Integrated software framework for processing of geophysical data

Glenn Chubak and Igor Morozov

University of Saskatchewan, 114 Science Place, Saskatoon, SK, Canada, S7N 5E2

Fax: +1-306-966-8593. E-mail address: gdc178@mail.usask.ca

Abstract—We present an integrated software framework for geophysical data processing, based on an updated seismic data processing program package originally developed at the Program for Crustal Studies at the University of Wyoming. Unlike other systems, this processing monitor supports structured multicomponent seismic data streams, multidimensional data traces, and employs a unique backpropagation execution logic. This results in an unusual flexibility of processing, allowing the system to handle nearly any geophysical data. A modern and feature-rich Graphical User Interface (GUI) was developed for the system, allowing editing and submission of processing flows and interaction with running jobs. Multiple jobs can be executed in a distributed multi-processor networks and controlled from the same GUI. Jobs, in their turn, can also be parallelized to take advantage of parallel processing environments such as local area networks and Beowulf clusters.

***Key Words*:** Seismic, Gravity, Processing, Multicomponent, Graphical User Interface, Parallel computations, Cluster, Beowulf

1. Introduction

Analysis of geophysical data nearly always involves application of sophisticated and multi-stage processing and inversion. With volumes of data and resolution of the datasets exploding in the recent years in nearly every field, the demand for computer packages facilitating handling, processing, analysis, and interpretation of large and complex datasets is growing. This particularly applies to exploration seismology, where development of processing packages has grown into a thriving industry. A number of integrated software systems, mostly specialized and streamlined for reflection seismic data, are available.

Although highly advanced, commercial processing packages are still built for specialized indutry users. For a broader geophysical community, reliance on such software may not be satisfactory for several reasons. First, while being highly efficient in their primary fields of application (typically, 2- or 3-D common mid-point reflection data processing), commercial packages could become awkward in handling other types of data. Examples from seismology include wide-aperture reflection-refraction or earthquake data, where native support for flexible, multi-component processing, spherical-Earth geometry, and travel-time analysis is critical. Second, commercial packages often require installation of other systems (e.g., databases or rendering systems) whose support could be difficult or expensive in a University environment. And finally, licensing costs are often prohibitive, particularly when utilizing large multi-processor computer systems.

Open-source seismic processing provides a low-cost alternative to commercial software and, with an appropriately directed development, an ability to adapt to the changing research needs. The best-known examples of such kind are Stanford Exploration Project (SEP) software, SIOSEIS (http://sioseis.ucsd.edu/), and Seismic Un*x, a free reflection processing system developed at the Colorado School of Mines (Stockwell, 1999). Seismic Un*x has been broadly used in research and teaching seismology (e.g., Templeton & Gough, 1998) and also in smaller-scale seismic processing in the industry. However, all these packages are still strongly optimized for reflection processing, and their ability to handle more complex datasets is limited. Examples of such complex datasets commonly encountered in refraction and earthquake seismology include multicomponent, variable-length and sampling interval seismic records combined travel times and amplitudes. Crustal wide-angle seismology requires an ability to account for the Earth's curvature during data processing and in some cases uses thousands of files for data input. In a broader perspective, a system that could handle borehole logs, potential-field data, velocity and gravity models, and offer improved PostScript rendering capabilities would help to integrate the data analysis and reduce the need for data reformatting.

Here, we present our ongoing development of a system providing the flexibility, and functionality that are found neither in Seismic Un*x nor commercial packages. The system, called SIA (no spelling out available!), was initially developed at the University of Wyoming and continued at the University of Saskatchewan (http://seisweb.usask.ca/SIA). It represents a decade of extensive efforts for integration of academic-style seismic data analysis with the polish and performance of a commercial

seismic processor.

The guiding principle of SIA design is decentralization of processing and its abstraction from the content of the character of the particular seismic and geophysical dataset. With the recent modifications of the system, this idea was carried further, to an introduction of dynamic linking, conversion to C++, parallelization, and integration with a Graphical User Interface (GUI). In the following, we describe the development of the package since the previous publications (Morozov and Smithson, 1997; Morozov, 1998). We start with a brief recast of the key design concepts and emphasize the new parallel and GUI functionalities.

