WAVEPRO: A NEW APPROACH TO THE DESIGN OF DATA PROCESSING WORK FLOWS

Mark A. Gonzales and Christopher J. Young

Sandia National Laboratories

Sponsored by the National Nuclear Security Administration

Award No. DE-AC04-94AL8500/SL08-IRP-NDD02

ABSTRACT

For the most part, both the types and the sequences of signal processing algorithms (work flows) used to process waveform data for the basic nuclear explosion monitoring tasks are well established and efficiently implemented in the major monitoring systems. However, the methods and tools provided to design or modify a workflow, can be improved to make these tasks easier and less time-consuming. In keeping with well-designed commercial software that most of us use on a daily basis, the work flow can be presented graphically, with each signal processing algorithm shown as an icon, and with the data flow indicated by lines linking these together. Even a complex data processing workflow can be understood at a glance, and changing parameters for any of the algorithms can be accomplished by clicking on the corresponding icon to access a well-formatted display with each parameter clearly labeled. We have begun to explore this idea, using modern software design techniques and tools. In this paper, we will present our original project goals, show the design that we developed to meet these goals, and then present and discuss the software package (WavePro) that we have implemented that realizes this design. We show how WavePro can be used to develop work flows for basic data processing tasks, and in turn how these low-level workflows can be cloned and assembled to easily form more complex work flows for an entire network. WavePro is coded in Java so it is platform independent, and it utilizes "drag and drop" technology to enhance the use of the user interface. In addition, the software is being designed as a pluggable framework in order that new signal processing algorithms can be added to the package without redelivery or recompilation of the software. WavePro uses multi-threading to exploit the multi-core technology that is available on virtually all current generation computers. Further, the package is designed to be compatible with distributed computing using the JPPF package, so WavePro can be used to develop efficient implementations for complex, computationally demanding work flows.

OBJECTIVES

The idea behind the WavePro software application came from observing the difficulty that many researchers encounter in setting up and testing even relatively simple signal processing algorithms using existing software. Though there is high-quality software available to *execute* data processing workflows for monitoring, to our knowledge there is no software designed to aid in the *development* or *modification* of such workflows.

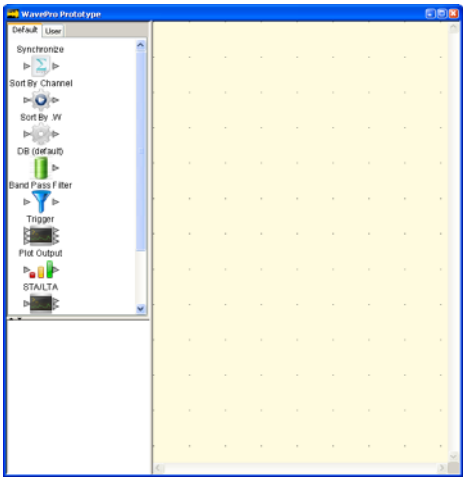
Our WavePro development has two major goals that we believe are essential for the software to prove useful. First, the WavePro application must include a framework to allow for new software modules to be added to the existing application without recompiling or redistributing the existing WavePro software. A software framework that allows for the dynamic adding of software in this manner is called a "pluggable" framework and the software modules added are called "plug-ins". Imagine an application with a software framework in which new signal processing algorithms could be added dynamically as a plug-in and work seamlessly within that application. Throughout this paper we will refer to a plug-in that is a signal processing algorithm as a code module. The second WavePro goal is to allow for signal processing workflows to be created and modified by a user in an intuitive manner without knowledge of a programming language; i.e. a user can arrange, integrate and interchange, via a user interface, well-defined code modules to achieve their signal processing needs.

RESEARCH ACCOMPLISHED

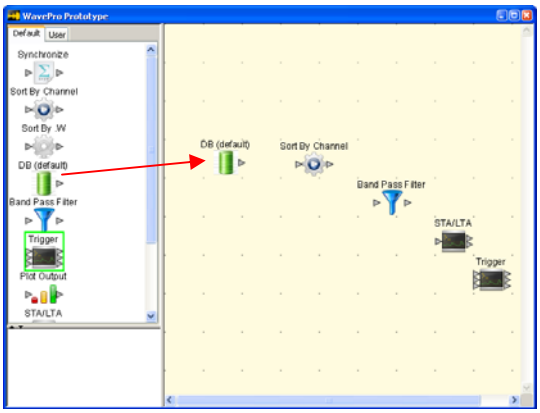
Creating a Graphical Workflow

Currently, the methods and tools provided to design or modify a workflow are much more difficult and time consuming than need be. In a typical current generation system, the users must navigate their way through a confusing hierarchy of directories to find the appropriate parameter files (often poorly indexed), which then must be brought up in an editor and scanned to find the parameters of interest. In an effort to improve this process, the WavePro application represents workflows graphically with signal processing algorithms (code modules) represented as icons that can be connected to form workflows. Using WavePro, a typical workflow design process would be as follows:

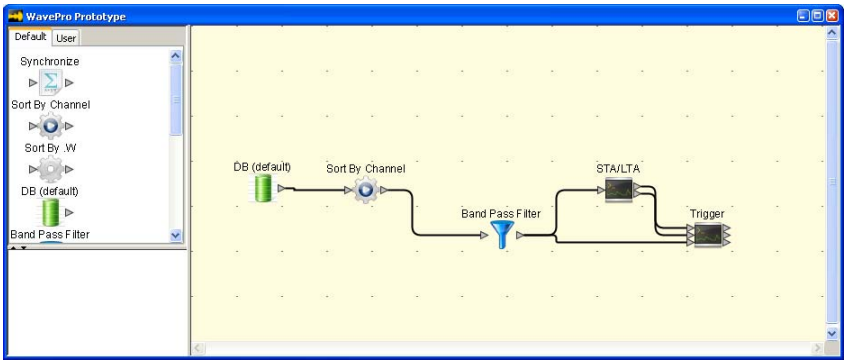
1. Start the application. The code modules are automatically loaded into framework and displayed as icons in a panel to the left.



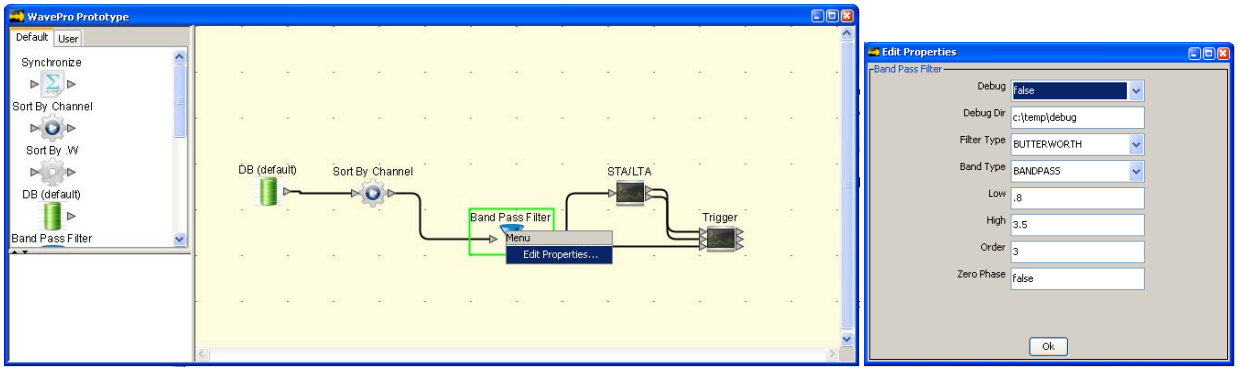
2. Build workflow by dragging icons over to workspace.



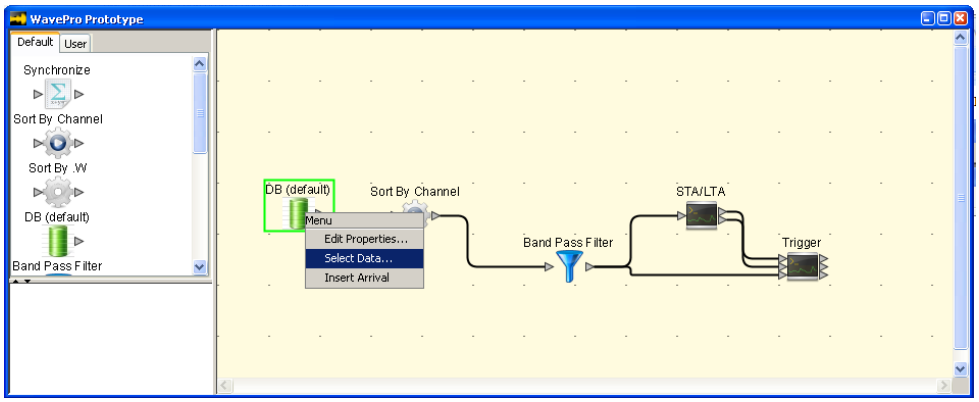
3. Connect code modules by selecting output and dragging to an input to form a workflow.



