical functions.

The issue of whether a system is or should be synchronous or asynchronous is strictly one of implementation. Possibly better phrases for the concepts that have been discussed at various meetings are micro-managed versus delegated-authority designs, respectively. In a micro-managed system there is strong coupling between the high level code and the modules it commands for various actions. Commands generally are short term, assume little intelligence in the subordinate modules, and require quick response time. In the M&C delegated-authority design, commands are issued to subordinate modules that specify actions over relatively extended periods of time, such as where the telescope should be pointed at all times during a five minute scan. It assumes that each module will execute its commands correctly and on time and that the modules have enough intelligence to detect and report error conditions. Either design can be made to work, but the micro-managed design is much more difficult to divide into well isolated subsystems.

Our arguments are that rigorous software modularization will produce software which can be tested more confidently than has been the case in past telecope control systems and that distributed processing is entirely consistent with this philosophy. This software should be more testable and less vulnerable to software errors introduced by changes or additions to the system, and additions should be less disruptive to normal operation.

Milestones

frontend proto 12/15/92	Demonstration of the basic coordination sequences, control, and monitoring of an emulated prime focus frontend from both the engineer's and observer's point of view using prototype software.
manager 2/ 5/93	Demonstration of the coordinator and various pseudo-managers controlling the iteration of the system through observations.
console 3/15/93	Demonstration of the console control windows for setting up and controlling various devices.
graph display 4/12/93	Menu driven selection and display via meters and/or graphs of monitor values from any of the currently operating hardware.
converter prot 6/ 2/93	Demonstration of the basic coordination sequences of the frontend mixer system for Doppler correction and frequency switching.
frontend beta 7/23/93	Demonstration of the basic coordination sequences, control, and monitoring of an active prime focus frontend from both the engineer's and observer's a workstation windows.
collator 7/23/93	Demonstration of the combining and packaging of data from an actual backend with partial header information and emulated data associated parameters.
mirror 8/20/93	Demonstration of the control panel and management of motions the primary telescope mirror.
LO 9/20/93	Demonstration of the LO system through a workstation window.
message 10/ 4/93	Demonstration of the message system including display, levels of messages, supplementary information, and generation.
OSH prototype 10/26/93	Demonstration of a UNIX shell (BASH) as an observation control language with graphical user interface.
on-line access 12/ 2/93	Run-time access of collated data for analysis and display.
frontend 1/11/94	Demonstration of all coordination sequences, control, and monitoring of a prime focus frontend fully integrated with its LO system from both the engineer's and observer's consoles.
OSH I 1/25/94	Demonstration of an enhanced UNIX shell (BASH) as an observation control language with graphical user interface.
frontend 12-18 2/ 8/94	Demonstration of all coordination sequences, control, and monitoring of the 12-18 GHz frontend fully integrated with its LO system from both the engineer's and observer's consoles.
holography 4/ 6/94	Operation of the holography backend via its control panel and manager.
OSH II 6/ 3/94	Demonstration of the full observation control language with. graphical user interface.
monitor analysi 8/ 2/94	Use of PV_WAVE or KHOROS to access and analize monitor information.
alt user-inter 8/16/94	Demonstration of an alternate user-interface as ported from another system on the GBT.
log 9/28/94	Demonstration of logging of monitoring information including selection, searching, and display.