MODS

Instrument Software Design

Document Number:	OSU-MODS-2004-002
Version:	1.4
Date:	2004 August 25
Prepared by:	R.W. Pogge, & J.A. Mason, OSU

T · H · E OHIO STATE UNIVERSITY

Distribution List		
Recipient	Institution/Company	Number of Copies
Richard Pogge	OSU	1 (file)
Darren DePoy	OSU	1
Jerry Mason	OSU	1

Document Change Record			
Version	Date	Changes	Remarks
0.1	2004-06-03		Initial Drafts & Updates
1.0	2004-06-18	Numerous and extensive	First Circulation Draft
1.1	2004-06-22	Comments by Jerry Mason	Second Circulation Draft
1.2	2004-07-26	Management additions	First Internal Release
1.3	2004-08-05	Minor revision based on comments from MODS team	First General Release
1.4	2004-08-25	Comments from DePoy	Revised Release

Contents

1 Introduction

1.1 Scope

This document describes the MODS data acquisition and instrument control software design. It covers system requirements, design goals and rationales, implementation plans and details.

1.2 Reference Documents

1. *ICIMACS Messaging Protocol Version 2 (IMPv2)*, OSU-MODS-2003-007, 2003 December 17
2. *Definition of the Flexible Image Transport System (FITS)*, NOST 100-2.0, 1999 March 29 [NASA/Science Office of Standards and Technology, Goddard Spaceflight Center]

1.3 List of Abbreviations and Acronyms

A&G	Acquisition and Guiding
AGW	Acquisition, Guiding, and Wavefront Sensing system
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CLI	Command-Line Interface
CSV	Comma-Separated Values (ASCII tabular data format)
CVS	Concurrent Version System
EAN	Europäische Artikel-Nummerierung (European product barcode standard)
FITS	Flexible Image Transport System (astronomical binary data format)
GCS	Guider Control System (LBTO interface)
GUI	Graphical User Interface
HTML	HyperText Markup Language
ICIMACS	Instrument Control & Image Acquisition System (OSU messaging protocol)
IIF	Instrument InterFace – LBT instrument interface API
IMPv2	ICIMACS Messaging Protocol version 2
IP	Internet Protocol (network layer of TCP/IP)
ISIS	Integrated Science Instrument System (OSU data system client/server system)
LBT	Large Binocular Telescope
LBTO	Large Binocular Telescope Observatory (operational arm)
LBTPO	Large Binocular Telescope Project Office (administrative arm)
MODS	Multi-Object Double Spectrograph
NOST	NASA/Science Office of Standards and Technology
OSU	The Ohio State University
POSIX	Portable Operating System Interface
RPC	Remote Procedure Call
TCP	Transmission Control Protocol (connection-oriented network protocol)
TCS	Telescope Control System
UDP	User Datagram Protocol (connectionless network protocol layered on IP)
WFS	Wavefront Sensor
XDR	eXternal Data Representation (e.g., as in RPC/XDR)
XML	eXtensible Markup Language

2 MODS Software System Overview

A functional block diagram of the MODS data-taking system is shown in Figure 2.1:

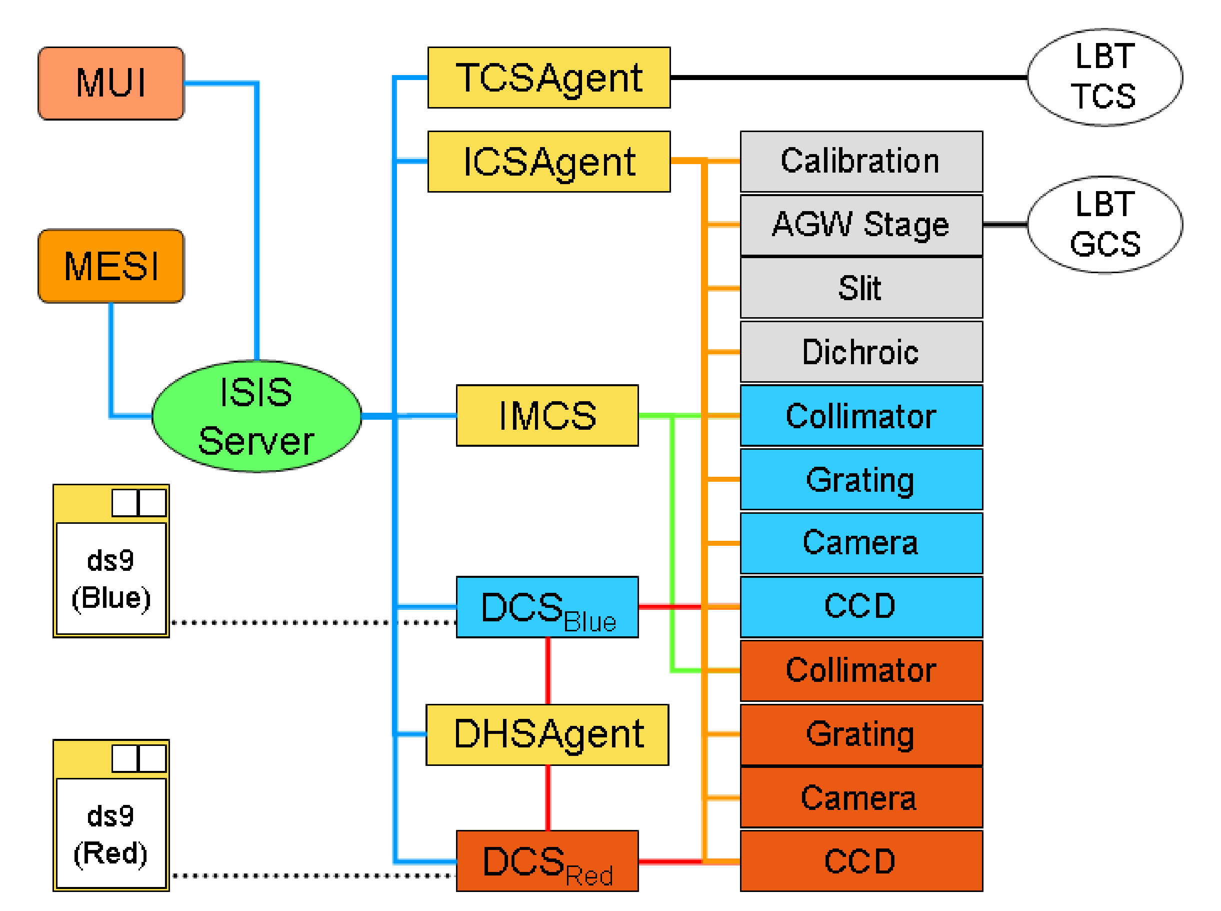

Figure 2.1: MODS Data Acquisition System functional block diagram, showing the major components. Different color lines code dedicated interprocess communications channels (network, serial, etc.). Ovals are servers (green is the ISIS instrument server, white are LBT Observatory servers), rectangles are clients, and orange lozenges (at left) are user interfaces. The components are described in the text.

The MODS data-acquisition system is an extension of our current Linux-based ISIS system, currently deployed with OSU-built instruments at CTIO, and soon to be deployed at MDM. It is an evolution of our earlier DOS-based ICIMACS system, retaining in particular certain key functional blocks (ported to the Linux environment and written in C instead of BASIC) and using the same, proven interprocess communications protocol. MODS, however, represents a considerable increase in complexity over all of our previous instruments, and will require features not previously deployed, as described in this document.

The basic architecture of the MODS system is a highly modular, distributed system in which the basic instrument functions are implemented as autonomous or semi-autonomous systems linked by a common interprocess communications protocol. This section describes the primary software systems and user applications that make up the MODS software package, and our coding and documentation standards.

2.1 Primary Software Systems

The MODS instrument software consists of the following primary systems:

System	Function
ISIS Server	Coordinate Detector, Instrument control, and data-handling functions, and maintains a centralized instrument status database.
Detector Control System	CCD detector configuration and readout.
Instrument Control System	Individual instrument mechanism control and overall instrument configuration.
Data Handling System	Controls storage and display of FITS format data acquired by MODS.
Flexure Compensation System	Operate the real-time flexure compensation system on both red and blue channels.
Mask Making System	Design and fabrication of MODS multi-object slit masks.
LBT TCS Interface	Interface between MODS and the LBT Telescope Control System (TCS).
LBT Guider (GCS) Interface	Provide a command/control interface for the LBT guide/acquire system (GCS) to operate the MODS pre-slit AGW stage.

Each of these systems will be described in §4.

