The MONSOON Implementation of the Generic Pixel Server

P. N. Daly and N. C. Buchholz

MONSOON Project, Major Instrumentation Program, National Optical Astronomy Observatory, 950 N. Cherry Avenue, P. O. Box 26732, Tucson, AZ 85726-6732, USA.

ABSTRACT

MONSOON is NOAO's diverse, future-proof, array controller project that holds the promise of a common hardware and software platform for the whole of US astronomy. As such it is an implementation of the Generic Pixel Server which is a new concept that serves OUV-IR pixel data. The fundamental element of the server is the GPX dictionary which is the only entry point into the system from instrumentation or observatory level software. In the MONSOON implementation, which uses mostly commercial off-the-shelf hardware and software components, the MONSOON supervisor layer (MSL) is the highest level layer and this communicates with multiple Pixel-Acquisition-Node / Detector-Head-Electronics (PAN-DHE) pairs to co-ordinate the acquisition of the celestial data. The MSL is the MONSOON implementation of the GPX and this paper discusses the design requirements and the techniques used to meet them.

Keywords: OUV-IR controllers, MONSOON, GPX

1. INTRODUCTION

The MONSOON image acquisition system is documented in other papers in these proceedings[1, 2] and references therein. Rather than cover such ground again, we show in Figure 1 a schematic of the design of MONSOON as applied to the NOAO Extremely Wide-Field Infra-Red Mosaic (NEWFIRM) camera.[3, 4] Here, a single supervisor node controls two PAN-DHE pairs with data delivered to the data handling system (DHS) without flowing through the observation control system (OCS). The topic of this paper is the highest level of MONSOON software: the *Supervisor Layer* (MSL).

2. REQUIREMENTS

The principal requirements, and desirable attributes consistent with the MONSOON paradigm, to be met by the supervisor layer are that it shall:

1. Be a single point-of-entry into a MONSOON system from (multiple) high level software (HLS) clients;
2. Understand and recognize GPX[5] commands only;
3. Handle connection security for protection of the focal plane;
4. Remain responsive at all times;
5. Maintain and monitor connections to a number of subservient PANs;
6. Send commands to subservient PANs;
7. Receive command responses from subservient PANs;
8. Return status information to HLS clients;
9. Respond, in a deterministic way, to error conditions;

Further author information: (Send correspondence to PND)
PND: pnd@noao.edu, NCB: ncb@noao.edu

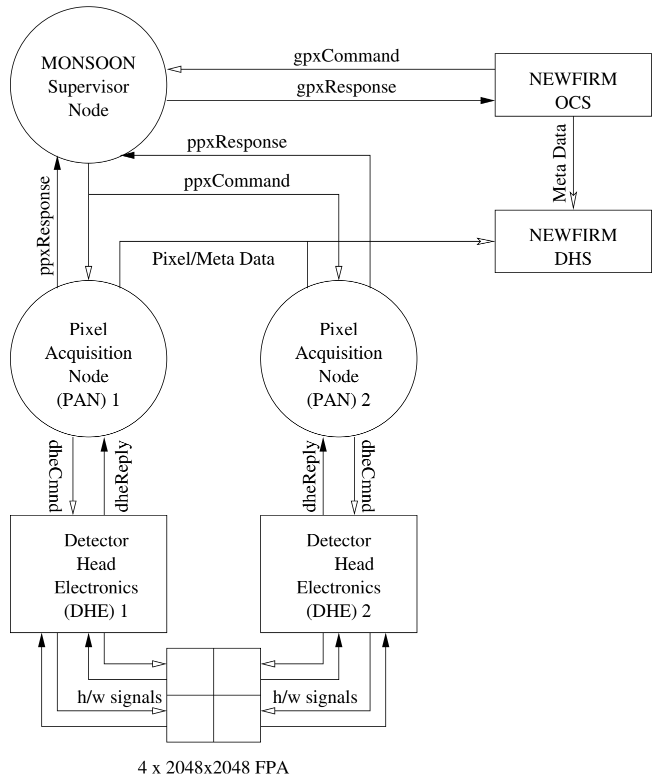

Figure 1. MONSOON Image Acquisition System as Applied to NEWFIRM.

