Publ. Astron. Soc. Aust., 1996, 13, 17–21

Remote Control of Astronomical Instruments via the Internet

Michael C. B. Ashley[1], Paul W. Brooks[1] and James P. Lloyd[2]

[1]School of Physics, University of New South Wales, NSW 2052, Australia
mcba@newt.phys.unsw.edu.au
[2]University of Chicago, Amundsen–Scott South Pole Station, Antarctica

Received 1994 December 7, accepted 1995 May 29

A software package called ERIC is described that provides a framework for allowing scientific instruments to be remotely controlled via the Internet. The package has been used to control four diverse astronomical instruments, and is now being made freely available to the community. For a description of ERIC's capabilities, and how to obtain a copy, see the conclusion to this paper.

Keywords: automated software — Internet

1 Introduction

In the last five years the bandwidth of the Internet has increased to the point where it is feasible to use it for remote control of astronomical instruments. At the University of New South Wales (UNSW) we have designed and built four instruments that can be remotely operated:

- The Infrared Photometer Spectrometer (IRPS), a near-infrared single-detector system that has been operating at the US Amundsen–Scott South Pole Station since January 1994.[1] The Internet reaches the South Pole via geostationary satellites that have drifted away from a low-declination orbit (and hence are visible from the Pole).
- The Automated Patrol Telescope (APT) at Siding Spring Observatory. This is a 0·5 m Schmidt telescope equipped with a CCD (Carter et al. 1992, 1994a,b). It is connected to the Internet via a 64 kbps ISDN link shared with the other telescopes on the mountain.
- The UNSW Infrared Fabry–Perot (UNSWIRF), a tunable near-infrared filter used at the Cassegrain focus of the Anglo-Australian Telescope. In this case the control console is only 100 m away (although the instrument could be controlled from any Internet site), and the method of communication with the instrument is a single Ethernet cable. The advantage of using Ethernet for control is that a single cable replaces the multitude of cables that are traditionally used.
- A 3·7 m radio telescope (Storey & Lloyd 1993; Storey et al. 1994) on the roof of the School of Physics, UNSW, designed for undergraduate laboratory courses in radio astronomy.

We began work on a remote-control kernel for our software specifically for the IRPS, since we needed to be able to control this instrument, and receive data back from it, while it was making measurements during the Antarctic winter. Following this successful use of the software, we isolated the IRPS-specific modules, and rewrote the remainder to be independent of the type of instrument being controlled. Thus was ERIC (Extensible Remote Instrument Control) born. We have since applied the software to control of the other three instruments mentioned above.

2 Design Constraints

Since ERIC was originally written to control the IRPS, it is worth describing the design constraints of this instrument since they strongly influenced the direction of the project.

- The data-rate to the South Pole was very low (9600 baud for a few hours per day), so the communication had to be concise.
- The intermittent nature of the connection to the South Pole made it desirable for the software to be able to communicate via e-mail (in addition to the direct Internet connection, when available).
- Budgetary and time constraints restricted the choice of hardware to an IBM PC-compatible, with National Instruments controlling cards. The potential difficulties associated with writing device drivers for these cards argued against using a UNIX operating system on the PC.
- The possibility that the PC might be subjected to extreme cold (−70°C) meant that we couldn't rely on hard disk drives. We therefore purchased a solid-state disk, which forced us to keep the software (including the operating system of the computer) under 2 Mbytes. The obvious choice for the operating system was, at the time of

[1]The IRPS originates from the Anglo-Australian Observatory (Barton & Allen 1980)—at UNSW we winterised and computerised the instrument, and designed and rebuilt most of the electronics (Ashley et al. 1995)

1996PASA...13...17A

development, MS-DOS version 6·2. We also had in mind the possibility of running future low-power experiments from palm-top computers (such as the HP200LX), which again dictated the use of MS-DOS. Without these constraints, a good choice of operating system would have been a UNIX system such as Linux.