2.2 User Interface Applications

The MODS system also provides the following user interface applications

Application	Function
MODS User Interface (MUI)	Primary observer interface for MODS, providing GUI and CLI options (scripting)
MODS Engineering Support Interface (MESI)	Engineering-level interface for instrument maintenance and support.
MODSTools	Observing preparation tool suite.

The MUI (pronounced “mooey”) is the principal way that most astronomers and LBTO support scientists will interact with the MODS instruments at the telescope. It will provide a full GUI as well as scripting capabilities, and may be operated remotely.

MESI (pronounced “messy”) provides support technicians at LBTO and OSU with direct access to all engineering-level functions of the MODS instrument for troubleshooting, maintenance and repair. It also provides remote connection capability to allow off-site personnel access to MODS engineering functions for remote troubleshooting.

MODSTools are a suite of standalone applications that would be used either at LBT or an observer’s home institution to prepare for observing with MODS. It includes functions for designing slit masks and for creating observing template files to facilitate both onsite and remote/queue-service observing.

2.3 Data Reduction Software

MODS data reduction software will use facilities of the NOAO IRAF package for basic 2D image reductions (bias, flat field, and zero/dark calibration), providing the necessary database files to instruct IRAF (e.g., imred) in how to handle MODS raw data.

Because the primary spectroscopic modes of MODS – long-slit and multi-slit – are mature techniques, we currently have no plans to create custom data-reduction software for MODS. We will, however, adapt current standard procedures (e.g., twodspec in NOAO IRAF and the GMOS software in the IRAF Gemini package) to MODS. Only if these prove to be inadequate in practice will we give priority to developing custom data reduction applications.

2.4 Coding Standards

All MODS software will be written and maintained following generally accepted Open Source/Open Standards practices. This includes

- Use of established standards for coding (ANSI C/C++, POSIX).
- Use of established standards for interprocess communications (TCP/IP and UDP/IP for networked communications via Ethernet, RS232 for serial communications).
- Maintaining all source code and documentation in the public domain (e.g., using a Gnu General Public License: www.gnu.org/copyleft/gpl.html).
- Use common public domain packages & utilities where possible to ensure portability and transparency under change of personnel. Where possible/practical, we will adopt the same standard packages as other members of the LBT consortium.

Public-domain packages that will be adopted for MODS include the following:

- Gnu gcc/g++ (gcc.gnu.org) for compiling C and C++ code.
- Gnu readline and history libraries for CLI shells. This provides an interface familiar to many Unix users (command-line editing and history mechanism like in the tcsh shell – see cnswww.cns.cwru.edu/php/chet/readline/rltop.html for details).
- X11/Qt package (www.trolltech.com) for GUI development. This is the GUI package adopted by the LBTO. We will use the public-domain version of Qt, rather than the commercial version, for better portability.
- POSIX pthreads library for multithreading/concurrency. This is a standard API for multithreading (IEEE Posix 1.3c), and is the standard way to multithread in the Unix/Linux environment (see www.humanfactor.com/pthreads for an overview).
- cfitsio for FITS support (heasarc.gsfc.nasa.gov/docs/software/fitsio/fitsio.html). FITS is supported by a NASA Office of Standards and Technology (NOST) standard, and has been formally adopted by the IAU for all data exchange for astronomical observations.

- SAOImage DS9 for image display (hea-www.harvard.edu/RD/ds9/). This is the most commonly used image display program in astronomy, and is well-known to all observers.

Additional standards and practices that we will adopt for MODS software include:

- Linux as the operating system for workstations running MODS applications.
- Unix make and Makefiles for package compilation.
- CVS (Concurrent Version System: www.cvshome.org) for code revision control and development distribution. Local (OSU) CVS services will be linked to the LBTPO CVS.
- Unix tcsh (www.tcsh.org) for interactive command shells in software maintenance accounts.
- X Window System (www.x.org) for the Unix/Linux windowing environment, using only applications compatible with standard xlib libraries.
- HTML v4.0 (www.w3.org/TR/html4/) with the ISO-8859-1 character set for online hypertexted documentation.
- Adobe PDF (www.adobe.com) for electronic document exchange via email or web.
- The Code 39 and Code 128/EAN barcode symbologies for ASCII-based barcode labels. Software must also be able to decode either transparently.
- ISO-8601 for encoding date/time information in FITS headers, for display by software modules, and in all software documentation. This follows standards established by the IAU FITS Working Group (fits.gsfc.nasa.gov/iaufwg/iaufwg.html).

2.5 Software Documentation

All MODS software will be documented, from internal documentation embedded in the source code to user manuals. Code-level software documentation will be created using the Doxygen documentation system (www.doxygen.org) adopted by the LBTO. Documentation will be generated as both hyperlinked HTML pages maintained at the OSU MODS website (with mirror copies at the LBTO software web site), and as PDF format documents. MODS has adopted the Qt-style of Doxygen markup for all source code.

User Documentation provided with MODS will include the following products:

- System Engineering Manuals
- Observer's Manual, including observing preparation, mask making, and observing manuals.
- Observer's quick-reference guides for use at the telescope.

Online documents available in HTML format from the MODS website at OSU (with mirrors at LBTO) will include full indexing, search engines, and formatted PDF versions for printing as a single document. All online user documentation will conform to accepted US federal standards for electronic content accessibility for persons with disabilities (Section 508 of the Rehabilitation Act: www.section508.gov).

3 MODS Software System Functional Requirements

This section describes the functional requirements of the various MODS software systems and interfaces.

3.1 Detector Control System (DCS)

The MODS Detector Control System software will provide the following functions:

- Control CCD detector preparation (erase/reset) and readout.
- Software-selectable on-chip binning and region-of-interest readout modes.
- Non-readout charge-shuffling mode in the slit-direction to allow straightforward implementation of "Nod-and-Shuffle" observing modes.
- Precision exposure timing and shutter control. The MODS shutter opening time is 300msec, timing precision must be at least 1% of this time, or 3msec.
- Create images in FITS format with full FITS header information. Archiving and logging will be done by the Data Handling System (§3.4). Images must conform to the formal NOST FITS standard, and comply with LBT FITS header specs.
- Display of raw images using SAOimage ds9.

3.2 Instrument Control System (ICS)

The MODS Instrument Control System software will provide the following functions:

- Provide a common interface to all MODS mechanisms and non-mechanical subsystems (e.g., calibration lamps), excluding the detectors (§3.1), for both observer- and engineering-level operation.
- Control of individual stepper motors for all of the MODS mechanisms.
- Control of MODS auxiliary functions like the calibration system lamps, barcode scanners, etc.
- Allow MODS mechanisms to be configured concurrently (i.e., in parallel rather than serially), within the limits imposed by safety and mechanical path conflicts. This greatly reduces the time required to (re)configure the instrument.
- Provide software interlocks for conflicting functions (e.g., don't allow calibration lamps to be lit when the instrument dark slide is open, don't permit users to move the slit mask storage cassette when the slit mask changer is in operation and vis-versa, etc.). All interlocks must be provided with executive overrides in the engineering layers, insofar as there are no safety issues (danger to personnel or hardware).
- Provide software locks for mechanisms that can be controlled by multiple processes. For example, lock out the collimator to user control when the IMCS is active, lock out the AGW Stage to MODS user configuration when the LBT GCS has control, etc. Also provide all software locks with appropriate executive overrides in the engineering layers, to prevent a dead/zombie process from holding a stale lock.

- Provide a status query mechanism to report the current state of all instrument functions, either individually or as a whole.
- Provide adequate runtime logging of all instrument mechanism operations to permit troubleshooting of problems after the fact, including time/date tagging and full transcripts of command requests and responses running through the system.

3.3 Image Motion (aka "Flexure") Compensation System (IMCS)

Compensates for gravity-induced flexures, structural "ticks" and thermal motions in the instrument using a closed-loop IR laser metrology system sharing the science light path. Note that "image motion" in this context is not associated with active or adaptive optics correction of the science light.

- Integrate all MODS functions required to operate the collimator-mirror steering image-motion compensation system into a single software module.
- Provide a simple user interface for setting up and using the IMCS.
- Provide a socket-based command interface compatible with the message-passing system used by all other instrument subsystems (§4.1).
- Control the IR metrology laser and readout the IR quadcells remotely via the IMCS controller application.
- Provide runtime logging and performance tracking functions to help evaluate IMCS operation.

3.4 Data Handling System (DHS)

This system mediates all data storage, access, and archiving for science and images and engineering data for MODS, and provides the point of interface to any observatory archive.

