SAM

Software Interface of Visitor Instrument with SAM

Prepared by: R.Cantarutti

Version: 2.2

Date: November 9, 2005

File: visitor_pdr.sxw

1 Introduction

This ICD document specifies the principal system software interface that lies between Visitor Instrument and SAM computing system.

2 Overview

At a minimum, any visitor instrument to be used with SAM must communicate through TCP/IP using the SCLN protocol for interprocess communication and control. This requirement directly derives from the SOAR control system environment in which SAM and the visitor instrument operate. Document "SOAR Telescope Control System" presents and overview of the SOAR control system environment.

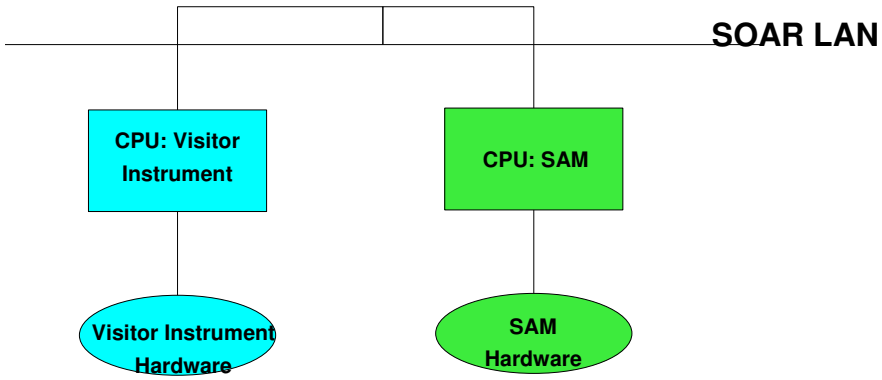

Figure 1. System Architecture. The Visitor Instrument communicate using the SCLN protocol across the SOAR control LAN.

The SOAR control system follows the client-server paradigm based on a layered design for communication. A *service layer* provides a service-independent software interface to the clients and servers in the computing environment. This layer encapsulates

the interprocess communication methods and is implemented by the SCLN library package. The SCLN is described in the document “The SOAR Communication Library New (SCLN)”.

On the other hand a *command layer* is built upon the service layer to make functionality available to other distributed processes (e.g. Visitor Instrument). This layer is represented by the set of commands that a server makes available for its clients.

Within the SOAR environment, the SAM computing system is enabled to act like a server over the SOAR Local Area Network. The Visitor Instrument will have to be enabled to act as a client to requests actions on SAM.

3 Implementation

The SCLN library package provides, as part of its native implementation, the necessary modules to enable a Labview application to act as a client. The Visitor Instrument may choose to use the SCLN implementation, depending on its decision to use Labview as software development environment.

A Visitor Instrument that chose not to use Labview will then have to implement the SCLN client-server protocol based on the examples provided with the SCLN library package and using the programming language of their preference. Appendix A in this document provides as a first reference an example of an implementation using C as the programming language.

The Visitor Instrument will receive an IP address and a port number to connect and then start sending commands to the SAM computing system.

4 Command Interface

The SAM command interface is the set of string commands available to the remote applications (e.g. Visitor Instrument) to perform actions on SAM. This set as described in section two represent the command layer of the communication between boxes in the SOAR control system environment.

The commands are given as messages consisting of a command keyword and optional parameters. A message is terminated with a message delimiter, that could be a CR or LF character.

All commands returns a specific response as a numeric string, or a status of the requested operation. Normally this is the string "ok", meaning that the command was accepted and acted upon. If the operation is a motion that takes some time to complete, then the string "moving" is returned. To determine the end of the motion, you need to

poll by sending again the same command but without arguments, until the string "ok" is received. An error condition is reported with a message consisting of the word 'error', followed by a number, then the character ':' and finally a short text describing the error. For example: error 4: motion already active. Then, the possible responses to a command are:

- a numeric string starting with a digit or number sign (e.g. -23.5 02:34:15.9)
- the string "ok"
- the string "moving"
- a string starting with the word "error"

The following list doesn't pretend to be complete but will serve as a first reference on the format of the available commands.

aoloop [open|close]

Open/Close the AO control loop

adc [enable|disable|stop|calc|move]

Actions on the ADC.

guider [enable|disable|move|stop|type|x|y|park|optical|xrate|yrate]

This command operates on the guider devices. If no parameter is given then returns the present x, y positions or the status of the motion.

info

Returns image header information.

instrument [n]

Selects between visitor and dedicated instrument configuration.