2. SIA seismic and geophysical data processing system

Initially, SIA started as a replacement for Cogniseis's DISCO processing system to provide a means to use many modules written by the students of the Program for Crustal Studies at the University of Wyoming. Consequently, its key design requirements were typical of massive reflection seismic processing: 1) high throughput achieved by processing tools (modules) operating in a common address space, with custom executables built for each job, 2) seismic processing sequences ("jobs") described using a specialized scripting language and executed in (normally) unattended processes, and 3) a multi-user development and processing environment. In addition, several extensions of the reflection data processing model were made, and particularly, an original backpropagation execution logic was introduced (Morozov and Smithson, 1997). The system supported (as it does now) processing scripts similar to those of DISCO.

The structure of an SIA data processing flow, implemented as C++ class named

`PROJECT`, is shown schematically in Fig. 1. Several tools are arranged in a sequence according to the processor's requirements. Each of the tools corresponds to a C++ class (referred to as "module" by Morozov and Smithson, 1997) implementing the "Edit Phase" (parameter input) and "Process Phase" (trace processing) methods. These are the only two methods required for seismic processing. Additional methods (such as providing dynamically changing tool names or progress indicators) can be implemented by the modules requiring closer integration with the GUI. The modules can also post their objects (such as velocity models) for their use by other modules in the flow. In addition, the modules have access to globally visible C++ objects providing the SIA system monitor, database, and the Parallel Virtual Machine (PVM) functionalities. All these classes are stored in precompiled object libraries and linked dynamically from shared libraries when the flow is built and started from the GUI or a batch script.

Unlike traditional seismic processing systems (e.g., DISCO, ProMAX, SEP, SIOSEIS, or Seismic Un*x) the system has no special input modules or expects any trace data at its input. Some tools (such as performing database operation or plotting) do not perform any operations with the traces, and the corresponding modules do not need to implement the Process Phase. This makes the system more flexible, making it a useful framework for more than just seismic data.

From any module, seismic traces are accessed by contacting the monitor via `SIA.input()` and `SIA.output()` methods. These methods return pointers to the structured input and output trace data gathers shared by the adjacent tools (Fig. 1; cf. Morozov and Smithson, 1997). The flow monitor takes no part in moving the traces

through the tool sequence, and the modules are free to modify the states of both of their inputs and outputs. This could result in data propagation pattern within the flow that could become elaborate (Morozov, 1998); however, for a typical single-trace filtering operation used in most seismic tools, the Process Phase code is quite straightforward:

```
boolean FILTER::process() {
 TRACE *t = SIA.input()->pass_trace(SIA.output());
 if ( t ) {
   filter(t);
   return OK;
 }
 return FAIL;
}
```

Here, tha `pass_trace(…)` method transfers the trace to the output of the tool, `return OK` statement informs the monitor that the module has produced an output, and `return FAIL` is used to request more input data (Morozov and Smithson, 1997). The seismic trace is represented by an object of class `TRACE` that provides access to all of its formatting, data, and header information. Trace headers are free-format and are fully customizable by the user (Morozov and Smithson, 1997). The method `filter(TRACE*)` above should implement the desired filtering of trace `t`.

The operation of the monitor is independent of the character of the data being processed and can be briefly summarized as follows (Morozov and Smithson, 1997). When the job is started, all data gathers are emptied and the modules are called recursively in reverse order, starting from the last one to the module currently marked with the “end-of-file” flag. Once a module returns OK (as in the example above), the process is repeated again from the end of the flow. If all modules return `FAIL`, the end-

of-file flag is moved to the next (end-of-file) module, and the process is repeated until no modules produce any outputs. This simple scheme resembling the backpropagation inference engine of the programming language PROLOG maintains the minimum possible number of traces in the data buffers and allows the modules to fully control the data flow and termination of the process.

Because the sequence of tool invocations in SIA is driven by a logical inference mechanism rather than by the input data, no restrictions on the types of data or character of processing are imposed. Data can be loaded, removed, or directed backward in the processing sequence, or the flow could operate without input data at all. In the course of its use in several areas of geophysics (mainly wide-aperture, reflection, and teleseismic seismology, and recently 3-D potential fields), data types were considerably generalized and several additional system features were implemented (Fig. 1):

1) Variable data formats, sampling intervals, record lengths and time starts.
2) “Traces” can now contain linear arrays (seismic records) or 2- and 3-D arrays (multicomponent seismic records, or 2-D grids used in potential field processing).
3) “Tools” can be represented by binary codes or macro-commands combining groups of other tools with coherent parameterization optimized for a particular task, Macro-commands can be defined by any user.
4) Graphics subsystem for rendering complex images in PostScript and building custom Graphical User Interfaces;
5) Extensive use of command line, trace headers, and an integrated job text

preprocessor for flexible tool parameterization.

