Stanford Exploration Project, Report 65, January 13, 1998, pages 293–311

An interactive processing environment

Steve Cole[1]

keywords: *algorithm, processing, sequence*

ABSTRACT

I have written an interactive software package that allows all SEP batch-oriented software to be used in an interactive manner, with no programming required of the end user. One can "build" a processing sequence, interactively modify the parameters for the various process, and see the result of each modification displayed on the workstation screen. The processing sequence and all associated parameters can be saved to a file so that the user can exit, then return at any time and pick up where he left off. With this environment it is much easier to experiment with different parameter values than in traditional batch processing. My software is written using XView, a freely-distributed X Windows toolkit developed by Sun Microsystems. Thus, like any X application, it can be ported to many machines.

INTRODUCTION

Interactive programs have obvious advantages. The ability to see the results of a job immediately, and to be able to modify the parameters and try it again are just a couple. At SEP we have the hardware necessary to do a lot of interactive work, but only a small fraction of our software is interactive. Interactive programs are very time-consuming to write, and a programmer needs to know a lot about the particular software environment he is using (Sunview, X Windows, InterViews, etc.) before he can even get started. The result is that we have a few very nice but highly specialized interactive programs. For most of our work, we run in a "batch" mode. This means editing some file containing the names of programs and their parameters, launching a job, and waiting for the results. Thanks to SEP's vplot graphics system (see Cole and Dellinger, 1989), batch processing can be reasonably efficient. I can launch a job, see the results on my workstation screen, go back and change some parameters in a file, and try it again. When I'm happy with the results, I can send them off to a printer. But the process is still rather cumbersome if I want to experiment with different programs or parameters very much.

[1]**email:** not available

IPE

Recently I realized that with a modest amount of effort by one person (me), we could make all our existing batch software interactive. We already know how to display results in real-time on workstation screens using vplot. What is missing is a way to interactively modify program parameters, and see the results change as we changed parameter values. I decided to provide the missing piece, a program that would let the user modify parameters interactively, then call existing software to run the job and display the results. I've called this intermediary **IPE**, which is short for Interactive Processing Environment.

Jean-Claude Dulac worked on this problem while he was at SEP, and wrote an interactive user interface manager called *SepView* (Dulac, 1988). IPE is more limited in scope than SepView, but I think its simplicity is its strength. It is easier to use or get inside of than SepView. No programming is required to use IPE or add new programs to its repertoire. Still, SepView has many nice features that IPE lacks. After I've explained IPE and how it works, I'll contrast it with SepView more carefully, and see what features of SepView can and should be incorporated into IPE in the future.

I've said that IPE is interactive, but I should qualify that statement a little. In a "truly" interactive program, if I change a parameter (for instance, if I modify the value in a text field or move a slider), the display is instantly updated where necessary to reflect the change. IPE is a bit more awkward. After modifying the slider or text field, I have to go and press the DOIT button Then my entire plot, not just the necessary pieces, is erased and redrawn. For this reason I would say that IPE isn't "truly" interactive. But it comes very close, for many different programs, at essentially no cost.

I wrote a first version of IPE a couple months ago, using Sunview, an interactive package for Sun workstations that has been around for several years. Early this year, along with a new release of X Windows from MIT came the XView toolkit, written by Sun and in many ways similar to Sunview. I converted IPE to XView, and now we have the advantage of being able to run it on all our machines – Convex, Sun, and DEC – rather than just one. Programmers familiar with Sunview will find the transition to XView relatively painless. Sun provides utilities that will do much (but not all) of the work to convert source code from Sunview to XView.

Using IPE

IPE is very simple to use — you simply type **Ipe** under X Windows. This brings up a small panel, shown in Figure 1.

The object on the right side of the panel is a "scrollable list" item. Its entries are the sequence of programs that IPE will execute. The default sequence is as shown in Figure 1. "Input" means read in a dataset. "Taplot" is an SEP program that converts a floating point dataset to single bytes. "Ta2vplot" is a program that takes byte data and produces a raster plot of the data in vplot format. Finally "Xvpen" is a program that displays the vplot result under X Windows. (It is a new vplot output filter, also written by me using XView.) IPE's power lies partly in the fact that this sequence can be modified. Programs can be added, replaced, or deleted using the buttons on the left panel.

Notice that one of the buttons in the scrollable list has its name boxed. (this is "Taplot" in the example of Figure 1). The box indicates that this is the "current" program. When you select a program from this list, it is boxed and a pop-up panel appears with the parameters for the current program. In Figure 2 I show the popup panel for the program Taplot. Parameters can be implemented as any of the following three types, all shown in Figure 2:

- text items. The user types text into a field of pre-defined length.
- slider items. A slider is used to set a parameter value from a given range.
- multiple-choice items. The user clicks on a box corresponding to one of several choices.

