CRS OFFICE: A JAVA GRAPHICAL USER INTERFACE FOR THE CONVENTIONAL CRS STACK

A. Gomes, L. Leite, J. Costa, Z. Heilmann, J. Mann

email: *agomes@ufpa.br*

keywords: *CRS, Java, CRS Office, Seismic Graphical User Interface*

ABSTRACT

*This paper describes the development of CRS Office, a friendly Java interface for the CRS (Common Reflection Surface) system, and an example of application to synthetic and real marine data from offshore Brazil. The interface is not limited to the conventional CRS stack process, but it is intended to integrate different processes through Java tools with application centered at seismic imaging with the CRS technology. The interface is extended to print on the screen the outputs results, and to do this it is used the CWP/SU (Center for Wave Phenomena, Colorado School of Mines, Seismic Un*x) format and packages.*

CRS technology has increased its participation in the last years in data processing, nevertheless, its use is restrict to some groups due partly to the non-existence of a friendly interface to make its use more efficient and enjoyable. In order to present a solution for this issue, we started the development of CRS Office, a graphical user interface for the CRS stack processing code of Dr. Jürgen Mann released in Karlsruhe, Germany. The CRS code is written in C++, and the CRS Office was developed as a wizard console in NetBeans, IDE (Integrated Development Environment) 5.5 GUI (Graphical User Interface) builder, in Java language program. CRS Office gathers the Java's platform advantages, like portability and re-usability, with the computational efficiency of the C++ program language. The GUI allow the interaction between the user and the CRS/C++ code without requirement to be involved with complex shell scripts and Makefiles. Basically, in the present stage CRS Office reads the parameters from the widgets (TextFields, ComboBoxes, Check Boxes, and so on), does the CRS stack based in these parameters, and print the output results creating and executing shell scripts and Makefiles in a hidden way. A further and natural development consists in a bigger integration between the CRS code and the CRS Office using JNI (Java Native Interface) tools. The project development is under maintenance, evolution and inclusion of new tools.

INTRODUCTION

There are several tools to enable quick and easy development of user interfaces. Before we chose the Java platform, we investigated some graphical interfaces, like Glade and Qt Designer. Glade is used for the GTK+ toolkit and the GNOME desktop environment, and released under the GNU GPL License. Qt Designer is a powerful GUI layout and forms builder, which enables rapid development of high-performance user interfaces with native look and feel across all supported platforms. Qt Designer works stand-alone, or integrated with IDEs like Microsoft Visual Studio, .NET, and Eclipse (Qt Jambi only). Anyway, it includes powerful features such as preview mode, automatic widget layout, support for custom widgets, an advanced property editor and more.

We chose Java due to the possibility of integrating the big amount of free geophysics packages based on Java platform and softwares already available in our own interface. Among geophysical packages based on Java platform that we investigated, we can cite Jest (Schwab and Schroeder (1998)) and JavaParty (Philippsen and Zenger (1998)). Jest is a software that comprises Jam (Java and mathematics), a

general and extendible library for numerical optimization for science and engineering, and Jag (Java and geophysics), a particular extension of a framework for seismic image processing. JavaParty is a software that transparently adds remote objects to Java purely by declaration while avoiding the disadvantages of explicit socket communication, and avoids also programming overhead of RMI (Remote Method Invocation) and many disadvantages of the message-passing approach in general. JavaParty is specifically targeted towards, and implemented on, clusters of workstations. It hence combines Java-like programming and the concepts of distributed shared memory in heterogeneous networks. Besides, Java presents itself as a reasonable easy language, and it has a relation to professional geophysical seismic processing packages. A friendly Java interface for the CRS system, and applications to synthetic and real data is a goal in our participation in the projects under development in our institution, and to the relation with the WIT (Wave Inversion Technology, University of Karlsruhe, Germany) Consortium through national and international cooperation.

BUILDING CRS OFFICE

CRS Office successfully separates optimization techniques, carried out by the CRS C++ code, from the application softwares represented by the Java GUI, discarding any knowledge about shell scripts and Makefiles by the user. In order to introduce CRS Office, some basical informations about Java is necessary.

In science and engineering today, new ideas are implemented, simulated and tested using computer softwares. In general, when a new idea arises and the development of a new software is required, it is necessary to start from scratch; consequently previous software will not be useful in the future for another software. This increases the cost and the development time of a new software. By thinking about time and cost situation, we idealized CRS Office as a Java application, since Java is an object-oriented programming language (Savitch (2006)) abbreviated OOP. OOP is a programming methodology that views a program consisting of objects that interact with each other by means of actions. CRS Office was developed under this paradigm, what makes the CRS Office code easier to maintain and re-use.

