An Aries Software Roadmap

January 2004

1 Introduction

ARIES is a high resolution infrared imager and spectrometer that will exploit the capabilities of adaptive optics (AO) at the upgraded 6.5-meter reflector of the MMT Observatory on Mt. Hopkins. It consists of four imaging scales that yield diffraction-limited performance and two principal spectroscopy modes that offer resolving powers of $R = \lambda/\Delta\lambda = 100 - 60,000$ from 1 to 5 μm in a relatively compact instrument. This flexibility places significant demands on the software that will control the myriad of optics, gratings, filters, and observing modes that ARIES will require. Thoughtful design will allow the ARIES software to be easily maintained, adapted and extended toward new capabilities and even entirely new instruments at Steward Observatory. The following characteristics define the most demanding software requirements that we require:

- **Low Latency**: Integrating with an adaptive optics (AO) system requires that software requests be carried out with a minimum of overhead and latency. Although ARIES itself should not need a dedicated real time OS, its kernel should be optimized for the lowest reasonable latency. The control software will also be scheduled to at a higher priority than most other processes on the system. A baseline specification for latency might require that 90% of all software requests be initiated within 1 millisecond.

- **Fully Modular Design**: Breaking up the control software into distinctive, independent modules will allow the software to be more adaptable to other (future) instruments. The user interface should be decoupled from the actual hardware control, for example. This modularity improves code maintainability and allows one to work on an individual part of an instrument without interfering with other components.

- **Remote Operation**: Although adaptive optics systems are sufficiently complicated that remote observing with ARIES is unlikely in the near future, it is still important to provide full remote access via ethernet. Instrument status and health can be monitored from Tucson, and 2 AM troubleshooting sessions can be more easily and conveniently diagnosed and worked around.

- **Open Source Components**: Using Open Source Software as the foundation for the instrument software provides lowest development cost, and critical independence from commercial product obsolescence. High quality operating systems (e.g. Linux, Darwin, FreeBSD), compilers (e.g. gcc) and development toolkits (e.g. GTK+) are available which meet and often exceed the capability of commercial products.

- **Observation Planning Tools**: Visualization of the usable fields of view upon 2MASS images, and visualizing the frequency coverage of echelle orders on the spectroscopic array is pivotal to making efficient use of observing time with ARIES.
- **Data Visualization and Assessment**: One of the most common problems with control software is assessing the quality of the data being taken. A separate quick-look package that can perform sky subtraction, basic flat-fielding, and very simple aperture photometry from a graphical interface is essential to an observer. A coarse pipeline for reduction of echelle data is similarly necessary for ARIES spectroscopy.
- **Data Storage and Organization**: Data storage and organization is often the least-considered aspect of instrument software development, yet it is one of the largest long-term headaches. We aim to develop an automatic system that stores data, writes observing logs, and databases all observations taken with ARIES for easy retrieval.

To realize all of these goals, a control system centered upon the Linux operating system is designed (Figure 1). Control of ARIES occurs via a Linux kernel 2.6 driver that commands ARIES' Leach camera controller via an internal PCI interface card. A low level kernel interface naturally separates the divergent issues of 1) hardware access and 2) user interface. A modular set of *core functions* with an "expert level" command line interface yields full control over all aspects of the ARIES instrument. The same set of functions is also accessed by a friendlier *graphical user interface* (GUI), which is what observers will use. The GUI will represent a compromise between the complete flexibility of the command line, and the usability of a graphical interface. Naturally, it will utilize the "most common" subset of the core functions, though the command line will also be accessible in the GUI. *Remote access* can be readily established by running the control software remotely and transparently over the Internet by tunneling X11 over ssh. Secondly, a web page with the latest instrument status can be displayed on any browser, even on PDAs. Finally, the graphical user interface can be run separately on remote computers, and only the instrument requests (not the entire graphical window itself) would be routed over the Internet. The possibility of instantly mirroring acquired data to the remote computer can also be considered.

2 ARIES Instrument Control System

We now explore each of these components in more detail.