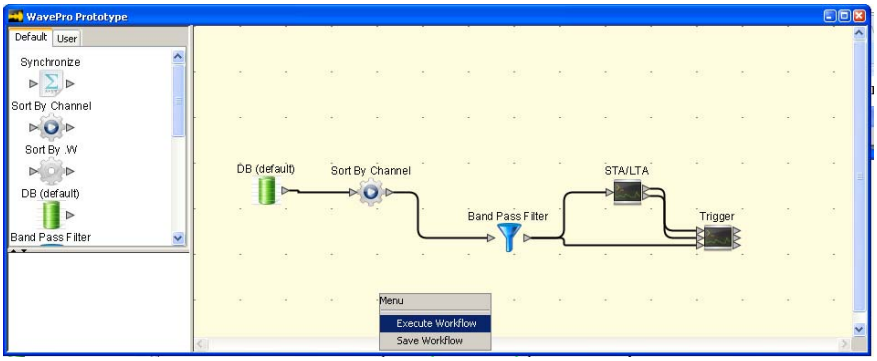
4. Modify algorithm parameters by right clicking on individual icons.



5. Select waveform data by right clicking on data source icon.

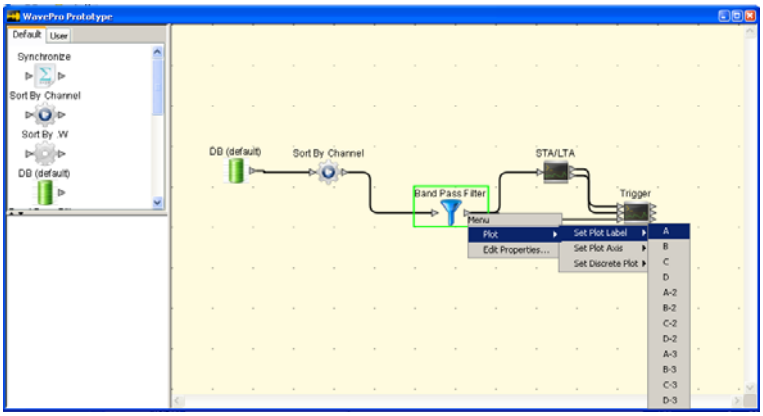


6. Execute workflow by right clicking workspace.

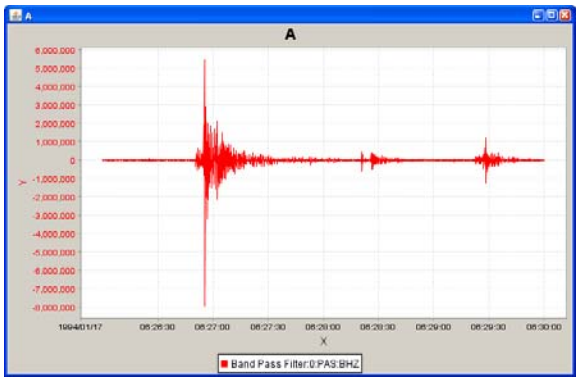


Analyzing A Workflow

To understand whether a workflow is working as intended, the researcher may have a need to view the output of the algorithms at various points. Our application allows for picking "plot points" within the workflow to view data. The researcher selects the output port of the signal processing icon and selects a plot point. For example, in the workflow shown below, the user selects the "Band Pass Filter" icon to send the output of the filter step to a plot.



When the workflow is executed, the application produces the specified output plot.



The concept of picking plot points within the workflow is just one idea of how to help researchers analyze a workflow. Other concepts within the WavePro framework include: plotting multiple plot points on the same graph (e.g., original waveform and STA/LTA output), dumping output at specific points within the workflow to files and databases, easily modifying workflows by deleting links between icons, replacing signal processing icons with different ones by deleting and dragging a new icon over, and saving workflows to a library in order to recall them for later use.

Pluggable Framework

In addition to the suite of algorithms available within WavePro, the application can be readily extended to allow researchers to develop their own signal processing algorithms and to automatically add them for use. The extendibility capability is possible because the WavePro software is implemented as a pluggable framework.