Appendix A

Example implementation of the SCLN protocol using ANSI C.

```
/*
 Procedures for SOAR TCS Communications.
 The command protocol is client/server with
 immediate response. A response should never take
 longer than 500 mS.
*/
#include <sys/socket.h>
#include <sys/time.h>
#include <sys/times.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <time.h>

#define INVALID_SOCKET 0

static int m_Socket = INVALID_SOCKET;

int OpenTCSSocket(char *host, int port);
int Demand2TCS(char *command, char *response);
void CloseTCSSocket();
void flaten(char *line);

/*
 Test example for SOAR communications
 The port number is 30040
*/
main(int argc, char *argv[])
{
 char host[80], command[80], response[128];
 int l;

 if (argc < 2)
 {
  printf("Usage: tsc IP number\n");
  exit(0);
 }
 strcpy(host, argv[1]);
 if (OpenTCSSocket(host, 30040) < 0)
 {
  printf("Error openning socket\n");
  exit(0);
 }
```

```
 while(1)
 {
  printf("-> ");
  fgets(command, 80, stdin);
  if (command[0] == 'q')  // quit on 'q' command
  {
   CloseTCSSocket();
   exit(0);
  }
  l = strlen(command);
  command[l-1] = '\0';  // get rid of end of line
  if (Demand2TCS(command, response) < 0)
  {
   printf("error in command %s\n", command);
   continue;
  }
  printf("*** %s\n", response);
 }
}


/*
 Open the TCS connection. The connection uses
 sockets under TCP/IP.
*/
int OpenTCSSocket(char *host, int nPort)
{
 union MA
  {
   char line[256];
   int l;
  } ma;
 int n, ll;
 int nResult;
 char response[256];
 struct sockaddr_in m_SocketAddress;
 struct timeval tv;
 fd_set fds;

 m_Socket = socket(AF_INET, SOCK_STREAM, 0);
 if (m_Socket == INVALID_SOCKET)
  return(-1);
 if (fcntl(m_Socket, F_SETFL, O_NONBLOCK) == -1)
 {
  close(m_Socket);
  return(-1);
 }
                                          // Setup the socket address
structure
   m_SocketAddress.sin_addr.s_addr = inet_addr(host);
   m_SocketAddress.sin_family      = AF_INET;
   m_SocketAddress.sin_port        = htons(nPort);
```

```
 nResult = connect(m_Socket, (struct sockaddr *) &m_SocketAddress,
    sizeof(m_SocketAddress));
 if (nResult == -1)
 {
  FD_ZERO(&fds);
  FD_SET(m_Socket, &fds);
  tv.tv_sec = 0;
  tv.tv_usec = 5000;
  nResult = select(m_Socket + 1, NULL, &fds, NULL, &tv);
  if (nResult != 0)
  {    // If we are connected, wait for
      // the acknowledge message.
   FD_ZERO(&fds);
   FD_SET(m_Socket, &fds);
   tv.tv_sec = 0;
   tv.tv_usec = 500000;
   n = select(m_Socket + 1, &fds, NULL, NULL, &tv); // wait ack
   if (n == 0) return(-1);   // timeout
   n = recv(m_Socket, &ma.l, 4, 0); // get buffer length
   flaten(ma.line);   // into C order
   n = recv(m_Socket, response, ma.l, 0); // get command response
   response[ma.l] = '\0';   // NULL is C terminator
   if (!strcmp(response, "DONE"))
    return(0);  // OK
   else
    return(-1);  // Authentication error.
  }
  else
   return(-1);  // Error in connection
 }
 return(0);
}
/*
 Send command to TCS and receive response.
*/

int Demand2TCS(char *command, char *response)
{
 union MA
  {
   char line[256];
   int l;
  } ma;
 int n, ll;
 fd_set fds;
 struct timeval tv;

 if (m_Socket == INVALID_SOCKET) return(-1);
 ll = strlen(command);
 ma.l = ll;     // header is string length
 flaten(ma.line);    // into LabVIEW order
 strcpy(&ma.line[4], command);   // add the command
```

```
 n = send(m_Socket, (char *) ma.line, ll+4, 0); // send the package
 if (n != (ll+4))
 {      // error sending
  close(m_Socket);
  m_Socket = INVALID_SOCKET;
  return(-1);
 }
 FD_ZERO(&fds);
 FD_SET(m_Socket, &fds);
 tv.tv_sec = 0;
 tv.tv_usec = 500000;
 n = select(m_Socket + 1, &fds, NULL, NULL, &tv); // wait for response
 if (n == 0) return(-1);    // timeout
 n = recv(m_Socket, &ma.l, 4, 0);  // get buffer length
 flaten(ma.line);    // into C order
 n = recv(m_Socket, response, ma.l, 0);  // get command response
 response[ma.l] = '\0';    // NULL is C terminator
 return(0);
}
/*
 Close the socket
*/
void CloseTCSSocket()
{
 close(m_Socket);
 m_Socket = INVALID_SOCKET;
}
/*
 Cast the byte order from/to LabVIEW/C
*/
void flaten(char *line)
{
 char local[4];

 local[3] = line[0];
 local[2] = line[1];
 local[1] = line[2];
 local[0] = line[3];
 line[0] = local[0];
 line[1] = local[1];
 line[2] = local[2];
 line[3] = local[3];
}
```