The major disadvantages of using MS-DOS are that it is not multi-tasking, does not have built-in support for Internet communication, and the 640 kbyte limit to executable program size is a nuisance. Fortunately it is possible to extend the operating system with interrupt-driven programs that act relatively independently, like background processes. To communicate over the Internet, a module that implements the Internet TCP/IP communications protocol suite was required. We chose to use the TTCP package from TurboSoft Pty Ltd. This provides TCP/IP services in a DOS environment, using the industry standard 'BSD sockets' interface, which makes it very easy to program.

The final decision was which programming language to use. We chose the Microsoft Visual C++ V1·0 compiler, but wrote only in ANSI C, without any of the C++ extensions. On the UNIX end (the computer doing the remote-controlling was envisaged as being a UNIX box) we used Perl, due to its ease of use and richness of functions to access the operating system. In fact, we had thought of writing the PC program entirely in a specially compiled version of Perl with built-in functions to handle the low-level instrument control. However, the urgency of the IRPS project prohibited exploring this interesting route.

3 Sockets

The concept of a 'socket' arose from the Berkeley Software Distribution UNIX system (for a guide to using and programming with sockets, see Comer and Stevens 1993). Basically, a socket is a logical communications 'endpoint', consisting of the computer's Internet address and a logical port number. Each computer can have a number of open sockets, and each socket can initiate a connection to another socket on another computer, which forms a channel for data communication. Data can be transferred bi-directionally, and data are guaranteed to be delivered in order, free of errors, with no duplication. The TCP/IP protocols are specifically designed to operate in environments that are hostile to data communications; they do this by checking the data and retransmitting should errors occur. Also, connections can be dynamically re-routed if an intermediate link fails, which is a common occurrence for noisy radio links. All this occurs in the background, inside the TTCP module, and requires no attention from ERIC.

A UNIX computer can treat sockets as just another type of file, to which a stream of bytes can be written/read. Under MS-DOS, life is not so easy, although various BSD socket implementations provide functions which closely mimic the UNIX behaviour.

We decided to use sockets as the basic means of communicating between the instrument-control PC and the remote UNIX computer. In our current implementation, the UNIX computer runs a daemon process that is willing to accept connections that are initiated from the PC on a particular pre-agreed (and hard-coded) port number. Once a connection has been established, the PC listens to the socket, and any characters that are received are treated just as though they were typed on the keyboard of the PC (the keyboard is still active while the socket is open). ERIC sends any requested data back along the socket—which is often more efficient than using NFS, even for large image files such as those generated by CCDs, since the transfer does not have to be synchronised with writing to disk (as it does with most implementations of NFS).

In order to provide additional feedback to the remote computer, all text written to the PC screen is also sent down a second socket interface to the remote computer.

4 The ERIC Command-set

ERIC commands are simple ASCII tokens, delimited by spaces or newlines. If a command requires one or more arguments, they are given immediately after the command. If an argument is not supplied, ERIC will prompt you with a description of what is expected.

A set of standard commands is provided with ERIC (see Table 1). These are expected to be useful regardless of the type of instrument being controlled. For examples of instrument-specific commands, see Table 2, which is taken from the software used to control the APT.

5 Log Files

A useful feature of ERIC is that all commands are time-stamped (to the nearest second) and logged in a log file on the PC. Any command arguments, and some types of data from the instrument (basically, anything that does not require too much space), are also placed in the log file. The log file is written in a compact binary form, with each command being abbreviated to a single byte. A companion program exists to decode the log file and display its contents as normal ASCII text.

A new log file is usually created each time the instrument control program is started. Facilities exist for reading the currently open log file (or any of the previous log files) from the remote computer.

Table 1. Generically useful functions built into ERIC

sdf	set default parameters	ee	enable error
rpf	read parameter file	de	disable error
wpf	write parameter file	rf	read file
dlym	delay by milliseconds	rbf	read binary file
dlys	delay by seconds	wbf	write binary file
stm	sleep till modulo	free	print fee diskspace
stt	sleep till time	intr	allow interrupts
alarm	set/clear/list alarms	comm	establish socket link
log	write a message to the log file	if	flow control
onlf	open new log file	beep	sound terminal bell
rel	read current log file	echo	display message
ts	write time-stamp to log file	spawn	execute subprocess
sel	select error level	exec	exit, and run program
ple	print last error	settime	set the PC's time
peh	print error history	help	display help message
pae	print all errors	man	access UNIX-like man pages
pei	print error information	q	quit the program
quiet	makes ERIC less verbose		

