Southern African Large Telescope
Prime Focus Imaging Spectrograph
Control System Software Design Document
SALT-3140AE0022
Jeffrey W Percival

Modification Record

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>02-Mar-2003</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>05-Mar-2003</td>
<td>Update reconfig times, add discussion of parallelism</td>
</tr>
</tbody>
</table>

Table of Contents

- Introduction
- Overview
- Environment
- Client/Server Design
- Interlocks
- Server Modes
- State Management
- Observing Procedures
- Error Management
- Data and Command Management
- Communication with Other SALT Subsystems

Introduction

This note describes the software design for the PFIS control system.

Overview

PFIS uses 9 mechanisms in support of its observing states. These mechanisms use a total of 8 stepper motors and 16 pneumatic actuators.

The software must provide the following capabilities:
- Configure mechanisms to support observing states
- Enforce interlock conditions to avoid damage to the instrument & operators
- Provide instrument status data sufficient to support operations, monitor instrument performance, and allow error recovery by the operators

**Environment**

The PFIS Control System software runs on a standard Windows (currently Windows 2000) PC. The software is written in the National Instruments (NI) graphical programming language LabVIEW. Windows is required for at least part of the control system software because although LabVIEW is supported on Mac OS and Linux, National Instruments currently supports motion control only under Windows.

**Client/Server Design**

We choose a client/server design, in which the user interface is the client, and the server runs the hardware. Each is a stand-alone piece of software, and they communicate over a network using LabVIEW's data socket server (DSS) function. The DSS provides a simple network communications function, and allows both client and server to run on the same CPU or different CPUs without modification of the software.

We also choose the format of the commands passing from the client to the server to be simple text strings (e.g. "mask fetch" or "grating insert"). This allows scripting in a natural way. Human-readable observing scripts will be generated in advance using web-based perl CGI scripts. The PFIS user interface (the client) can then simply read these scripts and forward the commands to the server. The server doesn't need to know about the source of the commands, and no script-dependent software will reside in the server.

Note that choosing a client design that does not use any LabVIEW motion control modules means that the client does not suffer the same operating system restriction that the server does. The client can run on Linux, or Mac OS.

The server runs on PCON, a Windows PC with a NI MXI fiber interface card in its backplane. This is the communications link to the payload. The server module is driven by a vocabulary of simple text strings. The strings represent commands at three different hierarchies of abstraction:

- low-level commands, specific to a given stepper or pneumatic
- mid-level commands, which operate at the whole-mechanism level
- high-level commands, or procedures, that execute observing sequences

This figure shows the server block diagram.
The PFIS user interface reflects this hierarchy, and uses LabVIEW tabbed controls to reveal the desired level of control at the right time.

Screen shots of the prototype GUI are shown in SALT-3140AE0023.

**Interlocks**

PFIS is designed with hardware interlocks sufficient to protect the instrument from damaging itself, and to prevent exposing humans to electrical shock or harm from actuators. See SALT-3140AS0015 in the PFIS CDR package. The server, however will also enforce the interlocks in software, with the goal of never actually invoking a hardware interlock.
We will implement the software interlocks at the lowest level, by "wrapping" each actuator inside a LabVIEW VI module. The VI will be responsible for check that the interlock conditions have been met before actually sending the control signal out into the hardware. If the conditions are not met, the wrapper VI will fail and pass an error cluster up the hierarchy that describes what condition was not met.

**Server Modes**

We define these modes for the PFIS server.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>All power off</td>
</tr>
<tr>
<td>Standby</td>
<td>PCON on, PXI Chassis on</td>
</tr>
<tr>
<td>Init</td>
<td>Includes sensing the instrument configuration</td>
</tr>
<tr>
<td>Ready</td>
<td>Waiting for command input</td>
</tr>
<tr>
<td>Busy</td>
<td>Processing a command</td>
</tr>
<tr>
<td>Error</td>
<td>Any detected error, including sensor error, interlock error, or actuator hangup.</td>
</tr>
</tbody>
</table>

![PFIS Control System Server Modes](http://www.sal.wisc.edu/PFIS/docs/archive/protected/pfis/3140/pfis-sdd/3140AE0022-sdd.html)
The normal sequence of mode transitions is 1-2-3-4-5-6-7.

The Error state is entered in three different ways:

- Transition 8: initialization error. This usually means that PFIS was unable to sense which of its configuration states it was in.
- Transition 9: self-check error. PFIS will continually perform a number of self-checks in the Ready mode. The various self-checks are described in the Interlocks Specification document SALT-3140AS0015.
- Transition 10: command failure. This can happen if a reconfiguration sequence fails due to a hardware failure, or if an error is sensed from the Detector subsystem or Telescope Control System.

PFIS will spend most of its time in one of two states: Ready or Standby. PFIS will seldom go to Off because if the PXI box is turned off, then no sensor data are available during the day, and the PFIS control computer must be rebooted each time the PXI box is turned on.

**State Management**

In a combinatorical sense, PFIS has a huge number of possible states. It offers imaging, spectroscopy, polarimetry (3 kinds), 2 resolution regimes of Fabry Perot narrow band imaging, 3 types of CCD readout modes, and optional interactions with the telescope during an observation. The combinatoric explosion is controlled, though, by pruning whole sections of the configuration state tree according to which states are useful, and which are nonsensical.

This figure shows the useful PFIS configuration states. There are 3 main states (S1-S3), with 3 variants that select polarimetric modes (S4-S6). This figure shows the possible state transitions, and the time it takes to execute them.
This state-space approach lends itself to a simple but powerful software implementation. We will control the PFIS configuration with a table-driven state machine. What is this?

