Low-Cost Beamline Control System

Takashi Kosuge and Yoshinori Uchida, KEK, Tsukuba, Japan

Abstract

The Photon Factory has many different kinds of beamline control systems, which are mostly based on PCs. Recently, many users have requested that a standard control system be made. The new system must be made at minimal cost and be easy to remodel.

Generally, the beamline optics comprising of a monochromator and mirrors have been controlled by pulse-motor controllers using GPIB at the Photon Factory. Since high-speed control was not requested, it was decided that an RS232C-GPIB converter would be used.

Our system has become hardware-independent. By writing the application in Perl, and running it on FreeBSD, the cost has been significantly reduced. Perl is a very powerful, and, from our experience, a very easy-to-use language. Staff with limited programming knowledge can learn to program in Perl at a low training cost. Perl is a popular, powerful CGI language used in the Web world. Through the use of Perl in the application of a beamline control system, the language's full potential has been realized.

This paper discusses the development of a low-cost standard system at the Photon Factory and stresses that in comparison to other languages Perl is far superior.

1 INTRODUCTION

At present, the Photon Factory has 21 main beamlines in the PF 2.5 GeV ring and 3 beamlines in the AR. Also, each main beamline has 2 to 4 branch beamlines for experiments. Generally, the beamline components (monochromator and mirrors) are controlled by pulse-motor controllers at the Photon Factory. Currently, there are various PC-based control systems (Windows95, DOS etc.) which are running with different software. This disparity has caused confusion. Of course, the staff has been demanding a standard beamline control system.

Coincidentally, the beamline control system was due for replacement at BL-16A. Therefore, we tried to develop a new standard beamline control system for BL-16A.

2 HARDWARE AND OS

This time, since we had to drive 27 pulse-motors, we connected 2 pulse-motor controllers to our system.

The layout of the system is shown Figure 1.

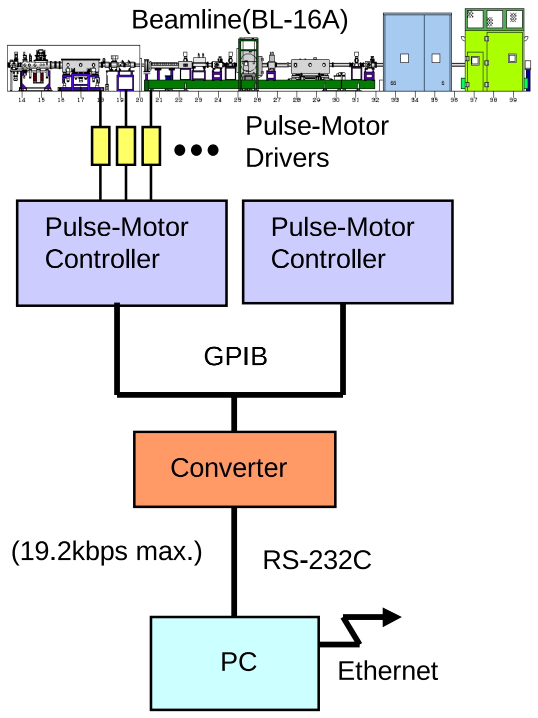

Figure1: Hardware layout

2.1 *Pulse-motor controller*

The pulse-motor controller (Tsuji-Denshi PM16C-02N) can drive 16 pulse-motors. It has a GPIB port on its rear panel, and switches and displays on its front panel. This controller is popular among the Photon Factory staff. Since pulse-motors can be controlled and their movement checked locally without a control system.

This time we used 2 pulse-pulse motor controllers of this type. They are connected to a converter with GPIB.

2.2 *GPIB-RS232C converter*

The converter (Network-Supply GPNET model-20+) has a RS232C port with a maximum data flow speed of 19.2 kbps.

The beamline control system does not require high-speed control. Sufficient speed can be attained even using this converter. Also, the system will not be hardware dependent by using RS232C.

2.3 *PC and FreeBSD*

We used a low-price PC. It has a 200MHz Intel Pentium processor with MMX technology CPU and 64MB RAM. We chose FreeBSD for the operating system. FreeBSD is very stable and we can use it without any cost. When designing a low-price system, using a PC and FreeBSD is a good option.

3 SOFTWARE DEVELOPMENT

This time we needed to receive commands from other software and computers. We made two separated programs, the server program and the client program, in developing the software (see Figure 2). They are connected using TCP/IP sockets.

The server program does low-level control and translates commands from the client program and pulse-motor controller. It can receive commands from other software. Also, the client program prepares the user interface and export commands for tuning the beamline optics.

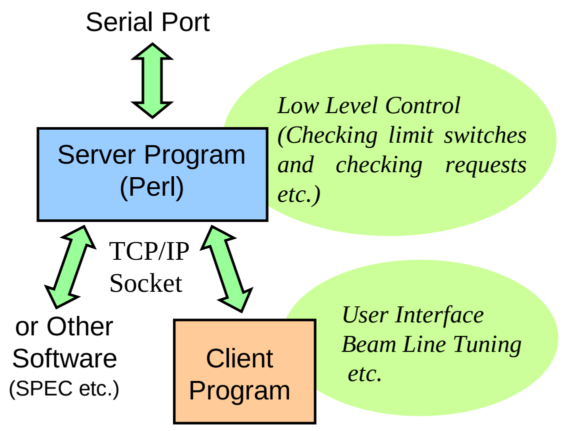
Serial Port
Server Program (Perl)
Low Level Control (Checking limit switches and checking requests etc.)
TCP/IP Socket
or Other Software (SPEC etc.)
Client Program
User Interface Beam Line Tuning etc.

