Proposed Open Source Data Processing System

J. Higginbotham, Chevron Energy Technology Company
January, 2006

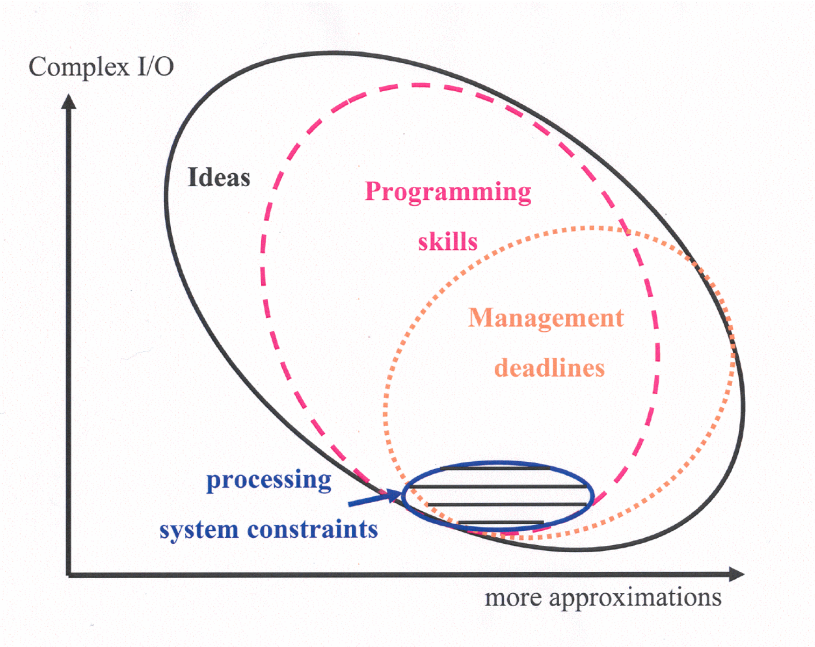

Figure 1: Current constraints on implementation of ideas in production processing systems

Management wants tools developed in the production processing system since they are then immediately available for application. It's hard to argue with this but the fact is that there are constraints placed on research and development by typical production systems. Researchers look at the data processing environment in which they are constrained to work and constrain the ideas they develop to fit that environment ... or create their own environment. This means that many good ideas are discarded up front. The alternative is for the research and development person to write a stand along tool or to spend large amounts of time learning the details of computer science in order to build an environment that minimizes constraints. This is where we are today in the industry. Research geophysicists spend most of their time doing computer science instead of geophysics. If you go from company to company you find that the most complex data processing operations are implemented as either stand alone programs or are part of an originally unofficial, and probably still unofficial, system developed by research people. This is true for both oil companies and contracting companies and is strong evidence that the current situation is as shown in figure 1.

Since the early 1990's there have been great improvements in user interfaces associated with data processing but I've seen little change in the developer interfaces, at least for commercial systems. It would be a mistake to give up these improvements at the user end in favor of changes for the developer. However if the users are to see higher levels of technology available at their friendly end of the processing system then the bottleneck at the development end must be opened. You can easily imagine having systems that fall into three categories: 1. those that serve developers well and users poorly, 2. those that serve users well but developers poorly, and 3. those that serve both users and developers well. When you read my viewpoint below you should remember that while I'd like to put forward something in category 3, as a developer I may unintentionally neglect something important to the user. I suggest though that a weak user interface on an operation in a familiar system is better than a weak user interface on a stand alone program. In any case you should judge what I put forward, take what is useful and combine it with other useful ideas to converge on the best for both users and developers - all ultimately in the interest of users whose work funds us all.