The Java platform is a programming environment consisting of the Java virtual machine (VM), and the Java Application Programming Interface (API). The Java API consists of a set of predefined classes. Any implementation of the Java platform is guaranteed to support the Java programming language, virtual machine and API. CRS Office, as all Java applications written in the Java programming language (see Figure 1), is first compiled and converted into a bytecode file, an intermediate language that is the same for all computers, that contains the same definition of the class or interface written JVM instructions. A bytecode file must have a file name identical to the name of the class or interface defined in the file, and a extension of ".class". Then, the bytecode is loaded into JVM, and the JVM executes its instructions translating the bytecode into the machine language for a particular operating system and hardware platform. CRS Office's instructions is to execute the CRS native code through shell scripts and Makefiles. Therefore, CRS Office code is partly portable, even among parallel computers with different memory models.

The first version of CRS Office was developed based on some free NetBeans IDE sample applications available on the Sun Microsystem homepage. NetBeans IDE provides great productivity tools. Also, Java allows bundling code into individual software objects that provides a number of benefits, including: modularity, information-hiding, code re-use, pluggability and debugging ease.

CRS OFFICE JAVA GRAPHICAL USER INTERFACES

CRS Office main frame is shown in Figure 2, and the interaction with widgets in Figure 3 where the user enters the processing parameters as shown. The respective explanations are in the figure captions. The Figure 4 is an example of information obtained after hitting a Help button. The bottom buttons start with the clear button, followed by the execution button, then the back and forward of the frames, and finally to cancel the executing process.

Continuing with the interaction with widgets of CRS Office, the Figures 5, 6 and 7 show up, if the options ZOsearch, Inistack or Optimize were chosen in the main frame, respectively. In these three frames the first top 4 text field give self-explanatory information of the contents that should be given, and the 16 text fields (in two columns) are processing parameters, where the help buttons can explain their content. The other bottom buttons are similar to the frame in Figure 3.

Clicking in the Execute button, if all the parameters are correct, a set of internal procedures that involve

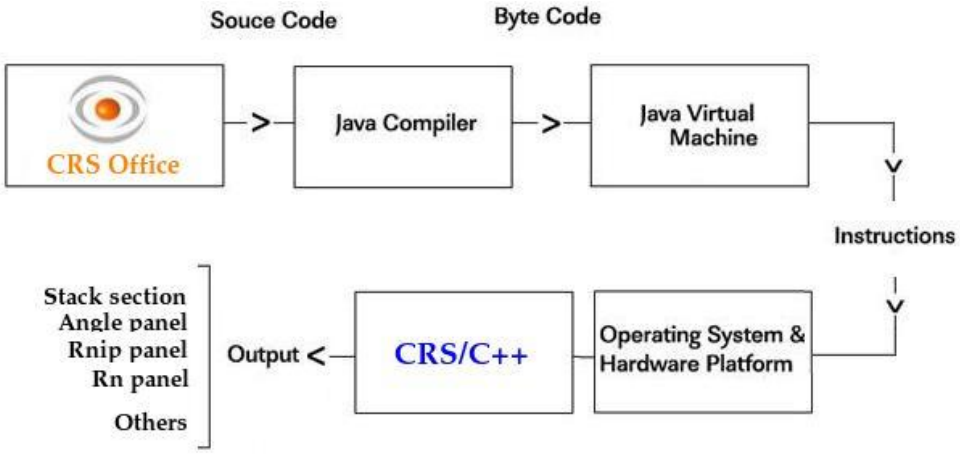

Figure 1: CRS Office execution layout. CRS Office source code is compiled and converted into a bytecode file that contains the same definition of the class or interface written JVM instructions. The bytecode is loaded into JVM, and the JVM executes its instructions translating the bytecode into the machine language for a particular operating system and hardware platform. The instructions of CRS Office is to execute the CRS native code through shell scripts and Makefiles.