- Copy raw images from the DCS hosts and archive on a centralized data disk, most likely using RAID arrays for data security. Users access MODS data via the DHS host computer.
- Logging of all acquired data in a form easily read and annotated by the observer, including provisions to export the logs in a variety of portable formats (ASCII, CSV, and TeX/LaTeX) for observers to take home with them.
- Provide facilities for safe backup of data, and copying of observer data to removable media (removable drives, tapes, memory sticks, etc.) for them to take home.
- Provide the main interface between MODS and any LBTO data archiving system.

3.5 LBT TCS Interface

- Provide an interface to the LBT Telescope Control System using the LBTO-provided instrument interface (IIF) library.
- Translate LBT TCS data structures with passive status info (pointing status, etc). into IMPv2 format message strings for the DCS and other MODS data-acquisition system modules.

- Provide the MODS user interface (MUI) with basic remote TCS functions for commanding telescope relative position offsets, telescope focus adjustments, autoguider engage/disengage, etc., insofar as those functions are supported by the LBTO IIF.

3.6 LBT Guider (GCS) Interface

- Provide an interface between the MODS pre-slit AGW stage (controlled by the Instrument Control System, §3.2) and the LBT Guider Control System. Functions provided will allow the LBT GCS to move the AGW stage in X and Y directions, focus the A&G and WFS cameras, and insert A&G camera filters.
- Provide an API to LBTO to assist in creating the interface for the GCS. MODS team programmers will collaborate with the LBT GCS Team at Heidelberg to create and maintain the MODS AGW Stage API.

3.7 Mask Design and Manufacturing System

- Provide portable host software packages to distribute to partners for slit mask design in advance of an observing run. This software will be available as binaries for Linux workstations (which all partners are assumed to have), or as source code that may be compiled on their systems.
- Provide output in the form of commands for the laser cutting machine that converts mask target coordinates into machine instructions for cutting the mask.
- Provide software for mask inventory tracking in transport from the laser machine to storage to deployment in MODS and back to storage, including a web interface into the inventory database.

Mask software will be created in collaboration with LUCIFER group which will share the same mask cutting machine. The idea is to create a common interface used for MODS and LUCIFER by users to create multislit masks, and a common mask tracking and inventory system to be used by both. Common systems are best for observers and LBTO support personnel alike.

3.8 MODS User Interface (MUI)

- Provide the main observing interface for the MODS instrument in the form of a graphical user interface built on the general LBT model. The look-and-feel should be similar to that of LBT GUIs, following LBT guidelines (LBT CAN 481s010a).
- Provide utilities for configuring the instrument, setting up and acquiring exposures, and monitoring the progress of an exposure.
- Provide a facility for executing observing "templates" that configure the instrument and acquire data using pre-prepared settings.
- Provide a facility for remote observing with MODS.
- Provide a basic scripting capability for setting up and executing complex observations, including telescope interaction (e.g., nod-and-shuffle modes, offsets and mosaics, etc.)

3.9 MODS Engineering Support Interface (MESI)

- Provide engineering-level tools to support troubleshooting and routine maintenance of the instrument.
- Provide multi-layered access to instrument functions, from high-level (observer interface-like) meta-commands, to low-level mechanism commands (direct interaction with stepper-motor drives and sensors), and access to interprocess telemetry and communications traffic layers.
- Provide tools for exercising instrument components, with quiet, verbose, and super-verbose "debugging" output modes as required.

3.10 MODS Observing Preparation Tools (MODSTools)

- Provide a suite of portable interactive graphical observing planning tools to be used by observers in advance of their run.
- Provide support for Remote/Queue-Service modes, including tools for organizing obs files into more complex “observing sequences” and preparation and testing of MODS scripts for uploading for execution by queue/service observers.

4 MODS Software Systems

4.1 Software Architecture Overview

The MODS instrument control and data-acquisition system uses a highly modular, distributed programming model in which the basic instrument functions are implemented as autonomous or semi-autonomous systems linked by a common interprocess communications protocol.

MODS software modules are designed to reflect the modularity of the MODS hardware, and will be developed in parallel with the assembly and testing of each major MODS instrument mechanism, and then replicated for all members of each mechanism family. For example, the camera control software will be implemented and tested in parallel with the assembly and testing of the first MODS camera: since all 4 MODS cameras are functionally identical, subsequent cameras will inherit the camera control module.

Individual MODS modules will be implemented as independent programs running on a single Unix host, or running on an independent machine, the latter especially if that module has significant real-time requirements that are satisfied by a machine with a custom kernel or runtime configuration.

All MODS modules are combined into an integrated system using a distributed client/server model. Our client/server is system is the ISIS package developed at OSU. ISIS stands for Integrated Science Instrument System. It consists of a central message-passing server that performs basic coordination and runtime logging tasks, with instrument functions implemented as distributed ISIS client applications, generically called “agents”. Clients communicate with the server and each other using the IMPv2 messaging protocol.

MODS user interfaces are designed to run on any Unix host, primarily on the observer and engineering “console” workstations, requiring no special configuration of those machines to operate. All modules requiring special hardware or operating-system software requirements will run on workstations provided and configured by the MODS project, in cooperation with the LBTO to ensure ease of maintenance by LBTO personnel. Standards for such machines will be negotiated in advance to ensure compatibility with the LBTO mountain network, and so as to not introduce a maintenance burden on LBTO personnel.

High-level user interfaces provide both graphical and command-line interfaces, and convert generic user commands (or GUI widget actions) into the necessary IMPv2 messages directed to the ISIS agents responsible for those functions. The interfaces are thus "skins" overlaid on the ISIS system, and can evolve and adapt nimbly without having to modify the low-lying client/server system.

4.1.1 The ISIS Client/Server System

Our current generation of OSU data-taking systems has implemented IMPv2 using a client/server system known as ISIS. ISIS consists of an IMPv2 message passing server (the *isis* server application), and a number of ISIS client applications that perform the instrument and data-handling functions.

The ISIS server application is an interactive message-passing server & interface multiplexer that can handle multiple network socket (UDP) and serial (RS-232) communications ports. It also provides an interactive command-line interface for engineering-level operations. Its basic services are

- Pass IMPv2 messages between ISIS client applications.
- Process and distribute broadcast messages for all clients.
- Provide runtime logging of all system telemetry.
- Handle "executive" server commands to control server operation.

ISIS client applications are generically termed "agents". A typical ISIS client agent has an interactive command-line interface for engineering purposes in addition to the ISIS socket interface which handles the interprocess traffic from the server and other ISIS clients. Runtime configuration is established by loading and parsing by human-readable and human-writable runtime configuration files. This allows for rapid custom configuration of an agent without necessitating recompilation of the code, making them flexible and adaptable.

An ISIS client agent may run as a semi-autonomous client of an ISIS system, or as a standalone user application independent of an ISIS server. In this latter mode, agents serve as simple interactive engineering command shells for system maintenance and development. Indeed, most instrument subsystems begin development as standalone ISIS client applications. Even in standalone mode an ISIS client has an active ISIS client socket interface available to serve as a "backdoor" command conduit. We often use this feature in the lab to write Perl scripts to run automated tests through a standalone client (e.g., we ran a 12-position filter wheel through 10,000 randomly selected filter positions as part of a life and durability testing exercise during lab acceptance work prior to shipping the instrument to CTIO). We expect to make extensive use of this feature in the integration and testing of MODS, and will provide a number of standard utility scripts for on-site testing. We've demonstrated that these Perl scripts work the same on Linux, Windows, and Mac computers.

All ISIS clients use a common ISISclient API that is stable and well-documented (see www.astronomy.ohio-state.edu/~pogge/Software/ISIS/ for the current version). All MODS system clients will be implemented as ISIS agent applications.

Versions of our ISIS system have been deployed in the OSU instrument lab, and in the field with OSU-built instruments at CTIO (the ANDICAM at the 1.3m telescope and Y4KCam at the 1.0m telescope), SOAR 4.2m (OSIRIS), and MDM (the MDM4K camera filter wheel). Components of the ISIS package have been used in an active observing environment for nearly 18 months with no problems. The ANDICAM on the CTIO 1.3m is operated as part of the SMARTS consortium. It is a dual-channel IR and CCD camera that has regularly acquired 2-4Gb of data per night on every clear night since it went into operation in February 2003. Engineering data collected has helped to improve the ISIS server and client systems, and to validate our basic message passing and interprocess communications infrastructure. When deployed with MODS in early 2006, ISIS will be a mature, field-tested technology.