2.1 Kernel Mode Drivers

All low-level hardware control is performed through kernel level interfaces. The advantages of doing kernel-level access is speed (non-preemptibility), low latency and a high

degree of hardware abstraction. That is, user interface software can control an ADC by performing basic `read()`, `write()`, and `ioctl()` operations upon a character device, such as `/dev/interfacecard/adc`. The job of actually communicating with that card, registering it with the system, handling power management, etc. is the responsibility of the kernel driver itself, and not the higher level ARIES software. For ARIES, most of the instrument control occurs through a PCI interface card, for which a usable kernel driver

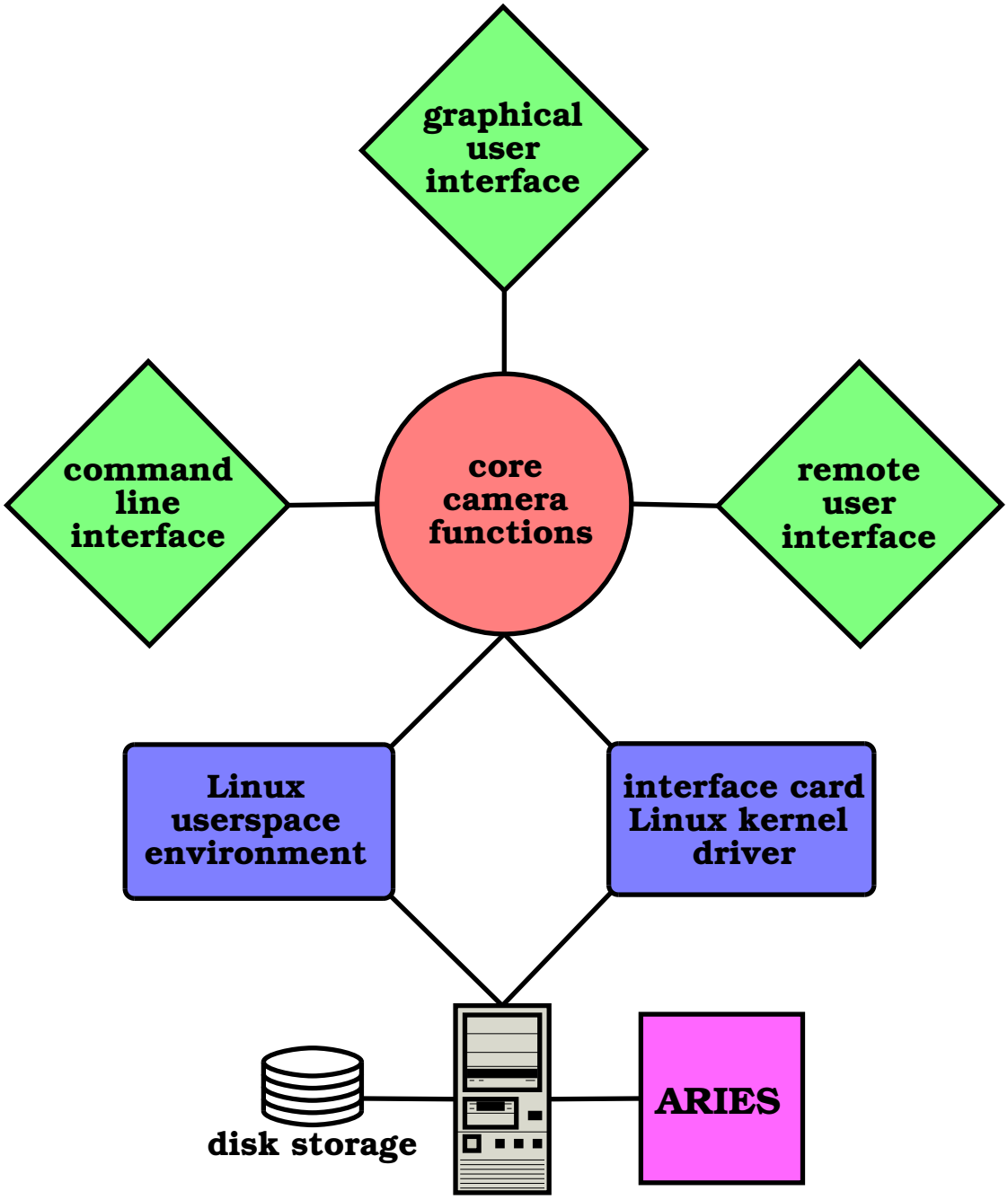

Figure 1: Block diagram of ARIES software system

exists. Additional coding of the kernel driver is necessary to use the card with Linux kernel 2.6 and to improve error handling for robustness. Other hardware access, such as serial or parallel communications, can be performed through the appropriate mainstream kernel driver and can be entirely decoupled from ARIES software development.

