National Optical Astronomy Observatories

GNIRS

GEMINI NEAR INFRA-RED SPECTROGRAPH

GNIRS Software Plan

April 19, 2000

Richard Wolff, Software Manager	Date
Neil Gaughan, Project Manager	Date
Larry Daggert, ETS Manager	Date
Sidney Wolff, NOAO Director	Date
Mark Trueblood, Work Package Manager	Date

GNIRS Software Plan

Table of Contents

Software Plan for GNIRS

1. Introduction

The purpose of this document is to describe the software that will be used to control GNIRS both at NOAO and at the telescope. In the former situation, the full OCS interface is not available, and the requirements of laboratory testing will require more extensive scripting and logging facilities than at the telescope. In the latter situation, where the full Gemini system *is* available, control is primarily via the OCS with recourse to the engineering interface only during commissioning and maintenance.

The design of the software requires that the EPICS layer be quite thin, and that the whole instrument operate without EPICS for initial laboratory testing. While this approach will require some additional work (see the detector section below), it provides an extremely important separation of the low level code from the EPICS code, which both simplifies the testing of the software and the instrument, and also allows more flexible scheduling of programmer time.

2. Interface Control Documents

There will be one major ICD, 1.9.b/3.1, which details the interface of GNIRS to the OCS. Two other control documents are important. The first, specifying the instrument to OIWFS, has been written by Hubert Yamada for NIRI and should require little or no modification other than trivial ones such as substituting GNIRS for NIRI. The second, for the detector controller, should also be "identical" to that for NIRI. Other documents, such as "Software Requirements" or design notes will be produced as needed.

3. Infrastructure

This code module provides system initialization, allocation of structures such as ring buffers and semaphores, connects interrupt handlers to interrupt vectors, provides an error logging mechanism, and the like.

4. Motor Control

Software to operate the mechanisms within GNIRS forms the most "obvious" chunk block of code. At the lowest level, interrupt driven code is used to funnel commands to the VME controller cards, return status such as motion complete flags, current position, home switch position, and so forth. This code also flags illegal commands and limit overtravel.

Code at the next higher level takes commands from a motor control task and sends them to the motor controller. Semaphores for controlling access to the motor controller are used to keep the several motor tasks from interfering with each other. A command that requires a response from the controller gathers the response while blocking issuance of other commands to the same controller board until the response is received. The time delay is not a concern here as the response (which is always status information) is received in a time short compared to any physical motion. Should a command take too long (typically due to a motor failure), the resulting timeout will be seen at this level and an appropriate error indication will be propagated "upward".

Above that layer, there are functions that perform specific motor tasks, such as moving to a position specified as a number of counts from the zero location, finding the "home" position, probing limits (to test for correct functioning), etc. Commands at this level are suitable for many laboratory tests.

One additional layer is required, which converts units and positions as specified by the OCS into the numerical position values required by the layer below. This layer is also used to coordinate moves, if that is necessary. A configuration file mechanism is also implemented here. The configuration file provides a way to specify parameters that may be changed and which should not be buried in the code. Examples are the backoff distance used to remove backlash, the velocity of the motor as it moves from position to position, the names and locations of filters, and so forth.

5. Digital I/O Board

One digital input/output board is used in GNIRS for control and some auxiliary motor functions. For instance, the power to the Phytron motor drivers' card cage can be turned on and off by a single output bit. The position of the cryoheads "computer-manual" switch is sensed via two input bits. The back-up switches are sensed via this software until they are moved from auxiliary to primary duty. Functions provided by this software block will permit easy read-write access to the input-output bits on this board.

6. Temperature Board

A specially built NOAO board provides excitation and readback for temperature diodes. Code is needed to support this, and will be modeled on existing code provided by Nicholas Roddier and Janet Tvedt. The temperatures are provided to the status/health module; out of range situations are detected here and the appropriate notifications are made.

7. Status and Health; simulation

Status consists of the position of each motor, limit and home switch positions (on or off), whether the motor failed to move, the state of the I/O bits, the temperatures, etc. Health is "good", "warning" or "bad" and overall health of the instrument is a composite of the health of the subsystems. Code in this module provides all those values to the OCS and to the engineering displays, and determines the overall health.