Figure2: Software development

3.1 Developing the server program

When developing the server program, we made a standard command set. The commands must be easy to understand for users. Also, the output message should be easy. We wrote the server program in Perl[1] [2] [3].

When the server program receives a command from the client program, the command is translated into a special command for the pulse-motor controller. The server program receives output messages from the pulse-motor controller, and then translates the special output message to a common message. Then, the server program sends it back to the client program.

3.2 Developing the client program with other staff

We had to develop a client program with another staff member who was a beamline specialist. However, his knowledge about programming was very modest. We needed to use an easy language for this situation. Also, we found that Perl is very easy to learn. We expected that Perl would help us when developing client programs with him.

First, we lent a primer to him, and he studied it for 3 days. Then, he took one week to make simple subroutines. During the same time, we made the main program. Finally, we finished the first version of the client program by merging the subroutines and the main program.

3.3 Macro mechanism

When making a flexible program, it is necessary to have functions similar to macro commands or scripts. The “eval” function of Perl is very effective in this situation. We produced a macro mechanism with the “eval” function.

When the program receives a request for a macro command, it loads a macro command file written in Perl, and delivers the strings to “eval” as arguments. Subsequently, they are executed by “eval”.

Users can use the very powerful macro commands of this program in Perl by using the “eval” function.

4 AVAILABILITY OF PERL

As stated above, we could develop the programs easily with Perl. Also, we found that Perl is effective on small control systems.

4.1 Transferring strings

List 1 is an example in Visual Basic. Here, we want to get a command and a parameter from strings, and set them to variables, “cmnd” and “prmt”. If strings in “cmnd” is “GetData” then change it to “Get”, and increase “prmt”. We must ignore whether the letters are capitalized or not here.

List 1: Example in Visual Basic (a=”GetData 2”)

```
 1: b = InStr(a, " ")
 2: If b > 1 Then
 3:     cmnd = Left(a, b - 1)
 4:     prmt = Mid(a, b)
 5:     prmt = Trim(prmt)
 6: Else
 7:     cmnd = Trim(a)
 8:     prmt = ""
 9: End If
10: If UCase(cmnd) = "GETDATA" Then
11:     cmnd = "Get"
12:     prmt = prmt + 1
13: End If
```

List 2: Example in Perl ($_=”GetData 2”)

```
($cmnd, $prmt) = split;
if($cmnd=~s/^GetData$/Get/i){$prmt++}
```

First, it finds the “space” character in the string in the first line, and sets variables with information of locations in line numbers 3 to 5. Then, it changes strings into capitals for ignoring the case in line number 10.

What does the same program look like in Perl? You can find that Perl requires only 2 lines on List 2.

In the first line, the string is divided into a command and a parameter with a “split” function, and they are set to variables. In the second line, the strings in “$cmnd” are changed to “Get” and increase “$prmt”, if the string is “GetData”.

This example shows that we can change strings very easily with Perl.

4.2 *Platforms*

The significant advantage is that Perl runs on many platforms. Table 1 shows that Perl has more platforms for running compared to other languages. Of course, occasionally we have to change a few codes for differences in the operating system, but we can mostly use the same Perl codes on many platforms.

Table 1: Programming languages and platforms (Jan. 1999)

	JDK 1.1.x	**Visual Basic**	**Perl 5.0**
Solaris	Yes	No	Yes
Windows 95/NT	Yes	Yes	Yes
Macintosh	No	No	Yes
Linux	Yes	No	Yes
FreeBSD	Yes	No	Yes

4.2 *GUI*

Finally, we have a problem which must be solved. It's a GUI. We are planning to use Perl/Tk. It consists of a portable tool kit and a Tk module. We can use this tool kit by writing “use Tk” at the top of the program (List 3). Then, if we execute the program, the GUI window will appear on the screen (Figure 3). If we push the button, the subroutine “start” or “stop” is be executed. We hope that we will be able to obtain a good GUI easily with Perl/Tk.

List 3: Using Perl/Tk

```
 use Tk;
$top=MainWindow->new();
$top->title("Test");
$label1 =$top->Label( text    => '=STOP=');
$button1=$top->Button(text    => 'Start’,
                              command => \&start);
$button2=$top->Button(text    => 'Stop',
                              command => \&stop);

$label1->pack();$button1->pack();
$button2->pack();

MainLoop();
sub start{$label1->configure(text => '=RUN=');}
sub stop{$label1->configure(text => '=STOP=');}
```

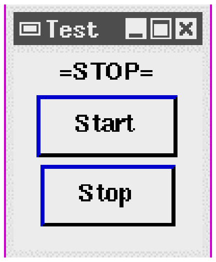

Figure 3: GUI with Perl/Tk

5 SUMMARY

We made a low-cost beamline control system with PCs and Perl.

This system has satisfied the requirements, “It must be low cost”, “It must be easy to remodel.” and “Anyone must be able to maintain the system”.

Now we have finished off-line tests and the system will be in use at BL-16A from this February.

REFERENCES

[1] Randal L. Schwarts, and Tom Christiansen, “Learning Perl Second Edition”, 1997, O’Reilly & Associates, Inc.
[2] Larry Wall, Tom Christiansen, and Randal L. Schwarts with Stephen Potter, “Programming Perl Second Edition”, 1996, O’Reilly & Associates, Inc.
[3] Sriram Srinivasan, “Advanced Perl Programming”, 1997, O’Reilly & Associates, Inc.