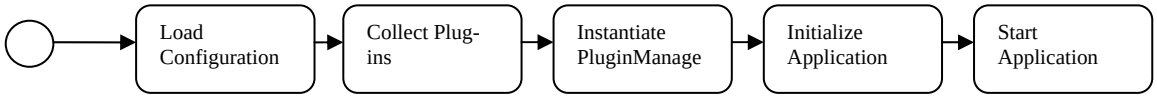
The goal of using a pluggable framework is to make your application extendable by adding functionality to already existing functionality. For example, suppose that a software package had an existing signal processing workflow with a Band Pass Filter and an STA/LTA processor. The pluggable framework will allow for a third algorithm, e.g. a Detector, to be developed outside of the existing software and added to the framework so that it can become part of the workflow. The extendibility of the software allows researchers to develop individual signal processing algorithms outside of the framework and be able to plug those algorithms into the framework. This gives researchers the ability to develop algorithms without having to worry about how to connect them within a workflow. Thus, the researcher could focus the majority of their time developing the algorithm, not on worrying how to integrate it into WavePro. In order for researchers to be productive using this method, the process of adding functionality must be easy to use.

WavePro utilizes an open source project called Java Plug-in Framework (JPF) to create the plug-in capability. JPF uses a concept of extension points and extensions. Extension points are the parts in the main software framework where new software can be inserted. The main software framework must define its own extension points. An extension is the software that exists outside of the main software framework that can be inserted into the extension point. Any piece of software that is an extension is called a plug-in. Currently, the WavePro software framework defines three extension points: *CodeModule*, *DatasourceModule*, and *ExecuteModule*. The *CodeModule* extension point is the interface that allows individual signal processing algorithms to be inserted as pieces of software called *code modules*. In addition to having the ability to plug-in signal processing algorithms, there are other software modules that are needed to process data. In order to process data, you need to get the data from some source. Therefore, the extension point *DatasourceModule* is the interface used to insert software modules that are designed to obtain waveform data. Currently, the WavePro software framework defines default *DatasourceModules* that get waveform data from databases and files. The final extension point defined is the *ExecuteModule*. The *ExecuteModule* interface is used to insert software that controls the execution of the workflow. This includes sorting the data received from the *DatasourceModule*, executing the code modules in the workflow, and receiving results from execution of the workflow. The *ExecuteModule* extension point was added as a need to interchange the way data is sorted and passed through a workflow. For example, the current WavePro application has three default *ExecuteModules*. There is an *ExecuteModule* that sorts data by station and channel, an *ExecuteModule* that sorts data by Wfdisc file, and an *ExecuteModule* that sorts data by breaking up the data into time segments. Using these

three extension points it is possible to create a fully function workflow that obtains signal data, sorts the data, processes the data and obtains the results.

Extending JPF

As mentioned above, the process for the WavePro software framework to start up, find plug-ins and automatically insert them into the framework must be simple. JPF uses a boot library that follows the following flow chart.



During the Load Configuration phase, the framework loads a configuration properties file. In the properties file, the user specifies the location of where to look for plug-in software. The Collect Plug-ins phase uses an XML Plugin Collector to locate the plug-in software and attempts to load it into the framework. The XML Plugin Collector is able to load plug-ins by searching for an xml manifest file. The xml manifest file is deployed with the plug-in software and contains all the metadata used to define the plug-in. The xml file defines the location of the plug-in framework file (a Java file needed for configuration), the location and definition of the plug-in extension point file (i.e., the signal processing algorithm software), and all of the library files that the plug-in will need to work.

While this type of configuration works, it is not simple. In order to simplify, the WavePro application extends the JPF framework by creating and using its own Annotation Plugin Collector in place of the JPF XML Plugin Collector. The Annotation Plugin Collector is similar to the XML Plugin Collector except that it finds plug-in software by searching the Java binary code for Java Annotations. Java Annotations are special metadata embedded in the Java code, hence eliminating the need for an xml file. In addition, the WavePro application takes advantage of object oriented techniques to combine the plug-in framework Java file and plug-in extension point file into one Java file. Therefore, by putting the Java annotations into the plug-in extension point file, the configuration for creating a plug-in requires only one file instead of three. For example, researchers can create a signal processing algorithm plug-in using one Java file similar to the one below:

```
@Module(type="CodeModule", name="Norm", iconName="filter2.png",
version = "1.0.0")
Public class NormalizeModule extends WaveformProcessor {
        @Override
        public void execute() throws ExcecuteException {
            ...signal processing code goes here
        }
}
```

1. Java annotations replace xml file
2. Object oriented extension allows for one file.
3. Custom user-provided algorithm.