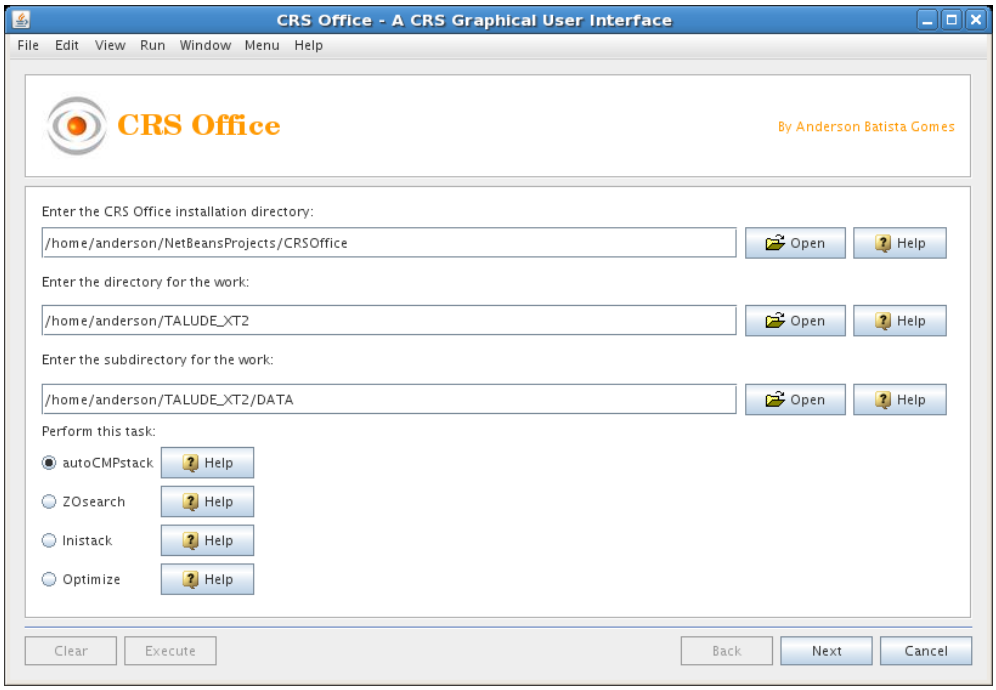

Figure 2: CRS Office main frame. It shows the text fields to enter the directory where the CRS Office is installed, the directory where the lines to be processed are located, and the sub-directory where the results will be written. The 4 tasks that can be performed are in the radio buttons. The help windows in the buttons give information about the actual task.

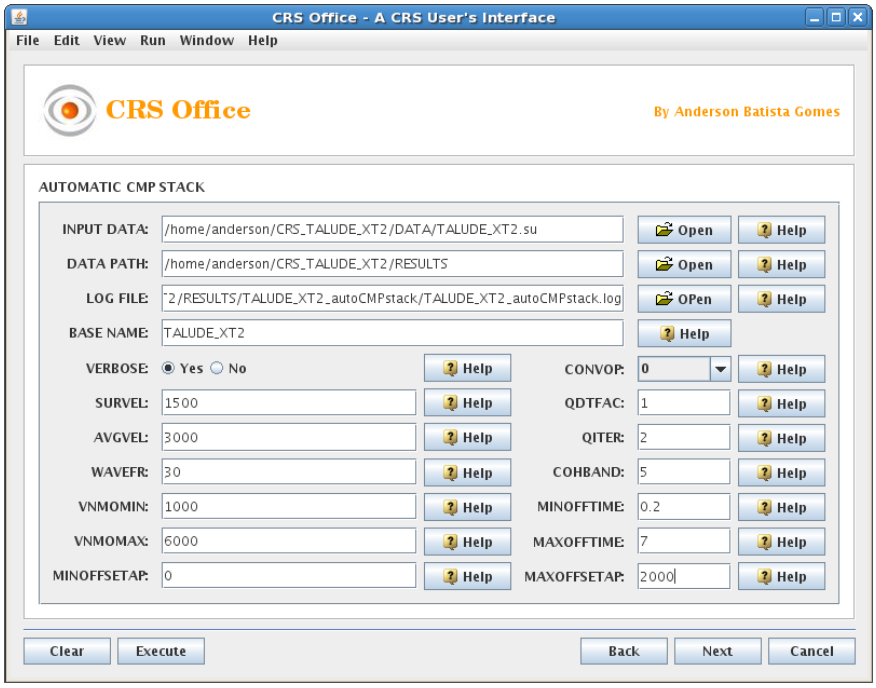

Figure 3: CRS Office second frame of widgets. This frame shows up only if the option autoCMPstack was chosen in the main frame. The first top 4 text field give self-explanatory information of the contents that should be given. The next 12 text fields (in two columns) are processing parameters, where the help buttons explain them. The radio button selects the verbose condition. The combo box serves to select the type of data for the coherence analysis. The bottom buttons start with the clear button, followed by the execution button, then the back and forward of the frames, and finally to cancel the executing process.

Figure 4: This frame is an example of information obtained after hitting a Help button, and in this case for the near surface velocity parameter (SURVEL).