2.2 Core Functions

A series of fundamental functions serve as the primary interface to the kernel mode hardware drivers. Each function should have a focused scope, performing a particular task. They are essentially "building blocks" that can be assembled and combined in various ways to perform progressively more complicated tasks. They can be enhanced and expanded to suit multiple instruments. They serve as the calls that actually do "real work", and are called by the various user interfaces. The baseline language will be "ordinary" C; if a more object-based approach is deemed better, C++ or Objective C (both supersets of basic C) may be used.

Examples of functions might be `download_ucode()`, `move_motor()`, `write_FITS()`, or `activate_array()`. The `astropci` API for performing basic detector array operations constitutes an important part of the Core Functions. Some functions may in turn call worker functions more specialized to a particular instrument or mechanism. Code duplication will be minimized by separating "generic" code from "specific" code.

2.3 User Interfaces

The Core Functions described above provide a a library-like interface to the hardware, but do not provide a user interface. One interface is not suitable for every circumstance however. An instrument scientist debugging ARIES in the laboratory may require far more flexibility and control over esoteric instrument features than the typical observer at the telescope. Furthermore, each instrument may warrant its own specific interface. It becomes necessary, then, to decouple the user interface not only from the kernel-level drivers, but also from the Core Functions. Multiple/different user interfaces can simply call upon the desired subset of Core Functions.

2.3.1 Command Line Interface

The command line interface is designed to draw upon all functionality in the Core Functions. It is designed for an instrument scientist (or advanced observer) for debugging, optical alignment, etc. The terseness of the command syntax means that the operator must know what [s]he is doing, though the commands and parameters should be fairly intuitive to a fairly seasoned infrared observer. All commands provide fairly verbose feedback and quantitative measurements of the instrument performance. The command language is very loosely based upon the basic syntax of the NOAO "wildfire" control

system. Specifying a command without arguments will show the current status of that command, and show the set of possible changes. A set of commands and parameters follows:

Instrument Commands

- `abort`: terminates the current observation and discards any data
- `activate`: activates the detector array
- `deactivate`: deactivates the detector array
- `diskspace`: ask how many images can fit on the current filesystem
- `download <ucode>`: kill off any microcode that is running, download new microcode from file <ucode>, and run it
- `setup`: configure the detector array, set bias levels for the 4 quadrants, etc.
- `freeze <on | off>`: disallow any changes to the detector array configuration, aside from activating and deactivating
- `status`: print thermal sensor, motor encoder and detector array status. This is the basic housekeeping command.
- `motor control`: Motion of the `slit`, `grating`, `optics`, `prober`, `adc`, `array` or `filter` motors can be performed by typing their names, followed by the position to travel to. Simply typing the name will show the current position, and the range of available positions. Example: `grating 4712.9 wn` or `filter Ks`, `optics f15` or `slit 0.2`.
- `observe`: initiate an observation. The current set of observing parameters is displayed for confirmation or editing, after which the observation is executed and data saved and archived.
- `go`: a shortcut for `observe`, performing an observation using the current setup without display or confirmation.
- `help <command>`: display basic help for a command
- `move <ra|dec|alt|az> <N>`: Move telescope in a given direction by N arcseconds
- `wobble <RA><DEC>` Specify a telescope wobble vector, offset in arcseconds from the current position in RA/DEC. This provides nodding observations that allow background subtraction.

- `raster <L><M><N>`: Raster the telescope in an LxM matrix with N arcsecond grid spacing. The offset for sky subtraction is specified in the `wobble` command, and relative to the raster map center.
- `movie`: Continual readout of the array with current parameters
- `bedtime` A shortcut for `viewer dark`, `slit blank`, `lyot dark`, and `deactivate` to be used at the end of a night of observing.

Instrument Parameters

- `coadds`: number of hardware-coadded frames to be accumulated to a single saved image.
- `reads`: number of low noise reads. 2 reads represent double sampling, >2 represent multiple reads.
- `pics`: number of images to obtain
- `exptime`: integration time in seconds
- `filename`: base filename for saved data
- `index`: number index to append to the filename
- `directory`: specify the directory for saved data
- `jitter`: specify a small amount of random motion to each move in a nodding or raster mode observation.
- `comment`: add a comment to the FITS header
- `title`: add a title to the FITS header
- `mode`: specify observing mode, such as "stare", "chop", or "nod"

2.3.2 Graphical User Interface

Although it will contain a more restrictive subset of functionality represented by the Core Functions, a graphical user interface (GUI) is more intuitive and less imposing than the corresponding command line interface. The GUI will include a text field in which text-mode commands can also be issued. The interface will be prototyped in parallel using both the GNUstep (using Objective C) and GTK+/GNOME (using C) development environments. The interface model will use a tabbed interface for organization and to minimize screen clutter, and will follow the most sensible user interface guidelines published by the GNOME Foundation and NeXT/Apple. Final decisions regarding the GUI await the first prototypes.