Figure 2 highlights one other feature of IPE not yet mentioned. On the parameter panel for Taplot are buttons named **Load from parfile** and **Save to parfile** as well as a text field for the specification of a parameter file name. These buttons appear on every parameter panel. SEP programs receive parameters either as command-line arguments or as values in a parameter file. IPE makes use of the latter method. For each program, IPE constructs a parameter file, and writes all the values entered on the panel to this file. This file is passed to the program. In addition to saving to a file, IPE can also load parameters from a file. Thus you can save a particularly useful set of parameters in a file so that you use them again later, without having to re-enter them. The "Save to parfile" buttons loads the parameters in, adjusting the sliders and other items automatically.

The left side of the main panel in Figure 1 contains buttons that perform various tasks in IPE. I've already alluded to the buttons **Replace**, **Insert Before**, **Insert After**, and **Delete**. These buttons add or remove programs from the processing sequence. The insert functions act relative to the "current" program, which is indicated by the box, while Replace and Delete act on the current program.

Use of the Replace, Insert Before, and Insert After buttons may initially be confusing. Buttons typically perform an action when the left mouse button is pressed while the cursor is pointed at them. These buttons behave differently. If I position the cursor on any of these buttons and hold down the *right* mouse button, a menu of available programs comes up, as shown in Figure 3. I select from the menu the program I want to add to the processing sequence. The presence of the pull-down menus attached to these buttons in indicated in XView by the arrow next to the button name.

The button **Save/Load** is used to save or load in an entire processing sequence, so that you can exit an IPE session, then come back later and pick up where you left off. For example, suppose I use IPE to filter some data. I build the processing sequence, set all the parameters, make some tests, and then leave IPE. I'd like to avoid having to do all that work again when I re-enter IPE. Above I showed how to save the parameters for any one program to a file. The **Save/Load** button lets you save the entire processing sequence – the sequence of programs, and the name of the parameter file associated with each – to another file. If you exit IPE and come back later, you can use Save/Load to quickly get back to the same point in your work.

The **DOIT** button causes the processing sequence that you have built to actually be run. Only when **DOIT** is pressed will results appear.

Finally, the **Quit** button exits from IPE.

Behind the scenes

As mentioned earlier, IPE stores the parameters for each program in a file. The format of this parameter file is quite simple; there are many lines of the form "parameter=value", where "parameter" is the parameter name (also the label on the IPE panel), and "value" is the value chosen by the user. Once this is done, IPE uses pipes to construct a single Unix command corresponding to the processing sequence. For the simple "default" processing sequence, the command would look like this:

```
<file Taplot par=parfile.taplot | Ta2vplot par=parfile.ta2vplot
        | Xvpen par=parfile.xvpen
```

This command reads in the file "file", runs the program "Taplot" on it using the parameters located in "parfile.taplot", passes the result to the program "Ta2vplot", and so on. Finally the result goes to Xvpen, which displays the results on the workstation screen. I could have manually edited these parameter files, then executed this same command myself. But to try out some new parameters, I would have to edit the necessary parameter files and run the job over again. With IPE the result appears on the workstation screen, and I am still editing the parameters on their panels. I can immediately try some new values and see how they change the results.

Components of IPE

Here are the software components of IPE:

- A program *Ipe* and some associated subroutines that handle the user interaction — adding, deleting, or changing programs, assembling the processing sequence, and launching the job. Ipe.c and its subroutines are roughly 700 lines of C code, with calls to the XView toolkit. No user needs to know anything about this code, unless he or she wants to modify the inner workings of IPE.

- A program *Ipemake* that reads in a simple ASCII description of the parameters for a given program, and generates the C language subroutine that IPE uses to know how to display the program's parameters on a panel. Ipemake.c is also written in C, is about 500 lines long, and again the end user never needs to touch it or know how it works. Users don't even need to use Ipemake unless they want to add a program to IPE's library.

- For each program available in IPE, an ASCII description of the program's parameters. The format is described below. A user would need to create this file to add a new program, but doesn't need to bother with this to use the programs already available.

Ipemake

The most interesting challenge in writing IPE, from a programming standpoint at least, was the program Ipemake, which takes a simple ASCII description of a program's parameters, produces the XView code necessary to display a panel containing the parameters, and convert those panel objects into values in a parameter file. To illustrate the action of Ipemake, suppose I have simple program "Myprog" that has just one parameter named "param". Here is the ASCII parameter description file:

And here is part of the actual code produced by Ipemake:

This part of the code creates the panel containing Myprog's parameters. Ipemake also generates the subroutine that builds the parameter file for Myprog:

And the subroutine that constructs the part of the final command string pertaining to Myprog:

In my first pass at installing IPE, I've supported twelve different programs. From a total of 125 lines of parameter description files, Ipemake has generated a total of 2205 lines of C code, as well as 120 lines of include files. It's clear that without Ipemake, I wouldn't have had the patience to write the code to support even these twelve programs. With Ipemake, I did it in a couple of hours.