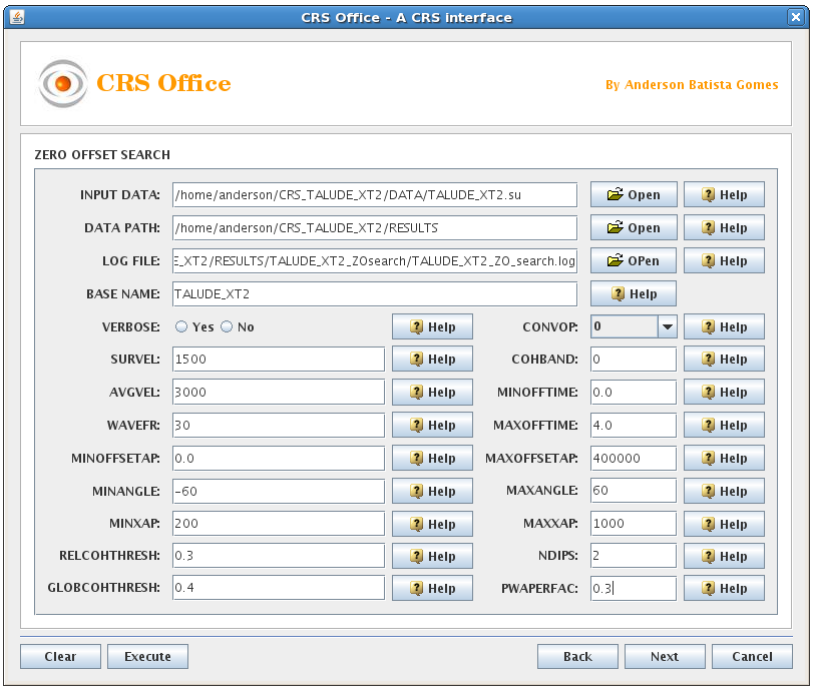

Figure 5: CRS Office third frame of widgets. This frame shows up only if the option ZOsearch was chosen in the main frame. The first top 4 text field give self-explanatory information of the contents that should be given. The next 16 text fields (in two columns) are processing parameters, where the help buttons explain them. The radio button selects the verbose condition. The combo box serves to select the type of data for the coherence analysis. The bottom buttons start with the clear button, followed by the execution button, then the back and forward of the frames, and finally to cancel the executing process.

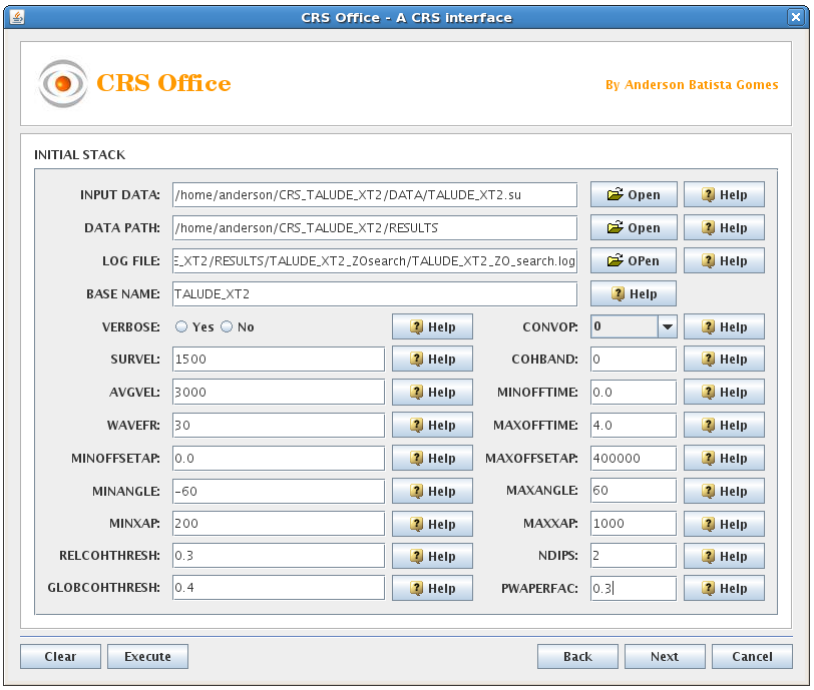

Figure 6: CRS Office fourth frame of widgets. This frame shows up only if the option Inistack was chosen in the main frame. The first top 4 text field give self-explanatory information of the contents that should be given. The next 16 text fields (in two columns) are processing parameters, where the help buttons explain them. The radio button selects the verbose condition. The combo box serves to select the type of data for the coherence analysis. The bottom buttons start with the clear button, followed by the execution button, then the back and forward of the frames, and finally to cancel the executing process.