10. Respond, in a deterministic way, to asynchronous messages generated by 1 or more PANs;
11. Provide a logging capability for post-mortem analysis;
12. Handle urgent messages;
13. Utilize common facilities provided by a freely-available OS:
    (a) Be a programming interface using sockets;
    (b) Use shared libraries (for code re-use);
    (c) Use shared memory, semaphores and/or queues as appropriate;
    (d) Run on a designated PAN or its own CPU-host system.

Since a key requirement is that the supervisor remain responsive at all times, this rules out a polling approach. The supervisor must, therefore, accept commands and—if such commands cannot be executed immediately—record them for future execution in the order in which they were received. Urgent commands are defined as those that can usurp the normal order of command execution and be forced to execute immediately. For example, a *gpxAbort* is an urgent command. Further analysis of the requirements for command storage and execution leads to the following desirable functionality:

1. Incoming commands should be stored in order for priority-based execution;
2. Pending commands can be saved to disk (to provide backup during error recovery, for example);
3. Saved commands can be restored (from disk) in a suitable order for execution (even whilst other pending commands are present via a suitable interweaving technique);

4. Urgent commands are flagged in a unique way so that they receive priority treatment;
5. The implementation should be efficient to provide low overhead during command searches.

Simply numbering commands as they arrive (starting at 0 and increasing) can be done but the save/restore requirements along with the ability to handle urgent messages and recover after system reboots *etc.* means dynamic re-assignment to keep the command queue correctly ordered with, potentially, high overhead. Furthermore, maintaining such lists is unnecessarily complex.

What is needed is a unique, monotonically increasing, time stamp that can be associated with each command.

3. MONSOON STAR DATES

A MONSOON Star Date (MSD) can be thought of as a hyper-accurate Julian day (JD). That is, the MSD comprises the actual JD plus the fraction of a day since midnight (in the local time frame) accurate to 1 μs. Since the JD increases at midnight and we use the canonical *gettimeofday()* function to calculate the day-fraction, we guarantee that the MSD can *never* be repeated even after a machine reboot[*]. So, we have developed a library of code for handling MSDs and their conversion to and from Julian days, modified Julian days, Lilian days, Gregorian dates and the ubiquitous *yyyymmdd* string format. This code is available as a MONSOON shared library, as a Tcl package and as loadable functions into a *PostGreSQL* database. An example MSD, as returned in Tcl, is:

```
[monsoon@decapod]% wish
% load $env(MONSOON_LIB)/libmsdTcl.so
msd package v1.0.0, P. N. Daly, (c) AURA Inc, 2004. All rights reserved.
% msd::msd
2453129.4587745796889067
```

Since the MSD is *always* unique and monotonically increases over time, its properties have distinct advantages in maintaining time-stamped lists:

- The latest (or newest) command to be added always has the maximum MSD;
- In normal priority mode, the oldest command is executed next;
- A FIFO buffer is handled by searching for the minimum MSD;
- A LIFO or FILO buffer is handled by searching for the maximum MSD;
- Urgency can be addressed by simple negation of the MSD.

This last point is worthy of further discussion. First, it is the responsibility of the HLS client to issue commands in the correct execution sequence. Second, when such a command is accepted, it receives its associated MSD and both the MSD and the command are recorded in separate arrays but at the same array index. So, for example, let us say, we have a queue with the following MSD-stamped commands already present but that have yet to be executed:

```
  2453129.4767786306329072 "gpxSetAVP intTime=6.6"
  2453129.4768044417724013 "gpxSetAVP fSamples=16"
  2453129.4768252740614116 "gpxSetAVP coadds=2"
```

[*]Of course, the pedantic reader could argue that we can never guarantee such uniqueness since clock drift between two machines—both of which are producer machines that can talk to a separate consumer machine—can generate badly sequenced, or even non-unique, MSDs. This is why the *ntpd* daemon should be enabled on all such systems. *Caveat emptor, caveat lector, Romani ite domum etc.*

By searching the MSD array for the minimum value, we see that we can recover the order by simple, and efficient, array traversal. That is, the execution order (in this case) is the same as the order displayed in the example. This need not be the case as we can save the MSDs in a random order (depending on where the next available slot in the array is) and still obtain the next command in the correct sequence. Therefore, queue re-ordering is not required.