4.1.2 Interprocess Communications

All interprocess command and status communications use standard networking and serial interface standards, as required by the hardware/software configuration. No proprietary standards or hardware will be used. All network communications will use standard Ethernet-based IP protocols, compatible with the LBTO mountain network.

Network communications between systems will use a standard client/server architecture implemented on connectionless User Datagram Protocol (UDP) sockets layered on standard IP transports. The typical bandwidths used for command messaging traffic are sufficiently small (largest messages used by our IMPv2 messaging protocol are 2kB in size, most are <100 bytes) that the traditional disadvantages of connectionless UDP are not an issue. We will use standard Unix *socket* utilities to implement UDP communication APIs, ensuring maximum portability between POSIX-compliant Unix environments.

Connection-oriented TCP/IP sockets are used when required by specific equipment (e.g., the network serial port servers we have adopted use TCP/IP), or when application demands on resources require it. An example of the latter includes systems that are responsible for transferring data from the CCD detector control systems to the data-handling system. In software we will use the standard Unix *socket* libraries to implement TCP/IP communication APIs to ensure portability and POSIX-compliance.

Serial communications will use RS232 (EIA232) standard serial communications protocols. RS232 is a common industrial serial interface standard for stepper-motor controllers and barcode readers. Our default serial configuration will be 9600 baud, 8 data bits, 1 stop bit, no parity, with no flow control and no software or hardware handshaking. Exact configuration details depend on the particulars of the serial appliance in use, but most commercial units start close to this. In software we use standard Unix *fcntl* and *termios* utilities to implement serial port communications APIs.

4.1.3 Interprocess Messaging Protocol

MODS will use the ICIMACS Messaging Protocol version 2 (IMPv2) developed at OSU for our data-taking systems and in use since 1995. IMPv2 is a simple, lightweight, text-based, messaging protocol for carrying interprocess communications between the various programs that make up the data-taking system. Messages are formatted so as to be both human-readable and readily machine parsed. IMPv2 makes no specification of the programming language or operating system to be used to implement it, nor does it specify the communications medium to be used to send and receive messages between processes. This makes it usable by and between any systems that use ASCII characters to represent text, and it is extremely flexible with regards to choice of the interprocess communications transport method (so far we have used it with RS232 serial ports, TCP/IP sockets, UDP/IP sockets, Unix FIFO pipes, Unix Domain Sockets, Remote Procedure Calls, and IR-based serial communications). The details of IMPv2 are described in a separate document (OSU-MODS-2003-007), available from the MODS website at OSU (www.astronomy.ohio-state.edu/MODS/Software/).

We explored other messaging protocols, particularly RPC/XDR and various XML-based systems. RPC has problems of being fairly system dependent, and is supported by a relatively limited number of platforms and languages. It also did not meet our needs for various types of communications. XML is very extensible and portable, but the syntax is very heavy weight for our application: typical small XML messages requires 3-10× more characters to transmit than simple IMPv2 messages to perform the same interprocess messaging task, and they are much less human readable. We have therefore elected to adopt and extend our own messaging protocol because we have a lot of experience with it (10 years and nearly a dozen instrument deployments on 5 continents), and because it satisfies all of our requirements. The human-readable virtue of IMPv2 makes it an excellent and simple basis for building a "command language" for engineering applications. A human operator can easily type a raw IMPv2 message string at a keyboard, and read it off the screen. This "direct access" to the lowest communication layers has proven to be of great value during development and support: we don't need a separate piece of software to read raw messaging traffic (as would be the case with an XML-based system).

4.1.4 Multithreading

A major requirement of the instrument control system is that most mechanisms must be concurrently configurable (§3.2), insofar as those motions do not physically interfere with each other. The standard way to implement concurrency is via multithreading. The standard API for multithreading is the pthreads library, which implements the IEEE POSIX standard interface for multithreading.

Multithreading, however, must be carefully applied to avoid race conditions, and to avoid imposing unnecessary execution overheads associated with launching a thread. Multithreading also complicates the programming in subtle ways (debugging, for example, can be greatly complicated). In general, a goal of the design phase will be to identify those components that will benefit most from concurrency via multithreading, and those which actions are best executed serially.

Another way to implement concurrency is to vest control in independent distributed processes, rather than trying to have one grand program that multithreads all concurrent operations. For example, the focal-plane section, red channel, and blue channel are natural candidates for control by three independent client agents. In general, it makes sense for processes that must share resources to be run under one master thread, whereas processes that do not require resource sharing could in principle be run independently and achieve effective concurrency without the problems associated with explicit multithreading, since Unix is, after all, a multitasking environment. During the design and prototyping phases we will identify how fine-grained we need to be in decomposing the system into independent agents.

4.2 Instrument Control System (ICS)

The Instrument Control System (ICS) controls the configuration of the MODS mechanical systems. It consists of low-level functions for direct control of stepper-motor drives and sensors, coupled with high-level programs for integrated mechanism control. This section describes the basic ICS architecture, the drive topologies, and describes each of the major components and their requirements.

4.2.1 ICS Architecture

The ICS software modules mirror the modularity of the main instrument components, and allow the same software agents to work with the full instrument or with a module extracted from the instrument and mounted in its handling cart for maintenance or upgrade. The 7 main functional modules of the ICS are as follows, listed in order from the top of the instrument, following the flow of photons through MODS:

ICS Module	Functions
Calibration System	Instrument Dark Slide (open/close) Calibration Tower (insert/retract) Calibration Lamps (select/on/off)
AGW Stage	AGW probe mirror X,Y position AGW camera focus (in/out) A&G camera filter (4-position)
Slit Mask System	Slit Mask Cassette (select) Mask Insert/Retract Slit mask barcode reader
Dichroic Beam Selector	Dichroic Drum (3-position)
Collimator System (Red & Blue)	Collimator Focus Collimator Tip/Tilt (for IMCS)
Grating Turret System (Red & Blue)	Grating Select (4-position) Grating Tilt (3 gratings/turret)
Camera System (Red & Blue)	Filter Select (8-position) Shutter (open/close) Camera Focus (in/out)

Each of the MODS ICS modules are controlled and coordinated by a central MODS ICS Agent. This is the primary interface into the instrument mechanism modules. Applications would query the ICSAgent application to determine the current state of all MODS mechanisms (e.g., such a query would be sent by the detector control system to fill in the instrument configuration keywords in the image FITS headers). A schematic of the ICS is shown in Figure 4.1.

The demands of concurrency require that we multithread some configuration operations. For example, the collimator mirror system must be able to move all three mirror support actuators concurrently for tip/tilt/focus operations, since serial configuration would be both timeconsuming (theoretical concurrency gain is a factor of 3), and seriously misalign the system during serial moves. We note minimum concurrency requirements for each system below.

The block diagram in Figure 4.1 shows functional modules, not independent programs. The degree to which these functional modules will be encapsulated into single programs remains to be determined. As a first cut, a logical division would be to divide the instrument into 3 zones: focal plane, red channel, and blue channel, and write client applications that control each functional block. Modules that require real-time interaction with other programs, like the AGW stage and the Collimator systems are also logical candidates for implementation as independent client applications. A goal of the early design phases is to identify the precise division of functions.