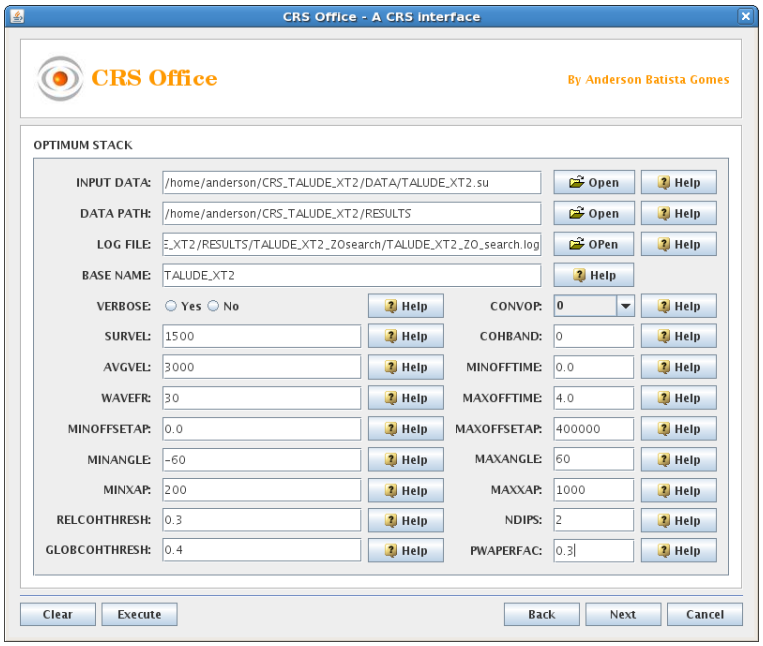

Figure 7: CRS Office fifth frame of widgets. This frame shows up only if the option Optimize was chosen in the main frame. The first top 4 text field give self-explanatory information of the contents that should be given. The next 16 text fields (in two columns) are processing parameters, where the help buttons explain them. The radio button selects the verbose condition. The combo box serves to select the type of data for the coherence analysis. The bottom buttons start with the clear button, followed by the execution button, then the back and forward of the frames, and finally to cancel the executing process.

Figure 8: Example of an error message that in this case the user entered an invalid value to the parameter of the type of data for the coherence analysis (CONVOP).

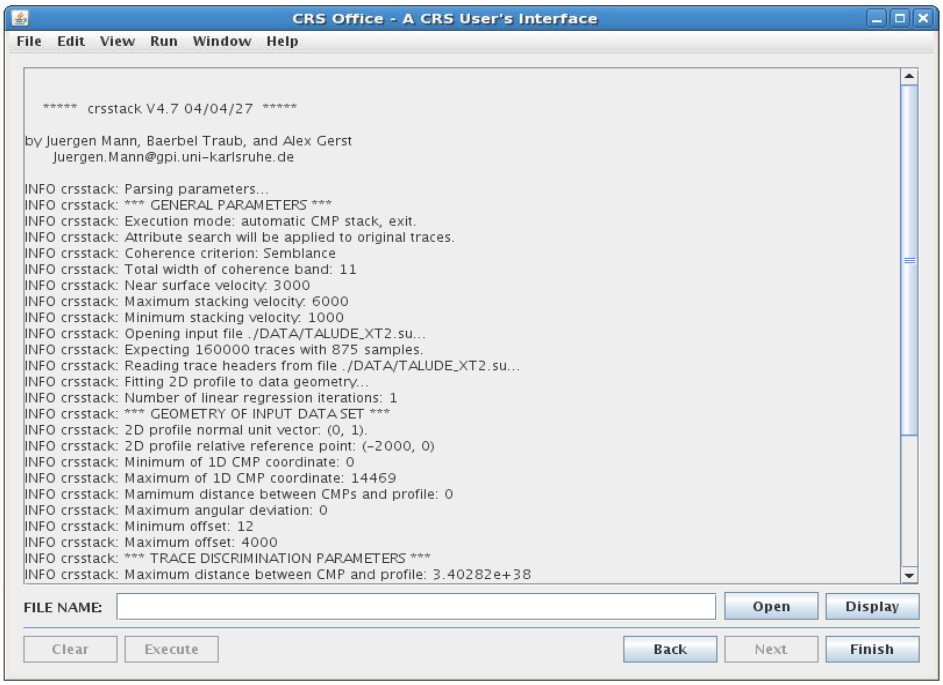

Figure 9: This frame shows the CRS Office log field giving some processing information. The text field is reserved for the name of the file in the directory Results to be seen by the SU tool suximage. The Open button is for browsing, and the Display button is to see the figure.

the creation of a *.par* file, shell scripts and Makefiles will be performed to initialize the execution of the native code responsible for the CRS processing. If the parameters are not correct, an independent subwindow will show up and an error message will inform the user how to proceed. The Figure 8 shows the example of an error message with explanation in the caption.

CRS Office executes these shell scripts and Makefiles implementing the methods *getRuntime* and *exec* of the *Runtime* class. The Runtime class extends further the Object class. The class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class. The class Runtime returns the runtime object associated with the current Java application. The method *exec* executes the specified string command in a separate process. This convenience method was the base of our ExecuteCRS class which executes the CRS native code. Once the process starts, the user can follow its evolution through a .log file shown in the CRS Office log field of Figure 9 with explanation in the caption.

Once the process is finished successfully, the user can browse and choose one among the CRS results to display it graphically using the suximage of the SU visualization tools. As we develop the interface, the CRS Office will tend to become more robust, and to support more SU tools. As a matter of fact, it will even

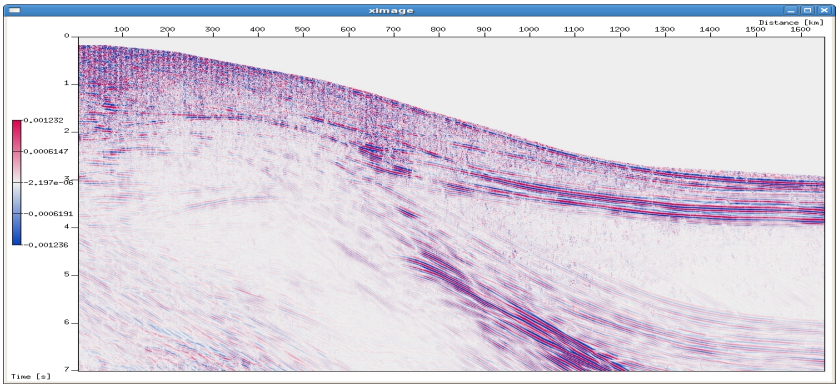

Figure 10: Stack section of the synthetic data.