What happens, though, if we need to precede a command with a prior command or send an urgent message? For example, suppose we wish to execute the above commands but precede them with a *"gpxSetMode FastIR"* so that the detector is setup in a given named mode. In this case, we simply obtain the next (monotonically increasing) MSD and *negate* it before filing so that the yet-to-be executed queue now looks like:

```
 2453129.4767786306329072 "gpxSetAVP intTime=6.6"
 2453129.4768044417724013 "gpxSetAVP fSamples=16"
 2453129.4768252740614116 "gpxSetAVP coadds=2"
-2453129.4768352430633316 "gpxSetMode FastIR"
```

Since the minimum MSD gets executed first, then, we recover the *"gpxSetMode FastIR"* before the *"gpxSetAVP ..."* commands. Thus, the use of MSDs is a simple, elegant and powerful technique for time-stamped queue management.

4. THE SUPERVISOR LAYER DESIGN

The preliminary design of the MONSOON Supervisor Layer is shown in Figure 2 on page 5. Identifiable elements are described below. It is anticipated that some security mechanisms will be implemented using standard techniques such as firewalls, allowed hosts and authentication checking. Internal security techniques are not discussed here (for obvious reasons).

4.1. The gpxCmdStr_t Shared Memory Segment

The following shared memory segment is defined:

```
#define GPX_MAXCOMMANDS 1024
#define GPX_MAXNODES     256
#define GPX_MAXMSG      4096
#define GPX_MAXGLOBALS  1024
typedef struct gpxCmdStruct {
 double gpxMsdInit;
 double gpxMsd[GPX_MAXCOMMANDS];
 long gpxGlobals[GPX_MAXGLOBALS];
 long gpxSysStatus[GPX_MAXNODES];
 char gpxCmnd[GPX_MAXCOMMANDS][GPX_MAXMSG];
 char ppxCmnd[GPX_MAXCOMMANDS][GPX_MAXNODES][GPX_MAXMSG];
 char ppxResp[GPX_MAXCOMMANDS][GPX_MAXNODES][GPX_MAXMSG];
 long ppxRecv[GPX_MAXCOMMANDS][GPX_MAXNODES];
 long ppxSent[GPX_MAXCOMMANDS][GPX_MAXNODES];
 long ppxStat[GPX_MAXCOMMANDS][GPX_MAXNODES];
} gpxCmdStr_t, *gpxCmdStr_p, **gpxCmdStr_s;
```

Note that it is a design decision to make this a static array. The contents of the array *gpxGlobals[]* has yet to be fully determined but some elements are defined:

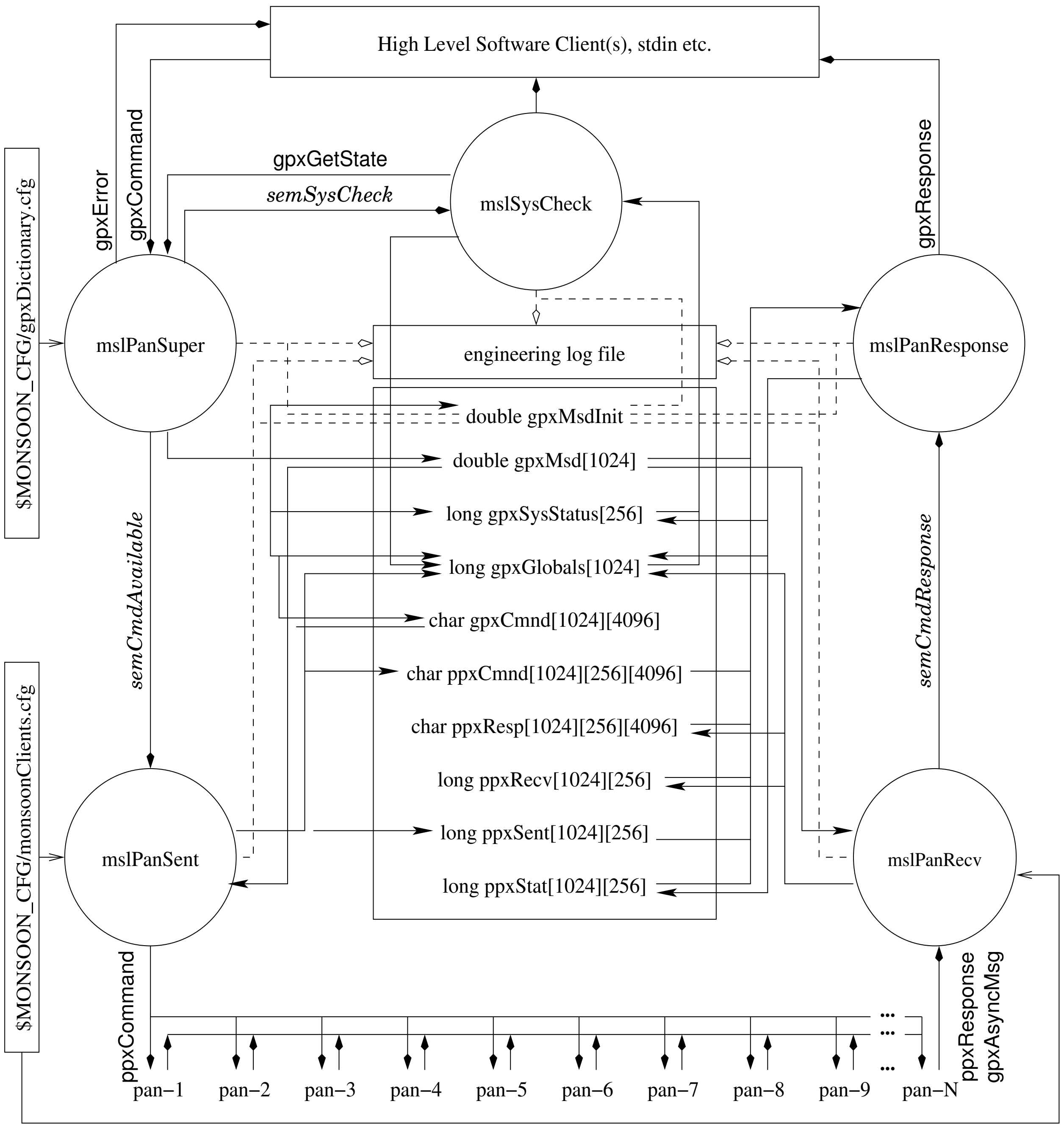

Figure 2. MONSOON Supervisor Layer Preliminary Design.

```
#define GPX_GBE_SYSID             0 /* first array element */
#define GPX_GBL_SYSID    0xABCDEF00
#define GPX_GBE_STATUS            1 /* second array element */
#define GPX_GBL_OK       0x00000000
#define GPX_GBL_ERROR    0xFFFFFFFF
#define GPX_GBL_WARNING  0xFFFFFFFE
#define GPX_GBL_INFO     0xFFFFFFFD
#define GPX_GBE_STATE             2 /* third array element */
#define GPX_GBL_READY    0x05011961
#define GPX_GBL_BUSY     0x18101995
#define GPX_GBL_PAUSE    0x22101959
#define GPX_GBL_INIT     0x02111991
```

4.2. mslPanSuper

This is the main entry point process into the MSL. On entry, *mslPanSuper* checks to see that the username/password authenticates and, if so, allows access to the system. Thereafter, it creates and initializes the shared memory segment. In particular, the *gpxMsd[]* array is initialized with `MAXDOUBLE` values whilst other elements are initialized with `0` or `NULL`. *mslPanSuper* also opens and listens on a well-known port for incoming connections and establishes a handler to deal with such connections. A logfile is also opened using the startup MSD, *gpxMsdInit*, as a unique name. It then enters a loop that can be terminated by a *gpxExit* command or an appropriate signal.

Inside the loop, *mslPanSuper* uses the *select()* mechanism to listen on *n*-client sockets plus *stdin*[†] for commands. On receipt of a command, it does the following:

1. Scans the string for a known GPX command as the first or second element. If the command is not recognized, it immediately returns an error to the HLS client;
2. Checks *gpxMsd[]* for free slot availability. If a slot is available it is stored in local variable *recno*, otherwise it immediately returns an error to the HLS client;
3. Checks for a prefix MSD (*i.e. viz.,* as the first element in the incoming command string). If present, it extracts it and records the given MSD/command in *gpxMsd[recno]*, *gpxCmnd[recno][0]* respectively;
4. If no prefix MSD is present, generates a unique MSD and records the MSD/command in the appropriate slot;
5. Sets appropriate elements in the *gpxGlobals[]* array;
6. If the queue is not paused, gives the counting *semCmdAvailable* semaphore;
7. If the *select()* times-out, gives the *semSysCheck* semaphore;
8. Logs all the above actions to the logfile with appropriate MSD timestamps.