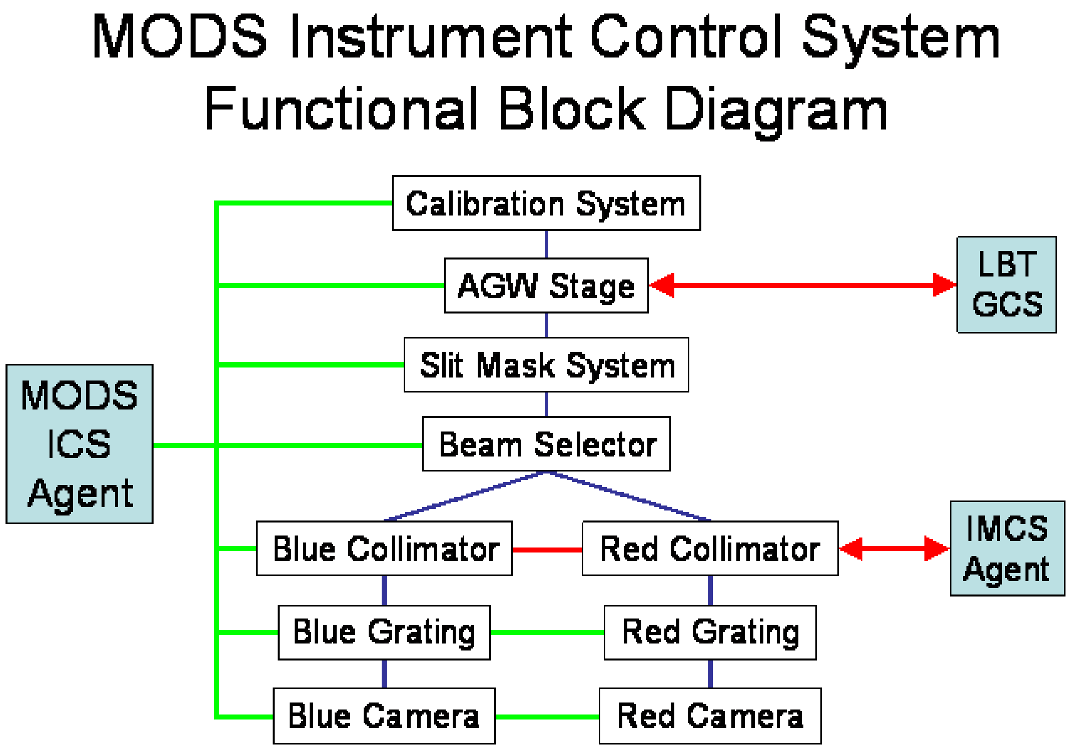

Figure 4.1 – Functional block diagram of the MODS ICS modules. The master ICSAgent is shown at left, and the two outboard applications that must interact with specific MODS functions are shown at right: the LBT GCS that interacts with the AGW Stage (§4.2.4), and the IMCS Agent application that controls the MODS Image Motion Compensation System and steers the Collimators (§4.2.7)

4.2.2 Mechanism Topologies

All MODS mechanisms are instances of one of three basic mechanism "topologies":

1. **Continuous Linear Drives**: continuous motion between two fixed limits and with an encoded zero (home) position. The location of the mechanism along this drive is determined by step counting from the home position (rather than by using an absolute position encoder). Examples of continuous linear mechanisms in MODS are the X-Y stage in the AGW Stage, the grating tilt mechanisms (3 per channel), the camera focus mechanisms (2 per channel), and the collimator tip/tilt/focus actuators (3 per channel).

2. **Indexed Linear Drives**: motion between detented "index" positions constrained between two fixed limits. Each indexed position is kinematically docked to ensure stability. Position sensors encode which index the mechanism is in, plus an in-position sensor that is asserted high when docked, and asserted low when out-of-position between indexed locations. Examples of indexed linear mechanisms are the slit mask, calibration tower, and grating select mechanisms (the latter may seem counterintuitive, but because of the cable wrap for the grating-tilt drives, the rotation of the grating select turret is constrained against rotating freely, and therefore the drive topology is the same as a linear indexed drive between 2 limits – i.e., to get from position 1 to position 4, you must drive through 2 and 3, not direct to 4 as would be the case for a true rotary drive mechanism).
3. **Indexed Rotary Drives**: mechanisms free to rotate without constraint in either direction between fixed, detented "index" positions (e.g., to get from position 1 to position 8 in an 8-position filter wheel, you drive 1 step from 1 to 8 – the instrument drives compute the least path clockwise/counterclockwise to reach the target position). Each indexed position is kinematically docked, and encoded using position sensors. An in-position sensor is used to tell when the mechanism is docked or in between valid indexed positions. Examples of indexed rotary drives are the camera and A&G filter wheels.

A fourth topology would be a Continuous Rotary Drive, of which there are none in the present MODS system.

A stepper-motor drive API is being created that implements all of these basic drive configurations. This API greatly simplifies the development of the mechanism controls, as systems with a common topology will inherit functions common to all mechanisms of that type, and the only details that must be worked out to implement a new mechanism is to determine the best set of drive parameters.

4.2.3 Calibration System

This system has three components:

1. Instrument Dark Slide (open/close), using a continuous linear drive
2. Calibration Tower (insert/retract), using a continuous linear drive
3. Calibration Lamps (select/on/off).

A unique feature of the Calibration system is the lamp interface to turn on/off the continuum and spectral-line calibration lamps mounted in the off-axis integrating sphere. This system requires software for a remotely-operated high-voltage switching power supply for the Pen-Ray spectral-line lamps, and a constant-current AC power supply for the Quartz-Halide continuum lamps.

<u>Interlocks</u>: A software interlock will prevent the calibration lamps from being turned on if the calibration tower is not in the beam (instrument looking at the sky instead of the integrating sphere through the calibration tower optics), or if the instrument dark slide is open. This will prevent light from leaking out of MODS into the LBT environment.

<u>Concurrency</u>: The minimum concurrency requirements of the calibration system are to be able to insert/retract the calibration tower while closing/opening the instrument dark slide. For

lamps, multiple lamps may be lit, though this may be done serially as the execution times are expected to be short (time to command a switch to open/close).

4.2.4 Acquisition/Guide/Wavefront (AGW) Camera Stage

The front-side AGW stage carries the A&G and WFS cameras, but does not operate them, that function is provided by the LBT GCS and associated camera controller hardware provided by LBTO. The front-side AGW Stage provides the following functions:

1. X,Y positioning of the camera pickoff mirror, consisting of 2 continuous linear drives.
2. AGW Camera Focus, consisting of a single continuous linear drive.
3. A&G Camera Filter wheel, a 4-position indexed rotary drive.

The AGW stage control will be implemented as the AGWAgent application, since this module must provide both an internal interface to the rest of the MODS ICS, and an external UDP socket interface to the LBT GCS. This remote socket allows control of the AGW stage by the telescope operators GCS console, and integration with normal LBT off-axis guiding and off-axis wavefront sensing functions.

The AGWAgent will be developed in collaboration with the LBT GCS programming team in Heidelberg. The MODS Team will provide the GCS team with a fully documented and tested API to enable the GCS, when operating MODS, to send and receive IMPv2 compliant commands to the AGWAgent. Responsibility for creation, maintenance, and documentation of the AGWAgent API will reside with the MODS team. A detailed AGW Stage use case document has been generated and distributed to the MPIA team as of 2004 July.

Interlocks: The GCS can assert a "lock" on the AGWAgent which will prevent the MODS ICS from moving any of the AGW stage's systems while the GCS has lock. When the instrument is idle (as signaled through the IIF?), the GCS releases the lock, allowing the MODS ICS to control the AGW stage configuration. This function is essential to allow the observer to stow the AGW probe, e.g., to remove it from obstructing the beam during unguided acquisition or calibration observations.

Concurrency: The minimum concurrency requirements of the AGW Stage are to be able to command stage motion simultaneously in X, Y, and focus.

4.2.5 Slit Mask System

The slit mask system has three components:

1. Slit Mask Cassette (mask select and off-axis storage), a 25-position linear indexed drive.
2. Slit Mask Changer (store/deploy a slit mask), a 2-position linear indexed drive (masks kinematically dock in the science field, and when stowed in the mask cassette).
3. At least 1 and possibly 2 barcode scanners to read the barcodes affixed to the slit masks. The primary scanner reads the label on the mask in the science aperture to provide a positive identification of the "active" mask. A secondary reader would be located on the mask cassette to scan stored masks.

The slit mask system module must provide a "create mask table" function that will read through the barcodes on all of the slit masks loaded into the cassette and build a mask table.

Interlocks: A hardware interlock is required to prevent the slit mask cassette select motion from interfering with the slit mask changer's insert/retract motion. Software interlocks will add an additional layer of protection to prevent simultaneous or interfering operations of these devices. Barcode scanners must be able to be turned off except when a barcode is being scanned. When MODS is in "acquisition" mode, the barcode scanners will be locked against reading, to prevent stray laser/LED light from interfering with science or calibration exposures.

Concurrency: **None**. The slit mask select and slit mask changer are coupled motions that must move serially. The slit mask system has no internal concurrency requirements (though it should be able to be configured concurrently as a unit with other modules).

4.2.6 Dichroic Beam Selector

The dichroic beam selector has a single component:

1. Dichroic Drum, a 3-position indexed rotary drive carrying the dichroic beam splitter, a red folding flat, and an open position to select between the dual-beam, red-only, and blue-only beams, respectively.

This simple mechanism is functionally identical to a 3-position filter wheel. It has a fixed population (none of the elements are removable in the same sense as filters), so a fixed population table can be provided to the MUI so that beam selection is transparent and does not requires users to know how position numbers correspond to beam select positions.

Interlocks: A software interlock prevents the beam selector from begin changed when the IMCS is in active. The IMCS lock is asserted through the ICSAgent.