6) Maintenance utilities including automatically generated HTML documentation and tools for generation of macro-commands and processing examples (see http://seisweb.usask.ca/SIA/examples/).

7) Web service allowing execution of complex custom flows on remote systems and providing software updates. This service was developed after the initial version of this paper was submitted and is described in a separate paper (see http://seisweb.usask.ca/SIA/ws.php; Morozov et al., in review).

3. Development

Addition of new tools into the generalized processing framework (Fig. 1) fills it with the content for the particular area of application. Compared to the original version (Morozov and Smithson, 1997), code development for the system was significantly simplified, mainly due to the use of C++ encapsulation and inheritance, dynamic linking, and improved maintenance and documentation support. The addition of new tools does not require any modification of the monitoring program and can be done by the users. Graduate Geophysics students at the University of Saskatchewan now routinely contribute new tools as parts of their class projects. Code templates are available for the basic methods of data handling, such as one trace in – one trace out, a buffered single gather, or a sliding trace window (see http://seisweb.usask.ca/SIA/examples/templates/).

Although the old code based on the C language (Morozov and Smithson, 1997) is still fully supported, we use the C++ model for all new development. In this model, an

SIA tool named, for example, `mytool` is described by a "parameter definition file" `mytool.mpar`. This file contains descriptions of all module's parameters, documentation, C++, C, or Fortran codes or object libraries used for its building. This file is used by a system utility to generate the corresponding UNIX `make` file, resolve library dependencies, and to create both HTML pages and the on-line documentation displayed by the GUI.

Apart from `mytool.mpar`, a single C++ file containing the C function `void *mytool_init()` must also be provided. This function is called once during flow initialization and returns a pointer to the module's data object. Normally, it simply returns `new MYTOOL`, where class `MYTOOL` is derived from a base class `SIA_MODULE` and overloads (if needed) two of its methods:

1) `int MYTOOL::edit()` – the Edit Phase performing parameter input. It returns an integer status specifying whether the module needs to be called during the Process or end-of-file Phases,

2) `boolean MYTOOL::process()`- the Process Phase called when data objects are propagated through the flow, as described above.

To implement the two methods, no knowledge about the monitor operation or presence of other tools is required. Along with `MYTOOL` class, any number of other C/C++, or Fortran codes can be included and placed into the shared module library. Libraries of C and Fortran subroutines and C++ classes (such as performing Fourier transforms, filtering, Least Squares inversion, and implementing complex arithmetic and Matlab-like matrix manipulations) are provided to facilitate development. In our

experience, a student familiar with C++ can usually develop a reasonably complex tool in several days,

The configuration of the system allows maintaining multiple versions of the binaries for different computer architectures from a single set of source codes. In such a way, the system was supported at the University of Wyoming and Rice University under Sun Solaris, 32- and 64-bit SGI Irix, and recently under 32- and 64-bit Red Hat Enterprise Linux at the University of Saskatchewan.

Accumulation and exchange of processing expertise is as important for working on complex research projects as algorithm development, particularly in an educational environment. To date, a limited support for systematic documentation is facilitated in SIA by a special tool posting fragments of job scripts in a common database. Any user can select a portion of a processing flow, specify a name and a category for the example, and post it where it can be viewed by others (http://seisweb.usask.ca/SIA/examples). Similar tools create macro-commands and build a library of standard default configurations for the various tools. In addition, processing examples can be simply cut and pasted from, for example, an Internet browser or email.