Note that when the command is written into *gpxCmnd[recno][0]*, it is *always* written with the MSD as a prefix. That is the command *"gpxSetAVP intTime=6.6"* associated with MSD 2453129.4767786306329072 is written into *gpxCmnd[recno][0]* as *"2453129.4767786306329072 gpxSetAVP intTime=6.6"* as well as storing the MSD in *gpxMsd[recno]*.

When the loop terminates, outstanding semaphores are given and released, the shared memory segment is detached and destroyed, all applicable sockets are closed and the logfile is closed.

[†]This is included for debugging and development purpose and will be removed for delivered systems.

4.3. mslPanSent

This is the process that sends commands to the PANs. On entry, it reads a configuration record that specifies the number of connected PANs, their IP-addresses and port numbers to use. It opens each socket connection for *write-only* and reports any dysfunctional connections. It then opens the GPX shared memory segment defined in § 4.1, opens the shared logfile, and enters a loop terminated by *gpxExit* or an appropriate signal.

Inside the loop, it waits on the *semCmdAvailable* semaphore and, when received, does the following:

1. Finds the minimum value of the *gpxMsd[]* array. If no minimum exists, it continues from the top of the loop;
2. Finds the slot (and stores it in local variable *recno*) associated with the minimum MSD;
3. Converts the GPX command to a PPX command and writes them to *ppxCmnd[recno][panID][0]*;
4. Sends the PPX command to each socket and returns the write status into each *ppxSent[recno][panID]*;
5. Sets appropriate elements in the *gpxGlobals[]* array;
6. Logs all the above actions to the logfile with appropriate MSD timestamps;
7. Takes the *semCmdAvailable* semaphore.

When the loop terminates, outstanding semaphores are given and released, the shared memory segment is detached, all applicable sockets are closed and the logfile is closed.

4.4. mslPanRecv

This is the process that receives responses from the PANs. On entry, it reads a configuration record that specifies the number of connected PANs and their IP-addresses and port numbers to use. It opens each socket connection for *read-only* and reports any dysfunctional connections. It then opens the GPX shared memory segment defined in § 4.1, opens the shared logfile, and enters a loop terminated by *gpxExit* or an appropriate signal.

Inside the loop, it does the following:

1. Finds the minimum value of the *gpxMsd[]* array. If no minimum exists, it continues from the top of the loop;
2. Finds the slot (and stores it in local variable *recno*) associated with the minimum MSD;
3. Sets up an appropriate *select()* in a loop to receive replies;
   - When a given socket descriptor becomes ready, read the stream and record the response in *ppxResp[recno][panID][0]*;
   - Returns the read status or timeout flag into each *ppxRecv[recno][panID]*;
   - Check incoming MSD with expected and, if different, treat this message as *gpxAsyncMsg*;
4. Sets appropriate elements in the *gpxGlobals[]* array;
5. Gives the *semCmdResponse* semaphore;
6. Logs all the above actions to the logfile with appropriate MSD timestamps.

When the loop terminates, outstanding semaphores are given and released, the shared memory segment is detached, all applicable sockets are closed and the logfile is closed.

4.5. mslPanResponse

This is the process that checks responses from the PANs. On entry, it opens a socket connection for *write-only* to the HLS client. It then opens the GPX shared memory segment defined in § 4.1, opens the shared logfile, and enters a loop terminated by *gpxExit* or an appropriate signal.