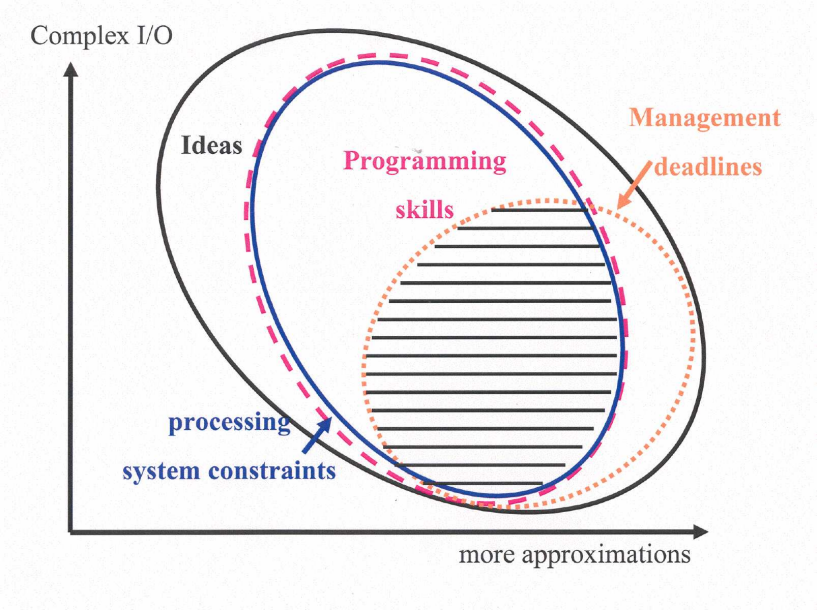

Figure 2: Alleviating constraints on implementation of ideas in an open processing system

The guiding philosophy behind building a processing system that stimulates algorithm development should be the *minimizing of constraints placed on algorithm development.* This should apply to both programs that require interaction with the user throughout the program's activity, interactive processing, and to programs that require only that the user choose a set of parameters that control the action of the program before the program is actually run to process data, batch processing. I am concerned here with batch processing because it is in batch processing that the most complex seismic input-output structures arise.

From the point of view of the developer the seismic processing system's task is to make available data and processing parameter information and to provide a means for passing processed data back to the system. This is a simple task that should not require the developer to become an expert systems programmer. This suggests that the processing system should provide the option of working only with seismic files and parameter argument files that are isolated from the complexities of the system. The benefit of this approach is that the data processing programs running in such a system are not tightly coupled with the processing system. *This will allow the processing system itself to be upgraded or possibly even replaced with minimum or no impact on the data processing programs.* In fact if this interface between the processing system and the data processing applications is sufficiently standardized then *the processing system itself, without the seismic operations, could become a commodity.* Taking this step could require some level of standardization of the seismic data format - I have something in mind of course but there isn't room here.

If I were to stop here I'd have given you some general ideas but not enough specifics to convince you that the ideas were realizable. So what I'll do next is provide you with a rough specification for a system that works. Instead of being formal I'll just draw a diagram and run through what happens when data is processed. It's a prescription kind of presentation but hopefully you'll see how it connects with the ideas mentioned above. And you'll see in the end that a more formal generic specification is available - generic because I'm not pushing my system but rather my ideas.

Consider the system diagram shown in figure 3. The user runs a graphical user interface (GUI) that reads a parameter file ***(P-file)***for every seismic operation that is implemented. The parameter information in the p-file provides the GUI with a list of parameters, how they should be grouped, a one line definition of each, a pointer to more complete documentation, and formatting information that the GUI uses to present the operation and its parameters to the user.