Table 2. The additional commands used to control a telescope and CCD

expose	expose the CCD	focus	adjust the telescope focus
dark	take a dark frame	calra	calibrate the RA axis
readout	readout the CCD	caldec	calibrate the Dec axis
stats	print statistics of the image	caltel	calibrate the telescope position
clear	flush the CCD	coord	give a new RA, Dec
time	set the exposure time	slew	move to the new coordinate
gain	set the CCD preamp gain	offset	offset from the current position
shift	set the CCD bit-shift	move	move an object on the CCD
speed	select lo- or hi-speed readout	centre	centre an object on the CCD
chip	specify the CCD dimensions	open	open the CCD shutter
sra	set the readout area	close	close the CCD shutter
dra	make a delta change to the area	status	display all CCD parameters
bin	set the on-chip binning	park	move the telescope to its park position

6 Error Handling

The ability to handle errors is a crucial part of instrument control software, particularly when the instrument is to be operated remotely. In common with many software systems, we decided to give each possible error a unique numeric code and an alphanumeric symbol (which hopefully is reasonably descriptive of what caused the error). In addition, each error has a one-line descriptive message (up to 80 characters long) associated with it, and possibly a paragraph or more of detailed description. The user can select which level of description is displayed when an error occurs.

When an error occurs, it is logged in the log file with a time-stamp, and is also stored in an internal history buffer. At any time the user can examine a list of all the errors that have occurred, in reverse chronological order.

To avoid the possibility that a non-serious error occurs so frequently that it disrupts the use of the instrument (simply due to the amount of verbiage on the screen, and volume of logging sent to the log file), the option exists to disable any given error message from being displayed/logged.

7 Macros and Flow Control

ERIC includes a very simple macro expansion capability that helps reduce the amount of typing involved in controlling the instrument. For example,

go = coord 12 34 45·6 −31 21 34
slew time 10 expose readout

Henceforth, typing 'go' would result in the commands to the right of the equals sign being executed, from left to right. Macros can be nested to any depth.

Loops can also be used, and nested; for example,

go2 = 5(go 3(stm 60 beep))

Finally, there is a simple 'if-then-else-fi' construct available that allows internal variables within the program to be tested against other variables or numbers, and for different commands to be executed depending on the result. This can be used, for example, to set the gain on a preamp so that the signal is not saturated, for example,

if average gt 100 then decgain else echo gain OK fi

These macro and flow-control capabilities are very rudimentary, and could be greatly improved. One possible approach would be to rewrite ERIC in Perl, which would immediately make the full richness of the Perl interpreter available.

8 Timing and Alarms

ERIC has many built-in commands dealing with timing. For example, 'dlym' will delay by a user-specified number of milliseconds (the delay is generated by a self-calibrated software timing loop), 'stm' will sleep until the current time is zero modulo a user-specified number (this is very useful for synchronising data-taking), and 'alarm' allows you to specify commands to execute at times in the future (either absolute times, relative to the current time, or repeating at a regular interval). Here is an example of using the 'alarm' feature to take a CCD image every 300 seconds:

```
doit = expose readout
alarm modulo 300 'doit'
```

9 E-mail Interface

The e-mail interface to ERIC is based on the remote computer, which is assumed to be a UNIX machine running some version of the UNIX 'sendmail' program. A user-account is established on the UNIX computer specifically for e-mail communication with the instrument. This account should contain a '.forward' file in its login directory that redirects all incoming e-mail to a Perl program (part of the ERIC software distribution). This program performs the following tasks: strips the e-mail header information from the message, checks for the presence of a password in the message, checks that the ERIC communication daemon process is running on the UNIX machine (if not, it is restarted), and then sends each line of the message to the daemon (this is done by writing the message to a specially-named file, and sending a 'kill -HUP' signal to the daemon, which then reads the file and forwards its contents to the instrument computer).

ERIC also has the ability to automatically compress and 'ftp' data files back to the remote computer.