Inside the loop, it waits on the *semCmdResponse* semaphore and, when received, does the following:

1. Finds the minimum value of the *gpxMsd[]* array. If no minimum exists, it continues from the top of the loop;
2. Finds the slot (and stores it in local variable *recno*) associated with the minimum MSD;
3. Determines a cumulative status by checking:
   - The contents of *ppxCmnd[recno][panID][0]* against the status in *ppxSent[recno][panID]*;
   - The contents of *ppxResp[recno][panID][0]* against the status in *ppxRecv[recno][panID]*;
   - The contents of *ppxResp[recno][panID][0]* against that expected from the command in *ppxCmnd[recno][panID][0]*;
   - Records the status in *ppxStat[recno][panID]*;
4. Based on the above, sends an appropriate response to the HLS client;
5. Sets appropriate elements in the *gpxGlobals[]* array;
6. Logs all the above actions to the logfile with appropriate MSD timestamps.
7. Re-initializes the now obsolete slot to make it available for further commands;
8. Takes the *semCmdResponse* semaphore.

When the loop terminates, the shared memory segment is detached, all applicable sockets are closed and the logfile is closed.

4.6. mslSysCheck

This is the process that checks on system wellness from time to time. On entry, it opens a socket connection for *write-only* to the HLS client. It then opens the GPX shared memory segment defined in § 4.1, opens the shared logfile, and enters a loop terminated by *gpxExit* or an appropriate signal.

Inside the loop, it waits on the *semSysCheck* semaphore and, when received, does the following:

1. Checks the *gpxGlobals[]* array for any suspicious values and reports them to HLS client;
2. Checks the *gpxSysStatus[]* array for any suspicious values and reports them to HLS client;
3. Sends an urgent *gpxGetState* to *mslPanSuper* to force a check of all PANs availability (from time to time);
4. Sets appropriate elements in the *gpxGlobals[]* array;
5. Logs all the above actions to the logfile with appropriate MSD timestamps.
6. Takes the *semSysCheck* semaphore.

When the loop terminates, the shared memory segment is detached, all applicable sockets are closed and the logfile is closed.

5. FUTURE WORK

The design presented in §4 has yet to be coded and, no doubt, difficulties will arise as use-cases are tested against it. We believe, however, that the design meets all the requirements of a preliminary stage coding. Nevertheless, we can foresee the design changing to allow for:

- Authenticating the unique 128-byte identifier code on each master control board;
- Configuration and named mode select via database access using the 128-byte code as a master key;
- GPX 1-to-many mapping to the PPX. In the above design, each GPX command is passed wholesale to the subservient PANs. However, there are instances when GPX commands with many arguments can be mapped to several single-argument PPX commands which this design does not, yet, address.

REFERENCES

1. N. C. Buchholz and P. N. Daly, 2004, *The MONSOON Generic Pixel Server Software Design*, Proc. SPIE Vol. 5496, Advanced Software, Control, and Communication Systems for Astronomy, Hilton Lewis, Gianni Raffi, Eds. (this volume).
2. P. N. Daly, N. C. Buchholz and P. Moore, 2004, *Automated Software Configuration in the MONSOON System*, Proc. SPIE Vol. 5496, Advanced Software, Control, and Communication Systems for Astronomy, Hilton Lewis, Gianni Raffi, Eds. (this volume).
3. R. G. Autry, R. G. Probst, B. M. Starr, K. M. Abdel-Gawad, R. D. Blakley, P. N. Daly, R. Dominguez, E. A. Hileman, M. Liang, E. T. Pearson, R. A. Shaw and D. Tody, 2002, *NEWFIRM: the wide-field IR imager for NOAO 4-m telescopes*, Proc. SPIE Vol. 4841, Instrument Design and Performance for Optical/Infrared Ground-based Telescopes, Masanori Iye, Alan F. Moorwood, Eds. pp525-539.
4. R. G. Probst, N. Gaughan, G. Chisholm, P. N. Daly, E. A. Hileman, M. Hunten, M. Liang, K. M. Merrill, and J. Penegor, 2004, *Project Status of NEWFIRM: the wide-field infrared camera for NOAO 4-m telescopes*, Proc. SPIE Vol. 5492, Ground-based Instrumentation for Astronomy, Masanori Iye, Alan F. Moorwood, Eds. (in press).
5. N. C. Buchholz and P. N. Daly, 2004, *The Generic Pixel Server Dictionary*, Proc. SPIE Vol. 5496, Advanced Software, Control, and Communication Systems for Astronomy, Hilton Lewis, Gianni Raffi, Eds. (this volume).