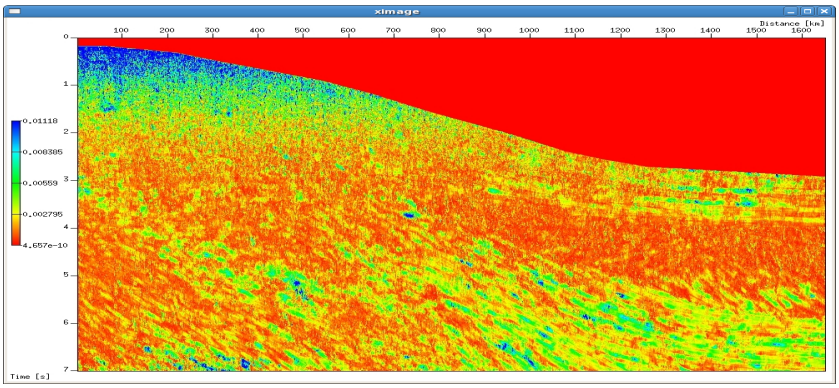

Figure 11: Coherence section of the synthetic data.

be possible to support all SU tools available; but, this is not our present intention, instead we are concerned directly only with the tools concerning with visualization.

CRS SYSTEM

The Common Reflection Surface (CRS) stack (Mann (2001)) is already rather conventional, and there is a tendency to demonstrate an advantage of CRS stack over conventional methods in simulating zero-offset sections. Besides that, for every zero-offset sample several kinematic wavefield attributes are obtained as useful by-products of the data-driven stacking process. The figures beneath show some results of a seismic processing of a synthetic and a real marine data from a set of a Brazilian offshore carried out using CRS Office. Figures (10), (11), (12), (13), and (14) show the stack section, the coherence section and the three attributes panels: emergence angle, Rnip and Rn for the synthetic data. Figures (15), (16), (17), (18), and (19) are for the marine data from a set of a Brazilian offshore.

FUTURE VERSIONS

CRS Office first version is still rather simple, but we can foresee it robust and intuitive. Among the future developments, we can have: (1) use of threads; (2) integration the CRS/C++ code with the CRS Office GUI with JNI; (3) inclusion of more SU tools; and (4) inclusion of more CRS system tools.

A thread can be loosely defined as a separate stream of execution that takes place simultaneously with and independently of everything else that might be happening. A thread is like a classic program that starts at point A and executes until it reaches point B, and it does not have an event loop. A thread runs independently of anything else happening in the computer. Without threads an entire program can be held up by one CPU intensive task, or one infinite loop, intentional or otherwise. With threads the other tasks

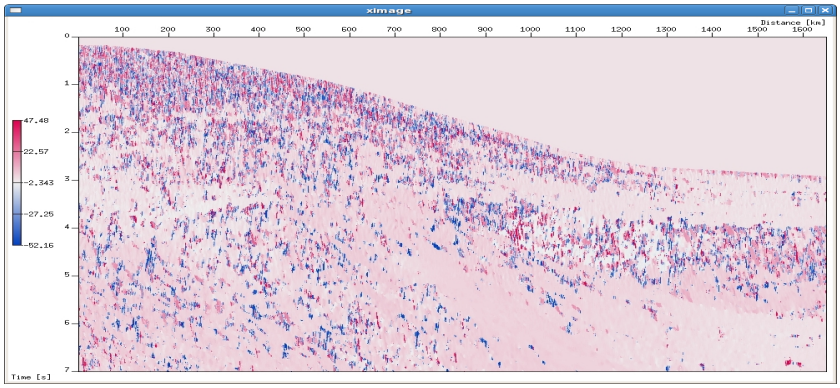

Figure 12: Angle panel of the synthetic data.

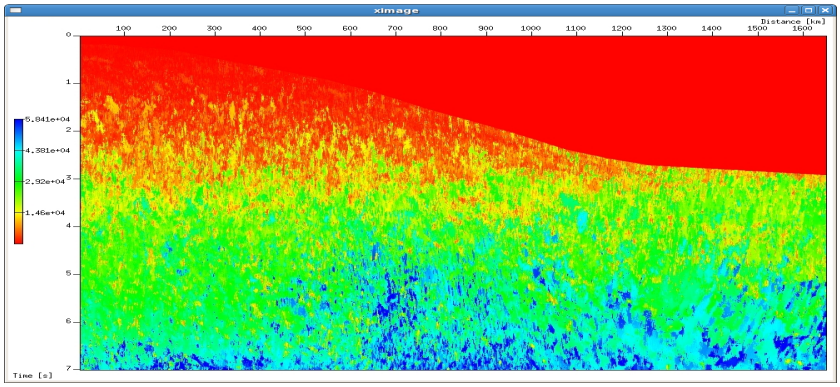

Figure 13: Rnip panel of the synthetic data.