Concurrency: None (single mechanism), but it may be configured concurrently with other modules.

4.2.7 Collimator System

The collimator system consists of three identical mechanisms:

1. Three (3) tip/tilt/focus actuators, continuous linear-drive piston actuators.

The collimator control system provides collimator mirror focus (piston all 3 actuators in synch in/out), and to steer the beam (actuators pushed/pulled to tip/tilt the mirror). The collimator must interact with both the ICS and the IMCS.

Interlocks: The IMCS can assert a software "lock" on the Collimator that prevents the MODS ICS from moving any of the collimator tip/tilt/focus actuators while the IMCS is engaged. When the IMCS is idle, it releases the lock, allowing the MODS ICS to control the Collimator Mirror configuration.

Concurrency: The minimum degree of concurrency required is be able to move all 3 collimator actuators together for set, tip/tilt, and focus motions.

4.2.8 Grating Turret System

The grating turret system has four components in 2 groups:

1. Grating Select, a 4-position rotary mechanism
2. Grating tilt mechanisms for 3 gratings, one continuous linear drive per grating.

The three grating tilt drives have the most stringent mechanical tolerances of all of the MODS mechanisms, and are consequently the slowest mechanisms in MODS. While the grating turret for selecting among the 3 gratings or an imaging flat mirror is a rotary indexed mechanism, the cable wrap for the grating tilt drives restricts the motion, so the basic topology is equivalent to a linear indexed drive (i.e., to get from position 4 to 1, you must drive through 3 and 2, rather than via the least path as in, for example, a free rotary filter wheel).

<u>Interlocks</u>: A software interlock will prevent the grating select and grating tilt motions from occurring when the IMCS is active. The IMCS lock is asserted through the ICSAgent.

<u>Concurrency</u>: The minimum degree of concurrency required is to be able to rotate the selected grating into the beam and set its tilt simultaneously. In principle, it is possible to change the three grating tilts simultaneously, but in practice this will rarely be necessary or indicated in normal observing or engineering situations.

4.2.9 Camera System

The camera system has four components:

1. Camera Shutter, a 2-position (open/close) linear indexed drive.
2. Camera Filter Wheel, an 8-position indexed rotary drive.
3. Camera Primary Focus, a continuous linear drive.

A unique feature of the Camera System is that the camera shutter will be triggered by a DC voltage signal from the Detector Control System, rather than under the control of the ICS. To manually open the shutter, the commands will be sent to the respective DCS rather than the ICS. This is the only separately-operated mechanism in MODS.

<u>Interlocks</u>: A software interlock will prevent camera focus or filter select from occurring when the IMCS is active. The IMCS lock is asserted through the ICSAgent.

<u>Concurrency</u>: The minimum concurrency is to be able to select a filter and set the camera primary focus at the same time. The shutter is independently operated. A concern to be addressed during lab testing phases is whether camera filter or focus motions during CCD readout introduce noise into the images. If so, then camera configuration will be forbidden during readout.

4.3 Detector Control System

The Detector Control System (DCS) components operate the CCD detectors. Its basic function is to acquire CCD data and deliver a FITS-format image file to the Data Handling System for subsequent storage and display.

Because detector readout and control has serious real-time hardware requirements, each CCD detector system has its own, dedicated workstation running a specialized kernel and software. Two detector control systems are required per MODS, one for each of the Red and Blue channels. The DCS will be implemented as an ISIS client application with an engineering interface and laboratory standalone modes. Specific functions include:

1. Detector readout control: on-chip binning, region-of-interest readout, overscan columns and rows, focus-plate charge shuffling, and nod-and-shuffle modes.

2. Exposure Control: execution of single or multiple exposure sequences, integration timing, shutter triggering.
3. FITS header creation: collects data from the ICS, TCS, and other subsystems, and binds this information into the FITS headers for each image created. FITS header cards will conform to MODS and LBT FITS Header Dictionaries.
4. Diagnostic modes for detector health monitoring, testing, and troubleshooting.
5. Runtime logging to provide a basis for troubleshooting.

The unique code required for the DCS is a kernel-level device driver for the PCI version of the OSU/Atwood sequencer-based detector control hardware. This will be the focus of most of the development work on the DCS. This module will develop standard *ioctl* functions to implement I/O operations, and use interrupt masking to ensure that the system is not interrupted during readout.

4.4 Image Motion Compensation System (IMCS)

The Image Motion Compensation System (IMCS) for each MODS is comprised of the collimator system module (§4.2.7) for each channel, the infrared laser, and a quadcell detector located next to each CCD detector (one for each channel). A single IR laser is launched from the focal plane and split into red/blue beams as required by the Dichroic Beam Selector optics. From there, it is directed to the "bypass grating" mounted inside the center of each "science" grating, and directed through the camera into the Germanium quadcell detectors. Motion of the IR laser spot will mimic motion of the images in the CCD plane due to gravity-induced flexure of the spectrograph structure, non-deterministic mechanical "ticks" caused by stochastic strain relief between different structural members, and temperature induced lateral image motions (it cannot correct for temperature-dependent changes in path length, aka focus). The system compensates for this image motion by steering the beam using the collimator mirror tip/tilt/focus actuators. Each channel must run closed loop for full compensation.

The IMCS must integrate control functions (laser on/off, mirror steering in both channels), sensing (IR quad cell measurements), and computations for compensation commands. Because both channels must be run independently, it will require a multithreaded architecture.

The first phase of the IMCS is in development in the lab, and is a purely engineering and test-and-evaluation system with no user interface to speak of implemented as a BASIC language application running on a DOS system. The second generation system will be used to develop the basic collimator system module (§4.2.7), and be developed in a Linux environment using ISIS client/server code. This, too, will be primarily an engineering system, but will develop the essential code infrastructure on which the deployment system will layer a GUI geared towards observers. This third-stage system will be used during lab acceptance and "flexure" testing at OSU prior to shipment and commissioning of MODS1.

During commissioning and early operational phases between the deployment of MODS1 and the commissioning of MODS2, the IMCS interface will look to the observer like a second "autoguider" system, this one guiding an artificial IR laser "star" using the collimator mirrors. It will require separate setup (and hopefully minimal tweaking), much in the same way that a guide star has to be acquired and the telescope guiding loop closed, before science data acquisition can proceed. A major task of this third-stage system is to gather data that will allow us to better understand how the systems interact in the real LBT environment, and how best to have it interact with observers. It is a sufficiently novel system that we have no prior experience to inform us.

The ultimate goal is to have IMCS functions operate below the notice of most observers, and just be an "engage/disengage" button on the main MODS GUI. We expect that this final system will be deployed with MODS2, as a more mature IMCS that runs as a turn-key system.

The IMCS, by its nature, requires a high degree of concurrency, and will likely be the most complex multithreaded application in the entire MODS system.

4.5 Data Handling System (DHS)

The data-handling system for MODS handles all FITS images created by the detector systems from each spectrometer. Hardware requirements for the DHS are being developed, but the current concept is a central computer with a dedicated 1Tb RAID1 or RAID5 disk array. Each DCS has its own data disks ("immediate" image storage), and the DHS coordinates copying the data from each DCS to the central storage array. It will also be responsible for queuing data images onto the observatory archive system when the latter is designed/specified by the LBTPO.

The DHS is also responsible for coordinating data logging, including creation of a nightly "observing log" which is built by scanning the FITS headers of newly-created images and adding their entries to the log with appropriate time tagging. A logging tool will be created that allows observers to annotate this log with their own notes, and to export a personal copy of the log in various formats (e.g., TeX/LaTeX, CSV, HTML table, or flat ASCII text).

Observers will make copies of their images for transport home using the DHS. Given the size of MODS images (two full 2-channel MODS with 4K×8K CCDs generate 256Mb of data per 4-image data set coded as 16-bit integers), the volume of data expected for night's worth of observing plus daytime calibrations is such that standard but slow removable technologies like recordable CD or DVD are impractical. The current working concept is to employ removable hard drives, most likely IEEE 1394 ("Firewire") drives that are currently available in 200−300Gb formats hardened for travel at a typical cost of approximately $US1/Gb. Whatever our final choice, the system will be designed flexibly enough to accommodate whatever relevant technologies are available in 2006 and beyond (e.g., FireWire or USB2.0 solid-state drives in multi-Gb capacities, FireWire800 drives, etc.).

4.6 MODS User Interface (MUI)

The MODS User Interface (MUI) will be an X11/Qt GUI application. The look-and-feel will follow the emerging LBT GUI Guidelines (see LBT CAN 481s010a). The MUI provides all of the functions required to observe with MODS.