The e-mail interface is particularly useful in the case that the instrument is not available at all times for direct Internet connection, as is the case for sites such as the South Pole. One has to be aware, though, that e-mail can arrive in a different order from that in which it was sent.

10 Making Software Changes Remotely

It is straightforward to set up the PC running ERIC in such a way that software updates can be sent to it from the remote computer, and the PC can then exit ERIC, recompile the software, and re-run it. To do this, the PC's AUTOEXEC.BAT file should do the following:

1 Recompile the software if it has been altered.
2 Run ERIC.
3 Reboot the PC when ERIC exits.

The software modifications can be written to the PC using ERIC's 'wbf' (write binary file) command. Then, exiting ERIC will cause the PC to reboot, and the software to be recompiled and executed. Alternatively, if the data-rate available is very low, it is possible to send all the changes as patches (i.e., the difference between the old source code and the new), compressed with a program such as 'gzip', and then to use ERIC's 'exec' command to exit ERIC and run a batch file containing commands to decompress the patches, apply them (using the public domain 'patch' program), and then reboot the PC to recompile and execute the new program.

This latter technique has been successfully used to update the software running IRPS at the South Pole. Needless to say, one has to be absolutely confident that the new source code will recompile correctly.

11 Caveats

ERIC is a fairly simple program (roughly 3500 lines) that is useful for controlling instruments. It would need more work before being considered for control of a major facility such as a large telescope, but could be used immediately for controlling a subsystem such as a weather station or a single instrument.

12 Conclusion

To summarise, ERIC is a remote-control kernel written in Microsoft Visual C++, designed to be incorporated into an instrument-control program. It runs on an IBM-compatible PC, for communication with a UNIX host. ERIC provides the following functionality:

- The instrument is controlled by a stream of ASCII commands, which can originate from the keyboard of the PC, from a file that is local to the PC, or from a 'socket' connected to a remote computer.
- Whatever is printed on the PC screen can also be sent to the remote computer for display.
- A set of standard commands (see Table 1) is available, regardless of the instrument being controlled.
- All commands sent to the instrument are logged in a compact form to a file on the PC, with time-stamps to the nearest second.
- ERIC incorporates a framework for handling error conditions.
- A macro-expansion capability is included, together with a simple if-then-else-fi construct and the ability to execute loops.

- Any file on the PC can be read or written, and the PC can be remotely rebooted. These features can be used to update and recompile the control program itself.
- On-line help is available through a UNIX-like 'man' command.
- Some degree of security against unauthorised access is given by hard-coding the IP number of the controlling computer, and by requiring passwords to be sent when initiating communication.
- Commands can be sent to ERIC via e-mail, and ERIC can respond similarly. Currently, this is implemented by software running on the remote computer, although it could be transferred to the PC.

ERIC is available on the WorldWide Web at URL `http://www.phys.unsw.edu.au/~mcba/eric.html`. TTCP is available from TurboSoft Pty Ltd, 579 Harris Street, Ultimo 2007, NSW (e-mail salesturbosoft.com.au).

Ashley, M. C. B., Burton, M. G., Lloyd, J. P., & Storey, J. W. V. 1995, SPIE, 2552, 33
Barton, J. R., & Allen, D. A. 1980, PASP, 92, 368
Carter, B. D., Ashley, M. C. B., Sun, Y-S, & Storey, J. W. V. 1992, PASA, 10, 74
Carter, B. D., Bembrick, C. S., Ashley, M. C. B., & Mitchell, P. 1994a, Exper. Astron., in press
Carter, B. D., Ashley, M. C. B., Bembrick, C. S., Brooks, P. W., Mitchell, P., & Storey, J. W. V. 1994b, in IAU Colloquium 148, ed. R. D. Cannon & B. Hidayat (Dordrecht: Kluwer), in press
Comer, D. E., & Stevens, D. L. 1993, Internetworking with TCP/IP, Vol. III, BSD Socket Version, (New York: Prentice Hall)
Storey, J. W. V., Lloyd, J. P. 1993, PASA, 10, 225
Storey, J. W. V., Ashley, M. C. B., Naray, M., & Lloyd, J. P. 1994, Am. J. Phys., in press