4. Parallelization

The complete processing flow objects can be copied across the PVM interface (Fig. 1) and executed separate processes on the same or remote hosts. This is the normal mode of GUI operation on multiprocessor subsystems (below), in which all of the computationally-intensive data processing is performed on remote compute servers without overloading the GUI host. Some tools (such as `flow`, used to organize parallel

processing; see http://seisweb.usask.ca/SIA/modules/flow/mod.html) spawn groups of processing sub-flows of their own. All processes communicate between each other and with the GUI using `printf(…)` – like messages facilitated by the SIA PVM interface. Along with these messages, the processes also exchange data traces, database tables, and other objects. Execution of the processes is asynchronous, with message queuing and retrieval handled by PVM libraries.

Due to encapsulation of the entire processing in a single `PROJECT` object (Fig. 1), sub-flows can also be invoked as parts of specialized algorithms. For example, such sub-flows were used to implement custom processing within the loop of generalized pre-stack seismic migration (Morozov and Dueker, 2003).

In order to manage submissions of specialized remote processes, an additional layer of abstraction was created. The user is allowed to define groups of compute hosts and applications assigned to the execution of specific tasks, such as running sub-flows, performing interactive displays, or creating log files. As a result, the tools do not have to specify the exact host and program names but use these task names in order to invoke these applications. For example, depending on the user's definition of "`psview`", a request for a `psview` executes `ghostview`, `kghostview`, `display`, or other PostScript viewing programs on different hosts. In a classroom setting, this technique could provide a near-synchronous cloning of displays on multiple computer screens. Also, parallel jobs can be easily reconfigured for using fewer or more nodes without any changes in their parameters, simply by changing the submission configuration (Fig. 2).

In parallelizing seismic and other processing, we implemented an additional

layer of abstraction by making job descriptions independent of the actual PVM configuration. For example, unlike in ProMAX, where the "parallel begin" tool has to be told explicitly the names of the nodes to which to distribute the load, parallel subflows in SIA are started on all the nodes assigned to performing the task named "compute" (Fig. 2). As a result, jobs can be easily reconfigured for using fewer or more nodes without any changes in their parameters, simply by changing the submission configuration (Fig. 2). Service programs (such as job logging, and graphics, including the various PostScript and X viewers, or GMT) can also be selected in the same way and executed on different (including multiple) nodes.

Finally, some tools can generate slave processes that do not execute processing flows of the kind shown in Fig. 1 yet employ the same PVM communication mechanism. For example, this approach was used to implement 3-D visco-elastic finite-difference modeling integrated into the processing system through module `efd3d` (http://seisweb.usask.ca/SIA/modules/efd3d/mod.html). In this case, model building is performed by broadcasting the corresponding instructions from the Process Phase of `efd3d`, followed by time stepping, editing, and output instructions used to control and synchronize the wavefield simulation.

5. Graphical User Interface

As with other similar projects (SEP, Seismic Un*x, SIOSEIS), the advantages of batch (unattended) processing of large volumes of data have historically come at the expense of an intuitive and consistent graphical user interface. Processing jobs had to be described using either UNIX shell or specialized scripts, which always resulted in a

significant learning curve and increased the likelihood of errors. A specialized GUI would relieve the processor of scripting, give the system a modern look and feel, and simplify learning by bringing all the documentation to the user's fingertips. Recently, a modern graphical user interface (GUI) was designed for the SIA system (Fig. 3).

The GUI is based on the cross-platform Qt libraries from Trolltech, the same libraries on which the popular KDE Linux interface is based. Using Qt relieved us of any X-windows event handling and allowed to incorporate many of the most up-to date GUI design approaches, such as the multiple-document interface, window docking, themes, and platform-independent configuration. Although currently we perform all our development under Linux, other UNIX-type systems such as Solaris, BSD, or Apple's OSX should also work with minimal effort.

The main GUIU frame is subdivided into four components that are used most often: the tool library, current module parameters, job editor, and job monitor (Fig. 3). When the job flow is edited and submitted for execution, the corresponding `PROJECT` object is spawned to the appropriate host(s) and executed. During its operation, `PROJECT` periodically sends information messages to the GUI process, and some of these messages are displayed in the job monitor window. In principle, running jobs may also be programmed to alter some of their parameters which will be immediately displayed by the job editor. At the same time, `PROJECT` is also constantly listening to PVM messages from the GUI, and through these messages, the user can control the remote execution of the flow. Note that different running flows, even those submitted from the same job editor, do not interfere with each other and are independently managed

by the monitor.