The basic conception is a user interface that has multiple, cascading layers of complexity from simple to black belt. The basic operational levels are as follows:

Standard "Built-In" Configurations: Here a user configures MODS by selecting from among a suite of built-in standard configurations. This gives MODS a "press of a button" configuration option. Standard configurations include:

1. Single- or dual-channel imaging.
2. Long-slit spectroscopy (low- and medium-resolution) using a limited number of "primary" setups with well-characterized grating and fixed long-slit combinations. Such primary configurations are a common way to work with complex instruments for routine observing (e.g., the STIS primary configurations for imaging and long-slit spectroscopy on the Hubble Space Telescope).
3. Standard calibration modes for flat fielding, lamp calibration, bias (zero) image sequences, etc.
4. Standard sequences for collimator focus and camera focus optimization, instrument and guider focus conjugation, etc.

Custom Observing Templates: Custom configuration of the instrument by executing pre-prepared observing template files (obs files). Configuration is "select-and-execute", whereby an obs file selected from a suite of obs files that have been uploaded to the observing console in advance of the night. These would be used, for example, with MOS-mode spectroscopy where the primary setup overhead should be aligning the mask to the sky, not choosing the spectrometer configuration (complex observing setups require advance planning). Setup overhead is reduced by using observing templates as these naturally dovetail with the concurrent mechanism configuration capability of the ICS.

Low-Level Configuration: Configuration of the instrument at the single-component level (e.g., filter, grating, collimator focus, etc.) "by hand" via GUI controls. Examples include an observer fine-tuning a central wavelength by tweaking the grating tilt, tweaking exposure times. The low-level instrument configuration GUI will permit uploading complex configuration choices for concurrent mechanism setting for efficiency.

Command-Line Level Configuration: This allows the observer to "pop the hood" on the GUI and provides a command-line interface for the execution of single commands or command scripts. This serves as a low-level "black-belt" interface that would most likely be used by the MODS Team and LBTO MODS support personnel for routine instrument checkout.

A particular challenge of the design of the MUI is how to make an accessible user interface for a very complex instrument. This becomes even more of an issue when we consider that with both MODS operating we must coordinate dual-instrument as well as dual-channel configurations. A key element of the MUI design is to ensure that we do not build in restrictions on "dual-dual" operations with both MODS on the LBT. This will be a unique corner of parameter space, so we are going to leave our options open.

4.7 MODS Engineering Support Interface (MESI)

The MODS Engineering Support Interface (MESI) is a suite of applications for engineering test and evaluation of all MODS components.

At the highest level, the first layer of MESI is the lowest command level of the MUI (§4.6, item 0) repackaged: a command-line interface that operates at the component level. It adds to this layer options for verbose diagnostic output (simple trace through functions), and super-verbose debugging output (e.g., device-level tracing of configuration operations, like watching drive step counts and sensor bits change as a mechanism moves or doesn't as the case may be). The main MESI command tool will provide both graphical and command-line interfaces, based on the same model as the MUI (but with an extended and potentially more dangerous command set than mere observers are permitted – everything you need to fix it or frag it).

In addition to the main MESI tool will be a suite of individual "mechanism agent" applications that provide engineering-level control of major components (e.g., a program for operating all camera system functions while a camera is mounted either in MODS or on its servicing cart). These are the same tools that we will be using during MODS component assembly, integration and testing.

All MODS software systems produce copious runtime telemetry logs tracing operations in time-tagged mode that should permit troubleshooting by reconstructing event histories. MODS is an order of magnitude more complex than any of our previous instruments, so we are investigating the best tools for traversing these log files. Ideas under investigation include importing the log files into a relational database system (like MySQL) and adapting standard database tools for analysis of the telemetry database.

4.8 MODS Mask Design and Manufacturing System

Since both MODS and LUCIFER will share a laser mask cutting machine, we should also share the same look-and-feel for the mask design and manufacturing systems, so that LBT observers do not need to learn two different systems. The MODS mask-making system will be developed in collaboration with the LUCIFER team.

The Mask Making System consists of the following components:

Mask Design System: This is the tool used by astronomers to examine CCD/IR images of their fields taken with MODS/LUCIFER, generate object catalogs, and design slit masks for these targets. Models for this are the GMOS (Gemini) and FORS/VIMOS (VLT) mask making systems, both of which have a common heritage, and will be familiar to US and European LBT partners. The "product" of this program is a "mask design file" that contains the detailed mask design (specification of the locations and shapes of the slits in units of millimeters on the mask substrate). The mask design file is then submitted, via a web interface to the ...

Mask Manufacturing System: This is software that will reside at LBTO with the laser cutting machine. It takes as input the mask design file and translates the mask parameters into laser cutting machine commands to create the mask in the substrate. Each slit mask will be given a unique numerical identifier in the form of a Code 39 or Code 128/EAN barcode. This label will be affixed to the mask and used to track its location. Mask design files are submitted via a web interface. This interface also enters the mask into the ...

Mask Inventory and Tracking System: Once a mask is designed and assigned an identifier, it enters the "Observatory Mask Inventory". Tracking software, modeled on commercial software for tracking packages from shelf-to-customer, will be used to track the progress of the mask from design to deployment and storage. A model for this system is the Lick/Keck mask tracking system developed for the DEIMOS instrument, which has to track masks from design submission to the server in Santa Cruz to fabrication, storage, and deployment on Mauna Kea. This system has proven to be robust and reliable, and Lick personnel have offered us the basic code for their system. Observers and support personnel can learn where a mask is (being made, being shipped to the mountain, on the mountain, in MODS, in storage) by visiting a webpage.

As part of the software development, we will evaluate software and hardware options for creating robust barcodes (likely thermal transfer printing), hand barcode scanners, and inventory/tracking databases and develop the necessary web interfaces.

4.9 MODS Observing Preparation Software (MODSTools)

User-friendly "Phase 2" Observation Template preparation tools. MODS observing template (.obs) files are the basic units for configuring MODS and controlling data acquisition. Obs files are analogous to ESO-style Observing Blocks, but simpler to create and modify (they are human readable and in principle human editable, though tools will be provided to ensure format integrity). Most observers will use obs files, though they will also have full low-level configuration and data-taking control through the MUI. Key components of this system are:

1. MOD sky simulator, which superimposes a view of the MODS science and guider fields on a digitized sky survey image of the field in questions. Useful for targeting, pre-selection of guide and wavefront-sensing stars, and selection of instrument position angles (e.g., to align the slit to include structure and/or avoid bright stars).
2. Web-based Observing Template preparation, editing, and tracking tools. The prototype is the (admittedly much simpler) Phase 2 observing preparation tools used by the ANDICAM as part of the CTIO 1.3m telescope SMARTS operations (see www.astronomy.ohio-state.edu/ANDICAM/Obs).
3. Observing Sequencer Tool that will let observers chain together groups of Observing Templates into complex observing sequences. Precise details are TBD, but options include Java-based web tools (analogous to HST phase 2 observing tools), or portable host programs (e.g., self-contained Qt-based GUI apps distributed as binaries).

All of these tools would be available on host computers at LBTO, and sources/binaries provided to all LBT partners to allow local generation of obs files, observing sequences, and observing planning. The suite of tools would most likely be bundled with the Mask Design System software (§4.8).

5 Software Development Plan

5.1 Existing Software

ISIS client/server software exists and has been field tested. This establishes the interprocess communications layer (UDP socket and serial ports), and the command protocol layer (IMPv2 messages) for the MODS data-taking system. Both are mature or nearing maturity, and are stable release code. Internal documentation uses Doxygen markup, and current documentation may be viewed at www.astronomy.ohio-state.edu/LBT/MODS/Software/.

A prototype motor-control API has been developed for the filter wheel component of the MODS Camera System (§4.2.9). We are using an Applied Motion Products (www.appliedmotion.com) Si series stepper-motor drive, which has an RS232 interface and uses a proprietary (yet simple) SiNet Command Language (SCL) for drive control. A prototype SiUtils API has been written, currently supporting basic I/O to the drives and motion control functions. A testbed ISIS client agent named *fwheel* has been written to evaluate the controllers and their API for use with the MODS blue filter wheel. Earlier versions (using a less general version of the SCL interface) are being field tested in 12-position filter wheels of the MODS design in instruments deployed at the MDM 1.3m and CTIO 1.0m telescopes earlier in 2004. So far they perform flawlessly.