2.3.3 Remote User Interface

The remote user interface development follows two principal paths. The first remote access method (Figure 2a) requires no additional work; it simply exploits the remote view capabilities of X11 (X Windows). The remote operator can log into the ARIES control computer via ssh and start a new instance of the control software, displaying on the remote machine (UNIX, Mac OS X, or MS Windows with X11 software installed). The second method (Figure 2b) runs the control software on the remote machine (UNIX, Mac OS X) and can simply send instrument requests to the ARIES computer using network sockets.

A fundamental design decision concerning both textual and graphical interfaces is the use of network sockets in general. One design would have network sockets be the default communication mechanism for all commands, local or remote. Constructed in this fashion, the physical location of the observer (i.e. local or remote) is merely a detail and not relevant to the core code.

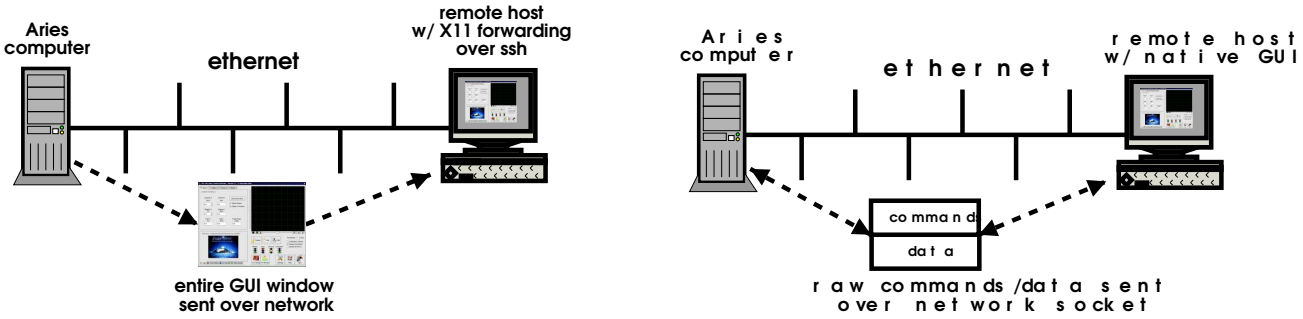

Figure 2: Methods of remote operation include remotely exporting the X11 display, and sending commands from a remote interface over a network socket.

3 Visualization of Data Products

3.1 Quick-Look Data Processing

SAO DS9 will be used to display images with the command-line interface, and can also be used in place of the rudimentary image display built into the graphical user interface (GUI). The GUI display routine will optionally display a sky-subtracted image, based upon an average of the nearest two adjacent sky frames. It will eventually be capable of displaying panoramic raster maps (averaging pixel values and aligning images by shared point sources in adjacent frames). Spectroscopic data can be coarsely processed by 1) identification of the continuum in each order, 2) summing an aperture defined by the FWHM of the seeing profile, and 3) displaying the resulting 1D spectrum with an approximate wavelength solution identified from atmospheric features. OH airglow lines

will be used for $\lambda < 2.3$ μm, and a composite spectrum will be computed from the HITRAN molecular database for $\lambda > 2.3$ μm.

3.2 Data Archival

In our automatic data archival scheme, every image taken with ARIES is saved. The data is saved locally, but optionally could be mirrored to a networked filesystem. The FITS header for each image is written into a relational data base program (e.g. MySQL or Postgres) which facilitates efficient logging and retrieval of the data. The observer can execute a separate graphical program which accesses the database, consolidates the requested scans into a list, provides tools for inspecting the raw data, and most importantly, co-adds and background-subtracts the data for viewing. The output is a set of FITS files, or alternately a FITS cube. In principle, access to these data products to the greater scientific community could be provided through a (Java-based) web browser interface that will interface with MySQL and the FITS data cubes.

3.3 Observation Planning Tools

As a guide to new observers, two new tools for observation planning will be developed. An imaging tool will overlay 2MASS data release images atop the ARIES fields of view for the 4 different "plate" scales. A spectroscopy tool will show the frequency coverage of the echelle orders on the detector array for a given grating angle.