The **tool library** (Fig. 3a,b) offers access to over 220 processing tools, about 30 of which are to various degrees experimental. The tools are arranged into packages (e.g., reflection, travel-time, earthquake, potential field data processing, graphics, or development) which may be tailored by the administrators to meet the needs of a variety of users. Within each package, groups of tools (such as input/output, plotting, etc.) are displayed on tab panes (Fig. 3). A mouse click action on a tool displays its documentation, similar to the one posted at http://seisweb.usask.ca/SIA/sia-index.html. As with ProMAX, typing within the library window invokes a search utility that attempts a keyword search for a tool. When a tool is found, it can be dragged and dropped into the processing flow.

Because the system is intended for users working in different research areas, tools extracted from the different packages could have different pre-set default configurations. For example, applications of the Automatic Gain Control (AGC) in high-resolution, exploration, and earthquake seismology typically use very different time gate lengths. Therefore, we provide several initial configurations for the same AGC tool included in these three packages, and the user is allowed to select the most appropriate configuration.

The job editor (Fig. 3c) is the central component of the user interface. A multiple-document interface allows several flows to be opened simultaneously, in which the user can edit and execute multiple jobs. Docking windows and tool bars allow a user to customize the layout of the program to make effective use of multi-display systems.

Tools and configurations may be copied between jobs, saving the user time and reducing entry errors. Clipboard functions, tool tips, and context-sensitive help are provided to further simplify usage.

Jobs are assigned descriptive names that are passed to the flows during run time (Fig. 1) and are used to identify them in the job monitor. Job flow descriptions can be built from the tool libraries and examples, and they can also be imported into the interface by dropping text into the flow window. Once placed in the job editor, both tools and parameters can be rearranged by the drag and drop process, making it easy to correct mistakes or change settings.

The job editor displays parameters of all tools in the form of a table (Fig. 3c). Parameterization can be extended (for example, several hundred lines to describe seismic velocity models). Several types of parameters are currently defined (cf. Morozov and Smithson, 1997): 1) integer, real, double, character string, and Boolean values, 2) selectable and editable text lists, 3) color, fill, line style, font, and color palette names used by the graphics subsystem, 4) file, module, or flow names, and 5) compute host names, including names of user-defined virtual clusters. Parameters of different types are rendered differently; for example, Boolean values are represented by check boxes, and selectable values – by drop-down lists. Color highlighting distinguishes between the floating point, integer, and character values.

Proper and sufficient documentation is critical in large-scale processing. At present, processing flow documentation is implemented by allowing the user to attach free-text commentaries to the modules, parameter lists, and parameter groups. The

commentaries can be edited and displayed in tool tips.

Once the job parameterization is complete, the flow is submitted for execution through a remote process communication interface utilizing the PVM (Fig. 1). Jobs may be submitted either for parameter checks (Edit Phase only) or for full processing. If an error is detected or a message issued from a running job, the display of the job (if currently open) is automatically updated using context-dependent color highlighting. The corresponding error messages are displayed in parameter tool tips and also saved in the job log.

Current module parameterization occupies a permanent window in the GUI (Fig. 3d) because of its continuous use during editing and also because some of the SIA modules can have quite extensive parameterizations. For example, module `image` (http://seisweb.usask.ca/SIA/modules/image/mod.html) currently offers 57 optional parameter lists to describe its various graphics elements. In the module parameterization window, these lists are displayed graphically in the form of a tree from which the lists can be dragged and dropped into the job editor.

The job monitor (Fig. 3e) is implemented by simply displaying the information PVM messages received from the running flows and relaying user's commands back to them. Therefore, operation of job monitor is completely asynchronous and independent from the job editing sessions.

In all GUI components, we make an extensive use of the drag and drop functions, tool tips and status lines to identify options and features while reducing screen clutter. The fonts, colors and other options can be modified to improve their appearance.

6. Discussion and conclusions

Although initially designed to extend a reflection processing package (DISCO) to wide-angle seismic data analysis (Morozov and Smithson, 1997), its generalized processing logic have allowed SIA to be extensible to a far broader range of applications. Neither its processing flows nor core databases (Fig. 1) utilize the specifics of seismic data analysis. The tools are not limited in their types of operation, and a number of non-seismic applications were included into SIA (see http://seisweb.usask.ca/SIA/examples), with the benefits of uniform parameterization, GUI, interaction with other tools, web service, and unified software maintenance and documentation.