Simulation is a distributed function. For the motor control, it is implemented primarily in the low-level routine that sends commands to the motor control board and waits for notification of completion. For fast simulation, the wait is not done, while for full simulation, the appropriate time interval passes before the function returns. Of course, in simulation mode, no errors occur.

8. Scripts and Logging

The extensive laboratory testing that will be done for GNIRS means that some software support for this functionality is required. Data, such as temperatures, or motor position, will be logged to a file and either displayed on the local workstation, or shipped to a PC for use in, for instance, an Excel spreadsheet. What data to log, and how often, will be readily changeable. In addition, it will be possible to write scripts to automate the testing process.

9. Detector Interface

For the initial system tests, the detector controller (the NOAO GNAAC controller) should be controllable from the same command interface that the rest of GNIRS is. However, the GNIRS components will be run from a system without any EPICS while the detector controller is an EPICS based system. Therefore, code in this module will use the channel access library to provide the message stream that the controller expects. Writing this module will take some time, and since it will ultimately be replaced with the full OCS compliant EPICS system, it might be considered unnecessary. However, the schedule does not allow for the full EPICS system to be ready in time for the initial testing that will involve the GNAAC controller, and proceeding in this manner does allow us to provide a complete test environment while the final EPICS system is being written and debugged.

10. EPICS

The design of the GNIRS software presumes a thin layer of EPICS. This means no SNL and minimal use of CAPFAST diagrams, an approach that has been successful with the CCD controller. However, an instrument is more complex, consisting of a detector controller, the OIWFS and the GNIRS instrument itself (the components controller), each of which uses EPICS as its top layer of software. Above all this sits the instrument sequencer, which communicates with the subsystems and with the OCS via EPICS. So it remains to be seen how thin the EPICS layer can still be and work correctly.

Another approach to the EPICS software may be to use the NIRI code. Since NIRI differs from GNIRS only in the components controller, in principle, it should be easy to convert the NIRI code to work with GNIRS. However, it is not clear that, in reality, this is true. This approach will be considered when it is time to begin the EPICS software task, which will also be after NIRI has been delivered and H. Yamada might be able to assist with the conversion.

The time estimates given below for the EPICS work assume that there is no dependence on the NIRI code.