**PFIS State Transition Table**

How to use this table: the rows are indexed by the state you are currently in. The columns are indexed by the final (not next) state that you want to end up in. For each (current, desired) state pair, the indexed table cell tells which transition to execute next, and what state that takes you to. Keep iterating until you end up in the desired state (you are done when you end up on the diagonal).

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>-</td>
<td>+T1, S2</td>
<td>+T3, S3</td>
<td>+T2, S4</td>
<td>+T1, S2</td>
<td>+T3, S3</td>
</tr>
<tr>
<td>s2</td>
<td>-T1, S1</td>
<td>-</td>
<td>-T1, S1</td>
<td>-T1, S1</td>
<td>-T2, S5</td>
<td>+T1, S1</td>
</tr>
<tr>
<td>s3</td>
<td>-T3, S1</td>
<td>-T3, S1</td>
<td>-</td>
<td>-T3, S1</td>
<td>-T3, S1</td>
<td>+T2, S6</td>
</tr>
<tr>
<td>s4</td>
<td>-T2, S1</td>
<td>-T2, S1</td>
<td>-T2, S1</td>
<td>-</td>
<td>+T1, S5</td>
<td>+T3, S6</td>
</tr>
<tr>
<td>s5</td>
<td>-T2, S2</td>
<td>-T2, S2</td>
<td>-T2, S2</td>
<td>-T1, S4</td>
<td>-</td>
<td>-T1, S4</td>
</tr>
<tr>
<td>s6</td>
<td>-T2, S3</td>
<td>-T2, S3</td>
<td>-T2, S3</td>
<td>-T3, S4</td>
<td>-T3, S4</td>
<td>-</td>
</tr>
</tbody>
</table>

Example: Suppose you are in state S5 (Spectropolarimetry), and you want to be in state S3 (Fabry-Perot Imaging) for the next observation.
1. (S5,S3) indexes the cell (-T2, S2). So execute transition -T2 (Remove waveplates). You are now in state S2 (Spectroscopy).
2. (S2,S3) indexes the cell (-T1,S1). So execute transition -T1 (Articulate camera back to home). You are now in state S1 (Imaging).
3. (S1,S3) indexes the cell (+T3,S3). So execute transition T3 (Insert etalons). You are now in state S3.
4. (S3,S3) indexes the diagonal. You are done.

This table-driven approach has many advantages over a procedural approach. For example, in the example given above, one made the move from S5 to S3 by:

- de-articulating the camera arm to its home position
- inserting the Fabry-Perot etalons.
- removing the waveplates and polarizing beamsplitter

If, for some operational reason, one wanted a different order, say:

- removing the waveplates and polarizing beamsplitter
- de-articulating the camera arm to its home position
- inserting the Fabry-Perot etalons.

then one has merely to update the state table. No procedures have to be written or modified. This approach has been used before at the Space Astronomy Laboratory, in Jeff Percival's amazing astrometric kernal, the "Telescope Pointing Machine" (TPM). The state diagram and detailed documentation are available on the web.

The PFIS configuration state table also represents a higher, 3rd level of interlock protection. The state table entries ensure that the instrument will be moved through a tested and well-known sequence of states. The hardware interlocks are shadowed by the low-level LabVIEW software interlocks, ensuring that the hardware interlocks will not be tripped. Likewise, the low-level interlocks are shadowed by the state table entries, ensuring that the software interlocks will not be tripped!

One might complain that using a table-driven state machine precludes the possibility of saving time by operating mechanisms in parallel. We point out, though, that the two longest configurations, the etalons and the camera articulation, are mutually exclusive in a mechanical sense, and therefore cannot be done in parallel. Moreover, the three reconfigurations that don't result in a state change (slit masks, gratings, and filters) can be done in parallel, and we will use LabVIEW's parallel execution capabilities to do so.

**Observing Procedures**

PFIS commands are organized into a 3-level hierarchy, allowing access at the actuator, mechanism, or procedure level. We want to avoid controlling PFIS observing sequences by sending large numbers of low-level commands from the client (user interface) to the server. The knowledge of how to execute an observing sequence should lie in the server.

We have identified the observation sequences that make sense for each of the configuration states. These are discussed in the Operational Concepts Definition Document (SALT-3170AE0002).
Error Management

We will manage errors using the standard error management tools in LabVIEW. Each VI will take an error input and provide an error output. The error flow will be wired up, one VI after another, in LabVIEW fashion. Each VI will inspect its error input, and decline to act if an error is indicated. If action is declined, then the input error will be passed on. If action is taken and an error is produced, then that error cluster will be passed on.

The error cluster will contain a specific description of the error: the interlock code, the sensor responsible for the failure, and so on.

Combined with a hierarchical set of VIs, this method will allow errors to flow up to the top, then over to the client interface.

Data and Command Management

All PFIS commands (text strings) and data clusters will be defined as LabVIEW "type def" clusters. The typedef clusters will all live in a single directory, allowing a convenient way of inspecting or using them en masse.

Communication with Other SALT Subsystems

PFIS must communicate with two other SALT subsystems: the SAAO CCD subsystem (PDET) and the Telescope Control System (TCS).

It sends set-up commands to PDET for configuring the CCD for exposures, and trades information with PDET for handshaking, slit mask peakup exposures, and FITS header keywords. No raw image data are exchanged; the interactions will be conducted using text strings. The vocabulary will be built up during the fabrication phase by the SAAO and PFIS teams.

PFIS interacts with the TCS to do slit mask peakups, and for the shuffle-and-nod observing procedures. These interactions will also use text strings. The vocabulary will be built up during the fabrication phase by the TCS and PFIS teams.