The development of the system was driven by the needs of a fairly broad research program extending from shallow to regional and global seismology (http://seisweb.usask.ca/ibm/research.html). As a consequence of this broad scope, SIA offers capabilities for nearly complete reflection and wide-aperture seismic processing combined with support for multicomponent, variable-format data, extensive database capabilities, and input/output in several formats (e.g., SEG-2, SEGY, PASSCAL-SEGY, SEG-P, GSE3.0, CSS3.0, and SAC). Several original inversion codes (such as 2-D reflection and generalized 3-D receiver function migration, genetic algorithms, artificial neural networks, and parallel 1-D and 3-D finite difference modeling) were developed. Tools for 2-D and 3-D processing and inversion of potential fields were recently included. Interfaces to popular program packages, such as Datascope, Generic Mapping Tools (Wessel and Smith, 1995), *rayinvr* (Zelt & Smith, 1992), *reflectivity* (Fuchs & Müller, 1971), and Seismic Un*x, simplify interoperability with other approaches.

Although SIA is being continuously developed, it already represents a fully functional system exceeding its commercial analogs in its scope and many other aspects important for academic researchers. With further development, it could provide an excellent research tool and software development and integration framework for many areas of fundamental and applied geophysics.

7. Acknowledgments

The initial SIA development was inspired by one of the authors' (I. M.) research on a number of projects sponsored by NSF (EHR-910-8774-02, EAR99-03235), Gas Research Institute (contract #5089-260-1894) and U.S. Air Force (Grants F49620-94-1-0134, F49620-94-A-0134, F49620-96-1-0326, DSWA01-98-0015, DTRA01-01-C-0057, and DTRA01-01-C-0081). However, none of these projects has directly requested or supported this development.

8. References

Fuchs, K., & G. Müller (1971), Computation of synthetic seismograms with the reflectivity method and comparison with observations, J. R. Astronom. Soc., 23, 417-433.

Morozov, I. B. (1998), 3D seismic processing monitor, Computers & Geosciences, 24 (3), 285-288.

Morozov, I. B., & S. B. Smithson, (1997), A new system for multicomponent seismic processing, Computers & Geosciences, 23, 689-696.

Morozov, I., B. Reilkoff, & G. Chubak, Prototype web service for geophysical data processing, Computers & Geosciences, in review.

Morozov, I.B., & K. G. Dueker (2003), Depth-domain processing of teleseismic receiver functions and generalized three-dimensional imaging, Bull. Seism. Soc. Am, 93 (5), 1984-1993.

Stockwell, Jr. J. W. (1999), The CWP/SU: Seismic Un*x Package, Computers & Geosciences, May 1999.

Templeton, M. E. & C.A. Gough (1999), Web Seismic Un*x: Making seismic reflection processing more accessible, Computers & Geosciences, 25 (4), 285-288.

Wessel P., & W. H. F. Smith (1995), New version of the Generic Mapping Tools released, EOS Trans. Am. Geophys. U., 76, p. 329.

Zelt C.A. & R.B. Smith (1992), Seismic travel-time inversion for 2-D crustal velocity structure, Geoph. Journal International, 108, 16-34, 1992.