A more general engineering tool that can operate most MODS mechanisms one-on-one has been written (provisionally named "sitool"). It provides high-level commands for prototyping basic motion-control functions (e.g., move to an absolute step-count position, move +/– steps relative to the current position, move to a specific indexed position, etc.), plus complete access to position sensor I/O and all low-level stepper-motor controller functions. It is also a command-line interface application written using the ISIS client API, which allows it to be controlled, for example, by Perl scripts for multi-motion testing, life testing, etc. It is already seeing use in the lab as we begin the assembly and testing of MODS mechanical subsystem.

Multi-threading for concurrent motions has been prototyped at the simulator level (currently capable of operating 7 phantom mechanisms concurrently). Incorporation of actual motion-control hardware and links to the SiUtils API is in development. The first instrument subsystem to test concurrent configuration will be the prototype Collimator System actuators (§4.2.7). This will lead to the development of the 3-axis IMCS prototype using the quad sensor in the lab this summer, forming the basis for the production IMCS system. In all, we should have all of the drive layers of the ICS completed by the end of 2004 Q3.

A prototype barcode reader API has been developed to operate a Microscan (www.microscan.com) MS-3 CCD barcode scanner connected to a serial port. This is implemented as a standalone MSUtils API, and is in alpha testing at this time (June 2004). Beta release should follow in July 2004. Work defining the barcode generation and mask inventory system is in progress for Summer 2004.

5.2 MODS Software Development Team

The MODS software development team consists of the following personnel:

Richard Pogge – MODS Project Scientist and Software Development Team Leader.
Overall responsibility for overseeing the software design, development, implementation, testing, and documentation activities for MODS. Has so far provided API code for interprocess communications, motor controller, and barcode scanner operation, and will primarily focus on observing interfaces for use by astronomers and support personnel, and various observatory interfaces, particularly the TCS and GCS. He is also the primary point of contact with the LBTPO programming team.

Jerry Mason – Systems Developer/Engineer
Primary responsibility for the design, development, implementation and testing of the CCD detector control and data handling systems, particularly developing the Linux

device drivers for the Atwood CCD controller PCI sequencer card. Also responsible for hardware related projects. Responsible for documenting these systems.

<u>Systems Developer/Engineer #2</u>

We are currently (2004 August) seeking to hire a second systems developer/engineer with C/C++ programming expertise to take on the main programming tasks for MODS with the Software Team. Their primary responsibility will be creating the GUI code and overall integration of the software systems that constitute the MODS instrument control system. This person will also be responsible for maintaining the OSU CVS archive, and coordinating with the LBTPO team to keep their mirror of our source codes up to date, and for maintaining the online software documentation site at OSU.

During the main development and deployment phases, the Software Team will meet weekly to review progress and set priorities for the following week. When problems emerge, this team with the project engineers will form a Tiger Team for attacking specific problems. All members will submit weekly progress report forms to the project manager for incorporation into the overall project database in order to track progress globally with respect to the MODS project.

5.3 Development Stages

The software development priorities for MODS are as follows, in order:

1. Design the MODS Software System and develop the functional requirements for the system components, and establish the development priority and implementation plan. The product is this document.
2. Adapt the existing ISIS client/server system for MODS, creating a formal ISISClient API that implements the communications transport and messaging layers of the system (essentially complete as of 2004 June).
3. Development, testing, and integration of the main software components of the Instrument Control System (ICS). This software is required for the testing and integration of the MODS mechanisms. This work began in April 2004 and will proceed through Summer and Fall 2004. Specific priorities are:
    a. Design and test the motor-control interfaces for the adopted stepper-motor drives. Code is being testing with real mechanisms to verify designs and implementations. The product will be the SiUtils API that will be used to implement the motion control functions of the ICS.
    b. Perform use case analyses of each of the primary ICS components (§4.2), leading to detailed functional requirements documents for the system "agents" that will control each ICS component or component groups.
    c. Design and test the mechanism control agents, especially development of multithreading routines for concurrent configuration of mechanisms where applicable to the mechanism. Order of development for particular agents will parallel integration and testing of the first examples of these mechanisms.
    d. Design and test the master ICSAgent application that will coordinate and control the individual mechanism-group agents. ICSAgent will be the main

interface to the MODS instrument mechanical systems seen by other elements of the MODS data acquisition system.

4. Design and Prototype low-level Observatory interfaces. These are given advanced priority because of the need to provide early feedback to the LBTO efforts during the telescope commissioning phases. The main priorities are:
    a. TCS interface, based on the IIF library and TCS simulator provided by LBTPO (Tim Schmelmer and team at Heidelberg). Product is the TCSAgent to be used by MODS.
    b. GCS interface for the MODS AGW stage, developed in collaboration with the GCS team at Heidelberg (Schmelmer, Leibold, & Kuerster). Creation of a simulator and API for the GCS team is a priority for 2004 (the GCS team's main priorities for 2004 as we understand them are for the LBC and the Potsdam AGW units for secondary commissioning). Iteration with the Heidelberg GCS team will lead to development of the production system when the MODS AGW stage is assembled and tested (thus well in advance of MODS lab acceptance and commissioning).
5. Design and develop the MODS user interfaces. For the GUI-based interfaces (e.g., the MUI and MESI), this involves lots of user testing to get the human factors engineering right. Basic steps
    a. Start with writing simple Qt-based GUI applications to cut our teeth on the package. This has already begun, but it will mostly await hiring of the new programmer.
    b. Develop Qt classes that interface with ISIS, forming the basis of our MODS/Qt suite.
    c. Adapt current tools and practices (e.g., observing templates, scripts, etc.) into the interface functions.
    d. Adapt our low-level engineering development tools into the components of the MESI system.
    e. Develop tools for consolidating the various MODS telemetry log files generated by the main software components into a database with a suite of search and analysis tools to aid in using these logs for event history reconstruction for troubleshooting problems.
6. Specify, design, and develop the slit mask design and manufacturing system. This will proceed in collaboration with the LUCIFER team. Details of development responsibilities and division of labor among the two teams need to be discussed later in 2004. Main tasks are:
    a. Design and develop the mask design modules. The model will be the mask design system used by the Gemini GMOS spectrometer, a close peer to MODS in characteristics. The product of this software should be a "mask design file" with the slit characteristics and coordinates in machine-readable form.

   b. Design and develop the mask barcode tracking and inventory system. The model for this is the Lick/Keck mask tracking system used with the DEIMOS spectrometer (elements of which have been offered to OSU by the Lick design team).

   c. Design and develop the mask manufacturing software that translates a common mask design file format into machine manufacturing instructions for a laser machining system. This development must naturally await a final decision by the MODS and LUCIFER teams on the exact choice of laser machine.

7. Design, development and testing of the Detector Control System and its related Data Handling System. The main DCS system is based on our existing DOS-based CCD controller system using an Atwood "Sequencer" architecture for array control. Programmer Jerry Mason will have primary responsibility for the development of the MODS DCS. Main tasks are:

   a. Specify, design, develop and test a Linux kernel device driver for our PCI sequencer card which controls I/O between the host computer and the detector "head electronics" (the clock/bias boards, ADCs, and detector drive electronics). We will design to the current v2.4 kernel, but maintain design compatibility with the emerging v2.6 kernel.

   b. Design, develop, and test a Linux version of our old "IC" (Instrument Controller) software currently implemented in BASIC on an MS-DOS PC. The new IC will be written in C as an ISIS client application, and use standard libraries (e.g., cfitsio for FITS file generation).

   c. Specify, design, and develop the data-handling system that will be integrated with the detector control system. Rigorously test the data handling system with real and simulated data fed to it at rates comparable to that expected from normal observing scenarios. The system must satisfy stringent requirements of data integrity, speed, and ease of access.

   d. Specify and test various data transport methods to be used by observers to get their data from the telescope to their home institutions.

8. Integration and Testing of the MODS Data-Acquisition System.

   a. First phase is integration and testing of MODS in the lab at OSU.

   b. Second phase is full system integration preparatory to lab acceptance testing at OSU prior to shipment to Mt. Graham

   c. Third phase is post-shipment installation and testing at Mt. Graham preparatory to installation of MODS on the telescope, and full testing of the LBT observatory interfaces (general network, TCS, GCS, etc.).

   d. Fourth phase is testing of MODS on the telescope as part of MODS Commissioning activities, including testing and qualification of on-site and remote observing interfaces, data-handling system, data analysis, etc.

9. MODS/LBTO Handover

   a. Presentation of final user documentation to the LBTPO.

b. On-going development and maintenance of all MODS-related systems at LBTO and OSU.

The actual efforts for Commissioning and Handover are still being developed at this writing.