NCCS - User Interface Design and Implementation

A. Götz

March 20, 1996

1 Introduction

The NCCS **User Interface** will differ quite considerably from the present user interface on the HP. It will try to avoid the pitfalls of the present interface which makes it difficult to use and maintain. This paper will present the basic design principles and implementation of the user interface.

2 Design Principles

The user interface will be based on the following design principles :

1. **modern UI concepts** - modern user interfaces differ considerably from the purely alphanumerical based interfaces which predominated in the days of the mainframe and the early days of DOS. User interface building is now recognised as a field on its own requiring special programming and design skills. Modern user interfaces aim at being *user friendly* and put the *user in charge* (as opposed to the computer). These aims are achieved in part by using graphics, user driven input (the user decides when and what), the mouse and a high level scripting language for customizing.

2. **scripting** - one of the basic interfaces to the NCCS will be via a high level language which can be used interactively or for executing scripts. The high level language will be based on the *Tcl* interpreter. Tcl can be easily extended using `C` or `C++` to include local commands. Tcl is a very popular language which runs on most platforms and which has a large user community. Tcl has recently been chosen as scripting language for the *VLT* control system (ref. *Messenger, January 1996*). The local extension to Tcl will be called `obsh` (for `ob`server's `sh`ell).

3. **application programmer's interface** - having only a scripting language is not sufficient as a user interface. It is also necessary to have a low-level interface to the system which can be called from `C` and `C++` (and eventually `Fortran`). These languages offer what Tcl does not i.e. *abstract data types*, *data manipulation* and *efficiency*. The low-level interface will be via an *application programmer's interface* (API). The API will offer a well-defined set of routines for accessing the control system. The API will only have C bindings defined callable from C or C++. Calls from Fortran will be possible if the compiler supports cross-language calling or if Fortran wrappers are written for the API.

4. **graphics** - one of the main distinctive features of a modern user interface is the use of graphics with a keyboard and a mouse. The NCCS user interface will make maximum use of graphics. The main graphics programming language will be **Tk** the Motif look alike Tcl extension. Tk is much easier to program than using Athena or Motif widgets or even worse Xlib. All observing program will have a Tk graphical interface.

5. **reliability** and **maintainability** - both are obviously desirable features for any system. The user interface will ensure they are achieved by using the principle of *one task - one source*. This means every task or function of the system must be implemented only once. This also means that code will only have to be debugged once.

3 Interactive Scripting Language

A locally extended version of Tcl will be used as the interactive scripting language. This section will briefly describe the implementation and the list of new commands which will be implemented as extensions to Tcl.

3.1 Implementation

Extending the Tcl interpreter is well documented in *Part III* of J.Ousterhout's book "*Tcl and the Tk Toolkit*".

The main feature of Tcl is the *interpreter*. Each Tcl command is implemented as a *function* in C. Commands should be atomic primitives which offer the smallest level of functionality required by the user but not less. Implementing too low a level of functionality makes script writing wieldy, whereas too high a level means too many commands have to be implemented.

When implementing new commands in Tcl a choice must be made between *action-oriented* and *object-oriented*. Actions i.e. commands, are useful when implementing single actions which do not create data structures which have to be maintained. Objects are useful when it is necessary to maintain data structures over longer periods of times and refer to the contents of the data structure. Most of the NCCS commands will be action-oriented. Where it makes sense to implement objects (e.g. maybe in the case of a *scan*) it should be done because objects offer a better description of the real-world than actions.

Input and output parameters will be passed the usual way - as strings. Objects have in addition access to client data via pointers.

A single version of the extended Tcl interpreter will exist - `obsh`, which will contain all new commands. This does not exclude there being test versions which exist for debugging purposes.

3.2 Commands

The main extensions to Tcl will be in the form of *commands*. Examples of telescope specific commands can be found at almost every observatory. The following list has been partly inspired by the OBSINP, OBS and OBS_E adverbs in use at the Effelsberg 100m, Pico Veleta 30m and Mount Graham 10m telescopes.

All commands are executed immediately. There is no notion of an *active* or *passive* field which is then activated by the operator as in the case of OBS. Commands are executed synchronously and the calling program waits for a return from the control system before continuing.

1. Low Level

   Refers to commands used for low-level actions at a system level. These commands should normally only be used by system administrators.

   - **dev_putget** - execute a single device server command
     input `device-name command-specific-input-parameters`
     output `command-specific-output-parameters`
   - **sdio read** - read an sdio node
     input `node-name whole/byte/nybble/bit [byte] [nybble/bit]`
     output `value`
   - **sdio write** - write to an sdio node
     input `node-name whole/byte/nybble/bit [byte] [nybble/bit] data`
     output none

2. General

   Refers to general commands which are useful for more than one type of observation.

   - **project** - specify project name
     input `project-id`
     output none
   - **observer** - specify observer's name
     input `observer´s-name`
   - **scan** - request a scan id
     input `type duration mode`
     output `scan-id`
   - **catalog** - select a source catalog
     input `catalog-name`
     output none
   - **source** - select a source from a catalog
     input `source-name`
     output none
   - **abort** - abort present observations (send appropriate commands to STEER and all configured hardware e.g. correlator, pulsar timer etc.)
     input none
     output none
   - **subref read** - read the position of the subreflector
     input none
     output `subreflector-position`
   - **weather** - report on weather
     input none
     output `temperature pressure humidity`

3. Steer

   Refers to commands sent to the telescope drive program - STEER.

- **steer coord** - select steer coordinate system
  input `topo/hori/appa/mean/b1950/j2000/gala/ecli`
  output none
- **steer feed** - select feed system
  input `feed-name`
  output none
- **steer lat** - select position and scan rate for latitude
  input `latitude scan-rate`
  output none
- **steer long** - select position and scan rate for longitude
  input `longitude scan-rate`
- **steer euler** - select Euler angles for coordinate rotation
  input `psi phi theta`
  output none
- **steer parallax** - set lunar parallax
  input `angle`
  output none
- **steer beamoff** - select beam offset
  input `hour-angle declination`
  output none
- **steer tilt** - select subreflector tilt for feed system
  input `angle`
  output none
- **steer focus** - select subreflector focus position for feed system
  input `angle`
  output none
- **steer alarm** - switch steer alarm on/off
  input `on/off`
  output none

4. Continuum

   Commands which are used for doing continuum observations (ask Jonathan and George for input on this section).

   - **cont onoff** - do an onoff observation
   - **cont channel** - select channel to record
   - **cont dscan** - do a drift scan
   - **cont point** - do a point source measurement
   - **cont cross** - do a cross scan

5. Spectral Line

   Commands used for doing spectral line measurements (ask Mike for input on this section).

   - **sline channels** - select number of correlator channels to use
   - **sline bw** - select correlator bandwidth
   - **sline cfreq** - select centre frequency of spectrum

- **sline cvel** - select central velocity of spectrum
- **sline ave** - select averaging time for observations

6. Pulsar

   Commands used for doing pulsar observations (ask Claire for input on this section).

7. Mapping

   Commands used for mapping (ask Justin for input on this section).

4 Conclusion

Once this document has been approved implementation of the user interface and the individual commands can start.