11. TEST

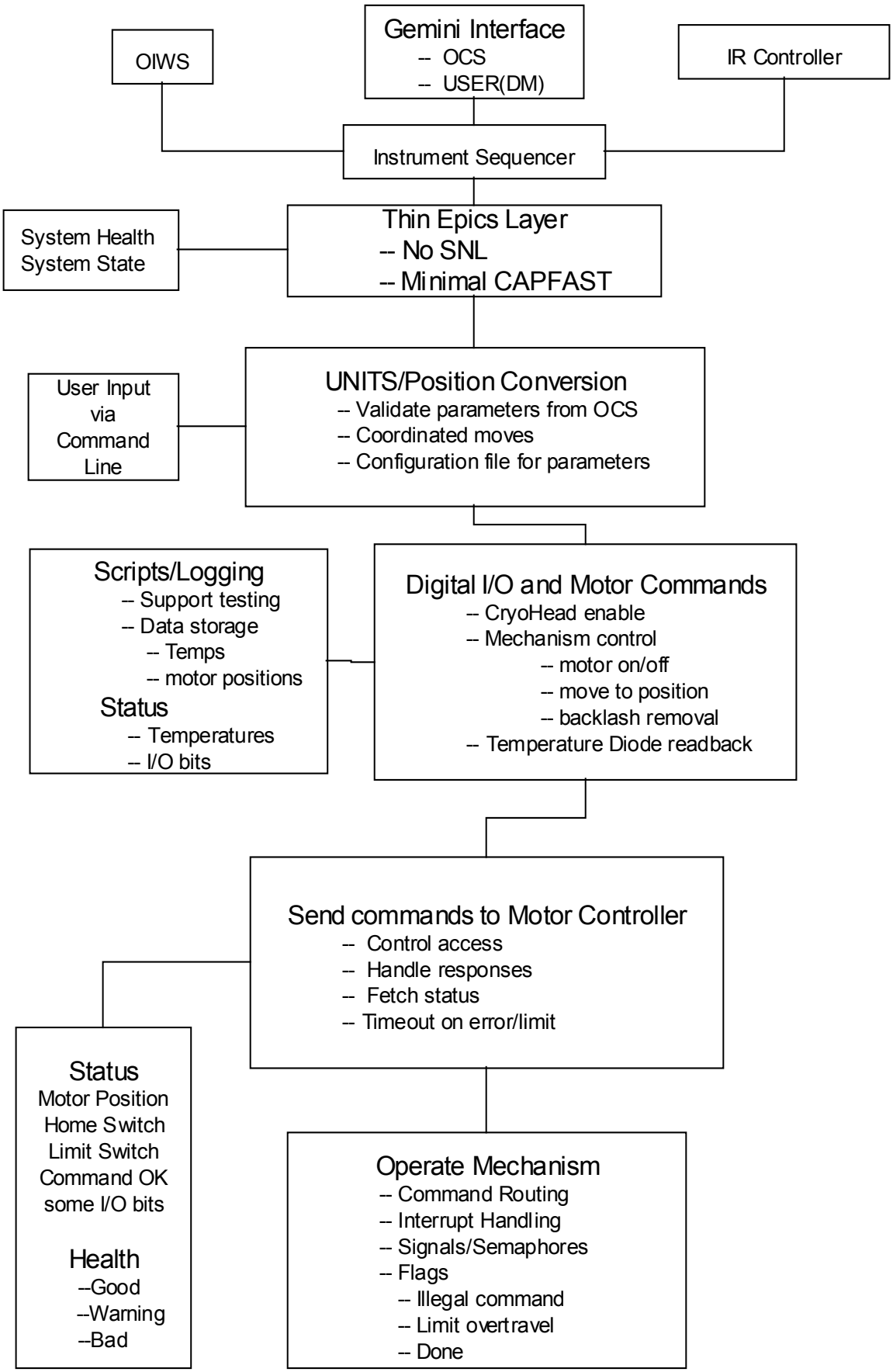
A specific test and debug phase for the software is included in the overall software plan. During this time, all the software that is not EPICS dependent will be verified using a test fixture designed for this purpose. The goal is to have, before integration of the final hardware begins, all the bugs removed from the software, or, more realistically, all the non-subtle bugs that inevitably appear when final software and final hardware first meet.

Software Tasks and Times, in approximate calendar order

Code Module	Time (weeks)
Infrastructure	2
Motor Control	
Low level	4
High level	6
Digital I/O	2
Temperature Board	2
Status/Health/Simulation	2
Scripts/Logging	
Script Language	3
Logging	1
GUI (if any)	3
Detector Interface	
Channel Access	5
Instrument Coordination	2
WCS (if needed)	1
EPICS	
Study, design	4
Capfast	8
Components Controller	4
Detector Controller	4
OIWFS	2
MEDM	4
SIR	3

12. BLOCK DIAGRAM

On the following page is a block diagram of the software. Reading down, it gives the command from the user/OCS to the hardware. Reading up, it describes the approximate order of the software development.


Gemini Interface
-- OCS
-- USER(DM)
OIWS
IR Controller
Instrument Sequencer
System Health
System State
Thin Epics Layer
-- No SNL
-- Minimal CAPFAST
User Input via Command Line
UNITS/Position Conversion
-- Validate parameters from OCS
-- Coordinated moves
-- Configuration file for parameters
Scripts/Logging
-- Support testing
-- Data storage
-- Temps
-- motor positions
Status
-- Temperatures
-- I/O bits
Digital I/O and Motor Commands
-- CryoHead enable
-- Mechanism control
-- motor on/off
-- move to position
-- backlash removal
-- Temperature Diode readback
Send commands to Motor Controller
-- Control access
-- Handle responses
-- Fetch status
-- Timeout on error/limit
Status
Motor Position
Home Switch
Limit Switch
Command OK
some I/O bits
Health
--Good
--Warning
--Bad
Operate Mechanism
-- Command Routing
-- Interrupt Handling
-- Signals/Semaphores
-- Flags
-- Illegal command
-- Limit overtravel
-- Done