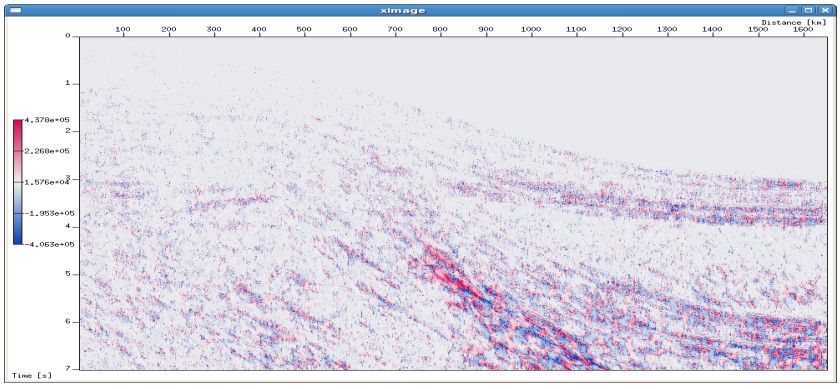

Figure 14: Rn panel of the synthetic data.

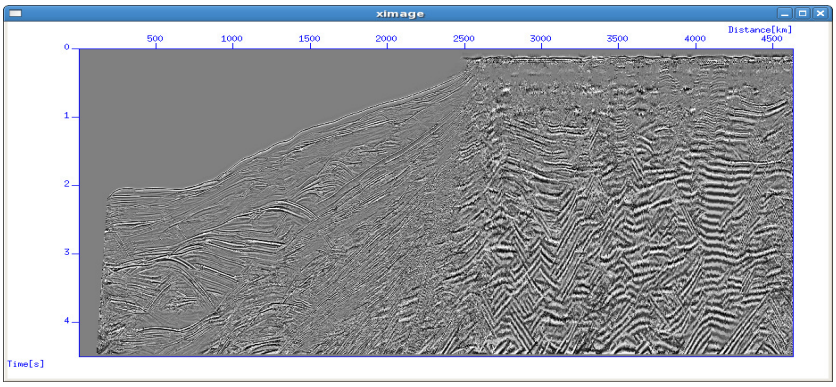

Figure 15: Stack section of the real data.

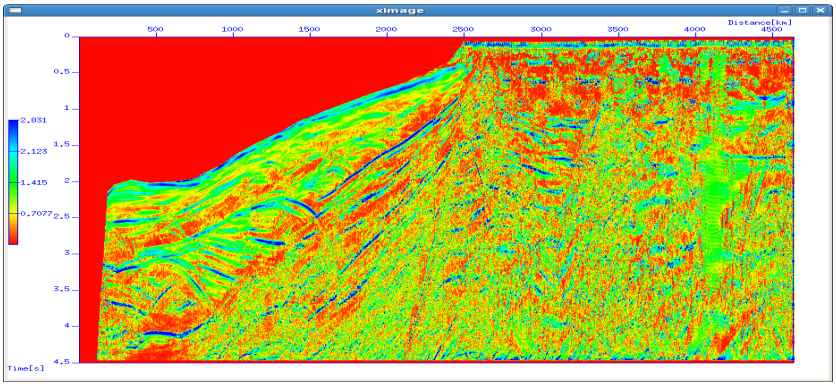

Figure 16: Coherence section of the real data.

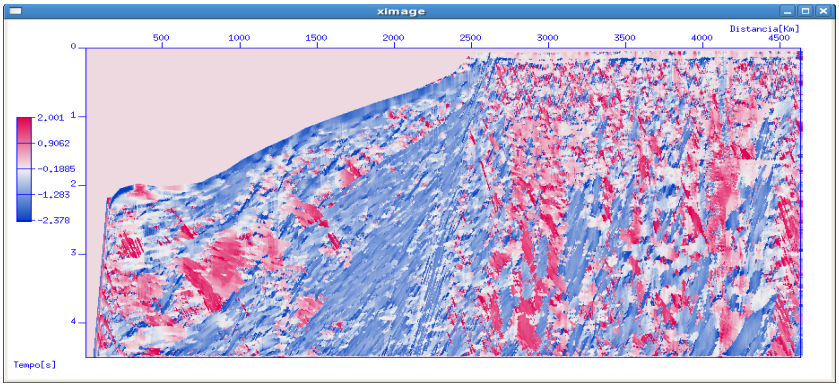

Figure 17: Angle panel of the real data.