Portability

As I mentioned earlier, IPE uses the XView toolkit developed by Sun for X Windows. Since X and XView are distributed freely through MIT and have been ported to many machines, most anyone interested could take advantage of IPE. While all the examples I will show here use SEP data processing and graphics programs, I've taken care to avoid any SEP-dependence in IPE itself. Any user who has programs to process data and display results under X Windows could use IPE. The only potential hitch would be the mechanism by which parameters are passed to a program, but the amount of code that would have to be modified to handle another method would be quite small.

IPE and SepView

As I said earlier, Jean-Claude Dulac's SepView user interface manager is similar in concept to IPE but far more general. Here are a few of the important differences between the two:

- SepView allows for a non-linear processing sequence. For instance, the user can input or output auxiliary datasets anywhere in the sequence, or split the sequence into two branches. IPE could handle the auxiliary datasets with small modifications, but branching would require a great deal of effort.
- In SepView the descriptions of program parameters are interpreted, while in IPE they are compiled into the Ipe executable. This means that in SepView the descriptions can be modified "on the fly" while in Ipe you would have to re-run Ipemake, and then re-compile everything.
- SepView allows the parameter descriptions to include some control logic. Parameters can be checked, the user can be prompted to supply missing or badly-specified parameters, and so on. IPE takes whatever the user inputs and passes it along to the program as is. Any error messages go to the screen, just as they would for batch execution.

Obviously, these differences all point in SepView's favor. But as one can imagine, SepView's complexity makes it more difficult to use. Constructing the parameter descriptions for even one program is a formidable task. IPE uses the very simple description scheme described above, and it is realistic to add a new program to IPE in the space of fifteen minutes.

My goal is to first make IPE a stable, useful product, supporting a large number of programs, and then gradually add features to it to make it more powerful. Undoubtedly some of the additions will be patterned after features of SepView, which I think is a really nice piece of work. I am particularly interested in allowing for a non-linear processing sequence. I am also interested in checking input parameters and ensuring that the processing sequence the user constructs is valid.

EXAMPLE

Figure 4 shows the workstation screen during a typical IPE session. I've plotted a dataset as a raster image, then modified the processing sequence extensively and replotted part of the gather using a wiggle-trace display program.

The various panels generated by IPE and the windows created by the display program Xvpen can all be moved and resized, so the session can be customized. Since multiple Xvpen windows can be open simultaneously, I can compare the results of different parameter choices or even different processing sequences. At any time, I can change the display program from Xvpen, which sends it result to the screen, to another vplot filter that sends a hardcopy to a printer. Using the Load/Save button, I can save all the attributes of this session, quit IPE, come back later and pick up where I left off.

THE ELECTRONIC TEXTBOOK

Jon Claerbout habitually saves the programs and parameter values used to create figures in his books, waiting for the day when he can distribute an "electronic textbook" where readers can experiment with his programs and see the effect of different parameters. I hope that IPE will be the vehicle for the first generation of such a utility. With IPE, we can load in an initial processing sequence and values for all the parameters in each program. Users can then make some changes and see what happens. To make an effective tool, we might need a way to hide irrelevant parameters, so that the student can focus on those that are relevant. This shouldn't be hard to do. In the near future I will work on supporting some of Jon's programs in IPE and constructing datafiles for different examples.

While IPE is far enough along that it is useful, the job is by no means finished. I've already mentioned some features I would like to add or modifications I would like to make. I think that one of the most important issues to be resolved is creating the parameter database that Ipemake converts to subroutines that are linked into Ipe. It is easy to add one SEP program, or even ten or twenty programs, using the method I have described. But it would be a formidable task to support *all* SEP programs in IPE, and then we would have to worry about maintaining the parameter database as programs are added or modified. We need to tie the generation of the parameter file to program development. Rick Ottolini has suggested a way to do this. At SEP, we have a utility called "Saw" (the Stanford Auto Writer) that simplifies the task of writing programs. Programmers don't have to worry about calling subroutines to get parameter values from the user; they supply a simple ASCII description of the parameters, and Saw turns this description into the necessary subroutine calls. We need to tie IPE into the this same framework, so that the information needed by IPE is generated automatically, with no extra effort required of the programmer, and no possibility that the IPE database will become out of date.

CONCLUSIONS

IPE is a useful framework for interactive data processing that allows all SEP batch software to be used interactively. No programming is required of users, even those who want to add new programs to IPE's database. An IPE session lets the user interactively select programs to use, and see the results from different tests simultaneously. Processing parameters can be saved at any point to allow a result to be re-created.

ACKNOWLEDGMENTS

Dave Nichols shared his C expertise at several points and as a result IPE is now written in a much more flexible, extensible manner that my first version. I appreciate Dave's help very much. I would also like to thank Andrew Gunstensen of MIT, who shared a nice interactive plot program called *Tplot* that he had written. Tplot, which is a truly interactive program, gave me the idea to make our batch programs "somewhat" interactive, which led to IPE.

REFERENCES

Cole, S., and Dellinger, J., 1989, Vplot: SEP's plot language: SEP–**60**, 349-390.

Dulac, J.-C., 1988, A user interface manager: SepView: SEP–**59**, 223-240.