Figures

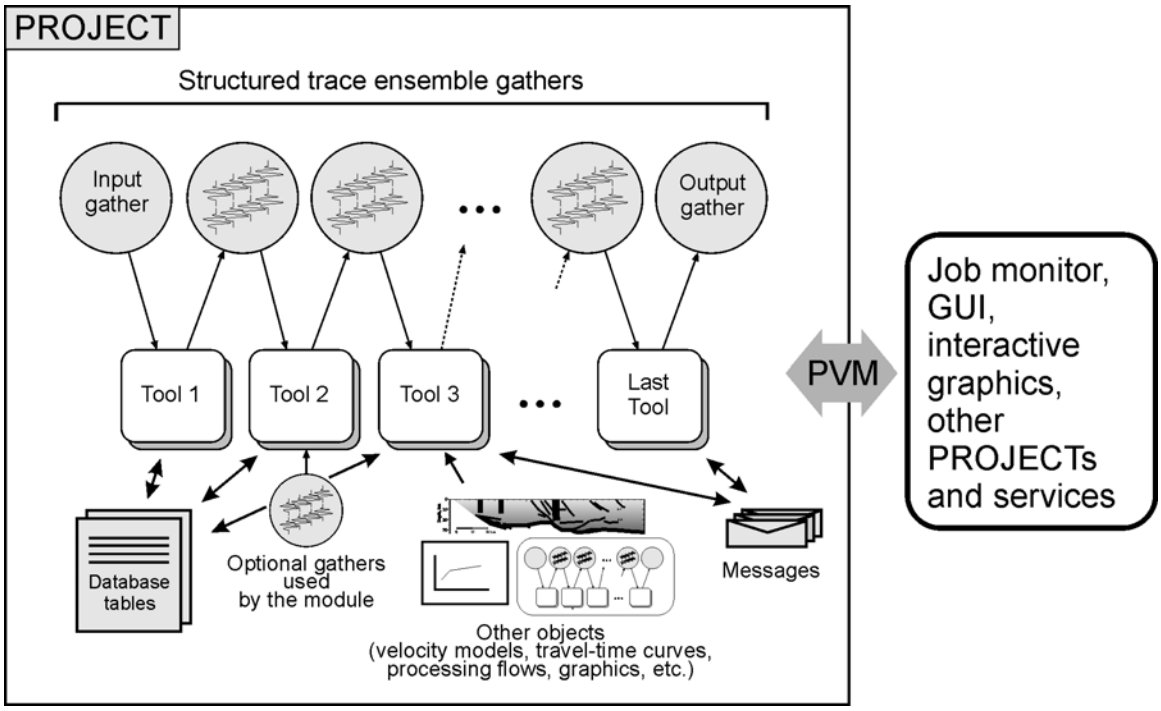

Figure 1. Structure of SIA processing flow, represented by class `PROJECT` (modified after Morozov and Smithson, 1997). Processing flows consist of linked sequences of tools sharing structured trace ensemble gathers (normally corresponding to multicomponent seismic records), database tables, and various custom data objects. Non-blocking PVM messages are used for communication of the flow with the GUI, other flows, and services.

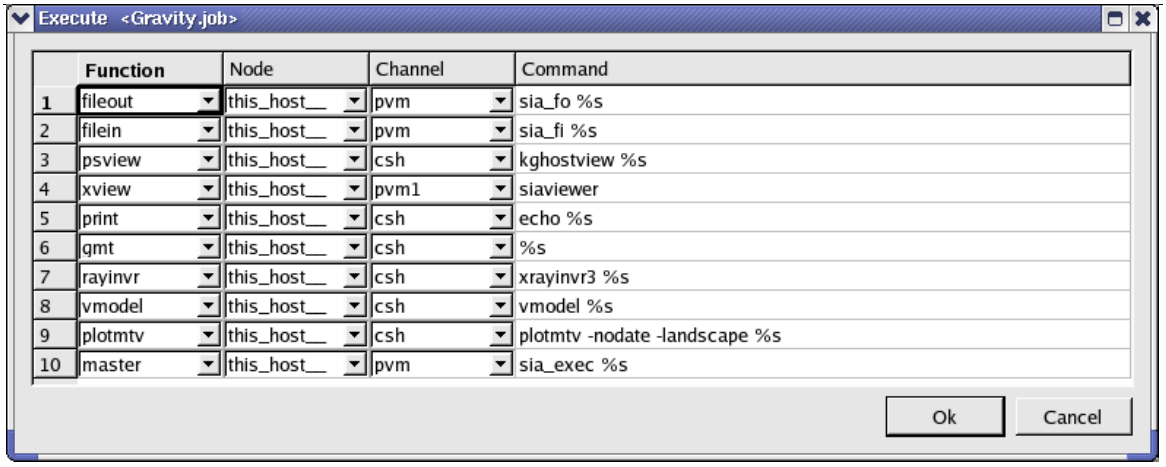

Figure 2. Process submission module of the GUI, showing the machines available to the user via PVM, the available task names, the types of submission (e.g., via PVM or UNIX shell), and the formats of the corresponding program calls. The tasks specify the symbolic names of the actions requested by the processing flows (Fig. 1), such as: "`master`" (for master flow processes), "`compute`" (embedded sub-flows), "`psview`" or "`xview`" (display PostScript or interactive X-windows graphics, respectively), and others. By checking the appropriate lines in this list, the flow can be executed on different host configurations without changing its parameters. Note that the machines listed in this view could represent individual compute hosts or their groups.