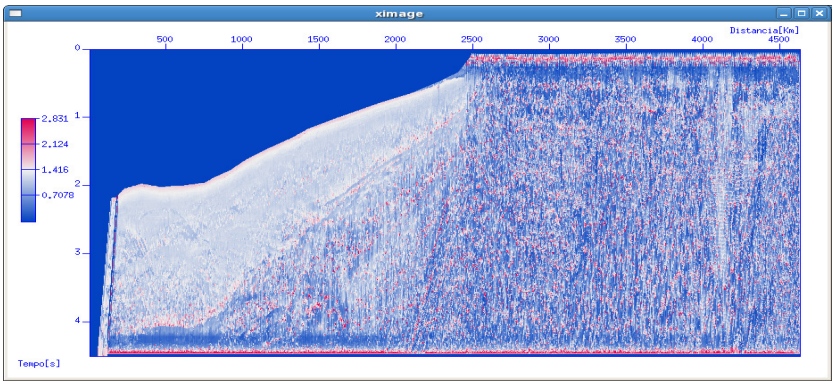

Figure 18: Rnip panel of the real data.

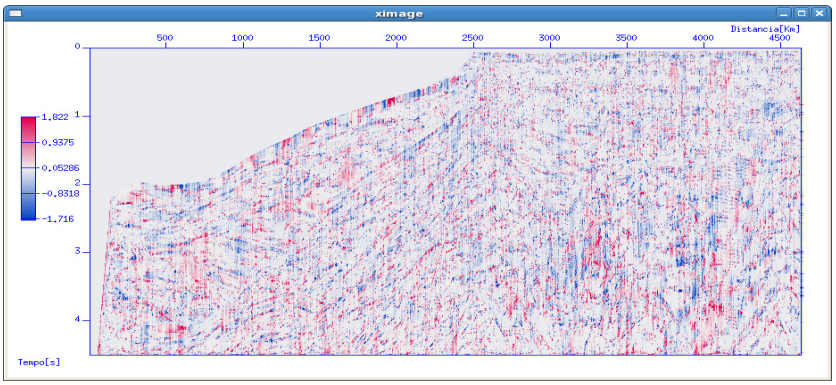

Figure 19: Rn panel of the real data.

that don't get stuck in the loop can continue processing without waiting for the stuck task to finish. CRS Office will be a multi-threaded application, what will permit us to take full advantage of the Java language. Our purpose is to allow the threads to put locks on shared resources so that while one thread is using data no other thread can touch that data. This will be done with synchronization.

The Java Native Interface (JNI) is a powerful feature of the Java platform. Applications that use the JNI can incorporate native code written in programming languages such as C and C++, as well as code written in the Java programming language. The JNI allows programmers to take advantage of the power of the Java platform, without having to abandon their investments in legacy code. Because the JNI is a part of the Java platform, programmers can address interoperability issues once, and expect their solution to work with all implementations of the Java platform.

CONCLUSIONS

Scientists are generally not so much involved in establishing software concepts for scientific problems, they are more interested in using the software to obtain results. CRS Office is being designed to be a consistent, simple and robust GUI for the conventional CRS stack and its further developments. Complexity in computer programs is not so welcome, and CRS Office was developed in Java using OOP to reduced complexity by the use of classes, methods and interfaces.

C++ also supports object-oriented approaches, but among languages that support object-oriented programming and meet other requirements of science and engineering today, Java is the simplest. Beside that, C++ failed in standardize its semantics, and mainly open multi-platform library and generic use. Java platform provides an enormous class library (a set of packages) suitable for use in someone's own applications. Therefore, choosing Java was fundamental for the success of the present project.

In the present stage, CRS Office arises as a useful tool for the conventional CRS stack. Making CRS Office available will help and support students worried only with the seismic processing and forget computational details. It is certainly important to provide graphical interface tools to make processors concentrate in their research work, looking for good imaging of the subsurface based on reflection seismic information.

ACKNOWLEDGMENTS

The authors would like to thank the Brazilian institutions PETROBRAS for the research support, to UFPA (Universidade Federal do Pará), to CNPq for the scholarships of Anderson B. Gomes, and to the WIT Group in name of Prof. Peter Hubral. The thanks are also extended to all WIT sponsors and administration.

REFERENCES

Mann, J. (2001). *Common-Reflection-Surface Stack: User's manual to version 4.2*. Geophysical Institute, University of Karlsruhe, Germany.

Philippsen, M. and Zenger, M. (1998). Javaparty - transparent remote objects in java. *Concurrency: Practice and Experience*, 9(11):1225–1242.

Savitch, W. (2006). *Absolute Java*. Addison Wesley, Pearson. 2nd ed.

Schwab, M. and Schroeder, J. (1998). Algebraic java classes for numerical optimization. *Concurrency: Practice and Experience*, 10(11-13):1155–1164.