Based on the user's selection of seismic operations to run and their associated parameter information,

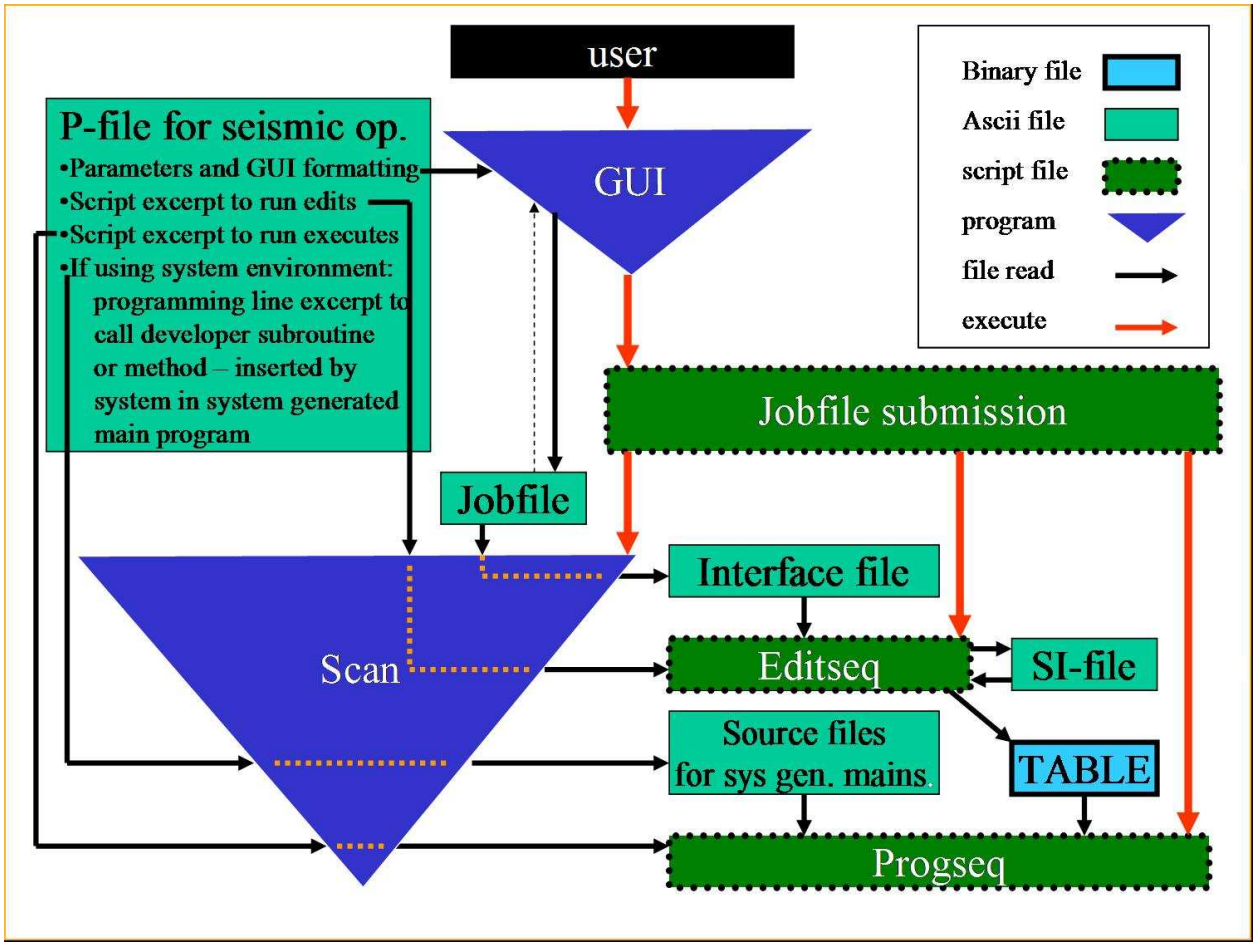

Figure 3: System diagram

the GUI builds an ascii file called a ***Jobfile***. The Jobfile is in a format that can be easily read by the user and can be inserted into reports to document the operations applied to data. The GUI then runs a shell script called the ***Jobfile submission*** script.

The Jobfile submission script runs a program named Scan, a script named Editseq and a script named Progseq. These three run in this order because latter depend on the former.

The ***Scan*** program reads the Jobfile and determines which seismic operations are to execute. As it reads the Jobfile it converts the file to another ascii file, called the ***Interface file***, that is formatted in such a way that its information can be easily read by a computer program without parsing.