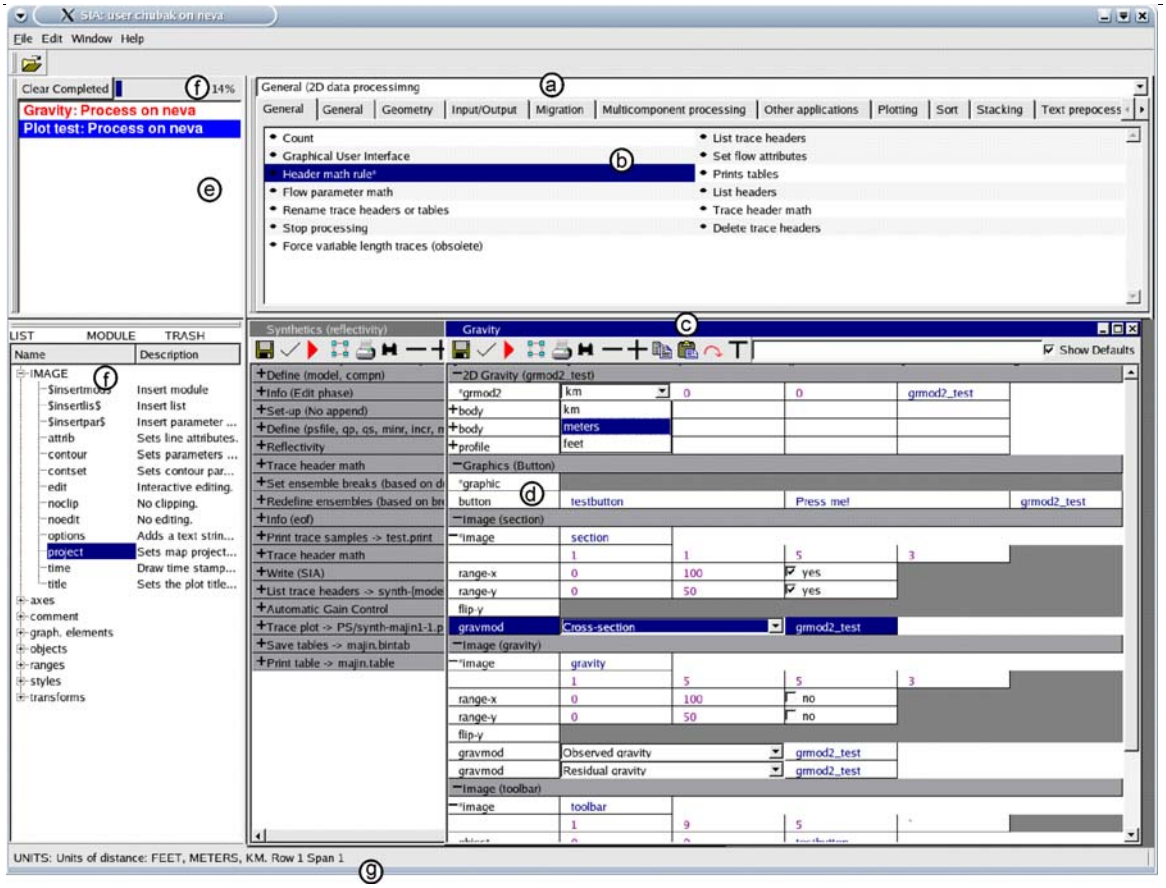

Figure 3. Main SIA Graphical Interface window including: a) selectable tool packages, b) tool library, c) multiple-job flow editor; d) parameterization of the selected tool; e) job monitor, and f) status line giving brief information about any item at which the cursor is pointing. The job in front of window (c) executes interactive 2-D gravity modeling, and the job in the back performs 1-D synthetic seismic modeling using the *reflectivity* method (Fuchs & Müller, 1971). For a compact display, tool parameterizations can be hidden leaving only one-line summaries that may change to display job progress (window c). Job flow editor windows (c) also include tool bars providing the flow-related functionality.