By default the seismic operations act on a seismic file named ONLINE. However every seismic operation has access to all seismic files that exist on the file system.

When the scan program determines that a particular operation is to be executed it reads the p-file for that operation and finds the shell script excerpts that are to be placed in the ***Editseq*** shell script. This shell script excerpt will run a stand alone program, called the *edit* for a particular operation. The scan program will have inserted a command into the Editseq script, ahead of the edit program, that will make the portion of the Interface file, corresponding to the particular operation, available to the edit in a file named ***PrmArgs***. The edit will be able to get parameter arguments from the PrmArgs file and check them for errors or process them as required. The processed information will then be written to a binary file named ***TABLE***. The location at which the parameters are to be written is read from the first word of the TABLE

file and this word is updated after the information is written.

When the edit is running it will need information about the seismic files to be processed in order to fully check parameters for accuracy. For those files that exist when the job is submitted this information can be read directly from the file. For files that will be created during the execution of the job this information will not exist. When an edit runs for an operation that will create a new seismic file it must place the fundamental structure information of that file in the Seismic Information file ***(SI-file)***. This is where subsequent edits will look for the information needed for files that don't yet exist.

The Editseq script runs to completion and executes the edit corresponding to each seismic operation to be run before any seismic processing starts. As the edit for a seismic operation becomes mature the probability that that operation will run properly becomes high if the Editseq script completes without errors. One of the last things done by the Editseq script is to reset the pointer in the TABLE file to the location of information for the first seismic operation.

When the scan program identifies a seismic operation to be executed it will also read information from the p-file that controls how the ***Progseq*** script will execute the operation.

The operations most isolated from the system, the ones I'm pushing hard here, are executable programs written by the developer. If this is the type of operation to be run then the scan program will read a script excerpt from the p-file that controls how this executable is to be run and insert it into the Progseq script file. A seismic operation run by Progseq is called an *execute*. The p-file, edit program, and execute program come as a triplet associated with each seismic operation. These are what the developer writes!

The other type of operation that Progseq can run is a seismic operation that is embedded in an *environment created by the system* that serves gathers of traces for processing and takes away gathers after processing. This environment is created inside a stand alone program that is built by the scan program. In a long sequence of operations there may be several of these. Most of this program already exists in a library. Scan only writes a small main program that wraps around calls to the developer routines. Excerpts of code are read by scan from the p-file and inserted into the main program to accomplish the call of the subroutine or method. The scan program will make an effort to group together as many of these types of operations as possible under a single main to minimize read and write to disk. In cases where it is appropriate and where the user requests, the scan program can select a main program that supports automatic parallelization over many slave nodes. Since the system builds the main program, this option for inserting seismic operations within the system places some constraints on the developer! So I'm not selling this one hard - but it is so powerful that it should exist.

Since the seismic operations execute as a sequence of stand alone executables it is possible to insert operating system commands between these operations to link, copy, and move seismic files. Corresponding to each operating system command a program should run at edit time to update the SI-file when seismic data is being moved around, copied or deleted - a job for the scan program.

A number of utility programs should exist: 1) one for reading and making minor modifications to seismic files (the sample interval frequently overflows in SEGY format), 2) one for logically mapping several seismic files to one seismic file, 3) one that will read header information in a seismic file and generate a database, ...

I run a realization of this system almost every day I work. The system has been around since 1984. This year I ran algorithms I wrote in 1984 in the pursuit of development of anisotropic imaging of shot gathers. By publishing the structure of the: Interface file, the SI-file, the TABLE file and the seismic format, this system would become fully accessible to any developer who would be free to develop a seismic operation in any desired language. The system has other desirable features that I don't have room to describe here. I hope that sharing my experience will help promote an open constraint-free system for developers so that users and managers can have what they want.

If you find this system interesting then you may want a more complete generic specification as well as information about the realization I use. I hope to publish these in the future.