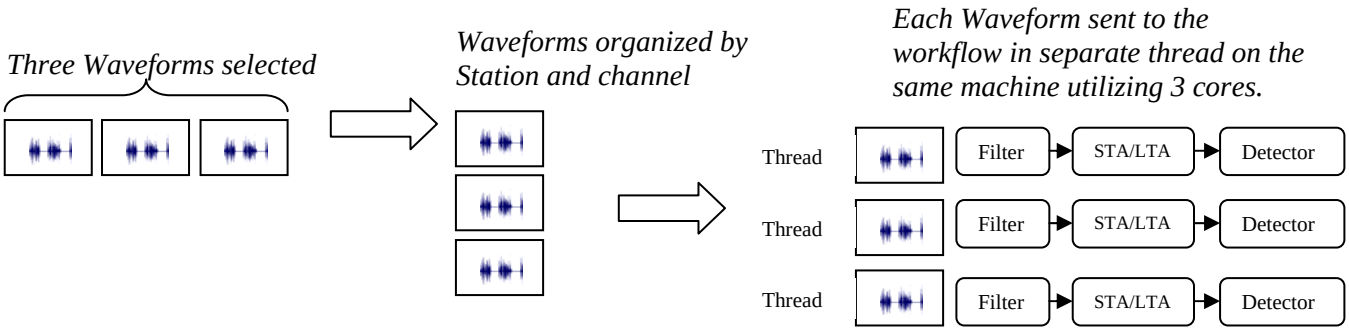
Notice the "type" annotation variable specifies that this software is to implement the extension point CodeModule as defined by the WavePro framework, thus making it a signal processing algorithm plug-in. In addition, the annotation variables define the name and icon to be displayed and the version of this particular plug-in.

Therefore, with one Java source file it is possible to create a signal processing plug-in that can be easily added to the WavePro framework and inserted into workflows. With this type of system researchers can create, modify, and test algorithms quickly.

Exploiting Multicore Technology

In recent years there has been a shift in hardware processor design research and development away from increasing clock speeds and toward expanding the number of cores per processor. One of our design goals for the WavePro application is to exploit this new multi-core architecture. WavePro is being designed to execute workflows in parallel using the concurrent (i.e., multi-threaded) capabilities within the Java programming language, as well as using distributed systems such as the Java Parallel Processing Framework (JPPF) that can execute on more than one

machine. While work on executing software in parallel is ongoing, the WavePro software currently implements a concurrent and distributed system. For example, WavePro has default *ExecuteModules* that work in the following manner on a single multi-core machine:



Hence, the parallelism occurs because the data is broken up into pieces sending each piece of data to a different thread then each executes at the same time among multiple cores. The same kind of parallelism can be achieved on a grander scale by using a distributed system such as JPPF. In this case the data is organized by station and channel and sent, via JPPF, to multiple cores on multiple machines. This approach is similar to the concurrent solution, but can allow for much more data to be processed in a similar amount of time.

CONCLUSIONS AND RECOMMENDATIONS

The purpose of WavePro software development project is to address the need for software specifically to aid in the design and maintenance of data processing workflows for nuclear explosion monitoring. Due to improvements in computer hardware, a robust signal processing workflow for a large networks of stations (such as the International Monitoring System) that a decade or so ago would have required processing on several workstations can now be easily executed on one or two commodity machines. However, the process involved in setting up such a workflow has not kept pace with other improvements. This project seeks to remedy that situation.

Our WavePro development so far has succeeded in achieving our two primary design goals:
1) WavePro allows signal processing workflows to be created and modified by a user in an intuitive manner without knowledge of a programming language. WavePro is designed to present the work flow graphically, with each signal processing algorithm shown as an icon, and with the data flow indicated by lines linking these together. Even a complex data processing workflow can be understood at a glance, and changing parameters for any of the algorithms can be accomplished by clicking on the corresponding icon to access a well-formatted display with each parameter clearly labeled.
2) WavePro includes a framework to allow for new software modules to be added to the existing application without recompiling or redistributing the existing WavePro software. This sort of framework is known in computer science as a "pluggable" framework and the software modules added are called "plug-ins".

ACKNOWLEDGEMENTS

We thank all of our GNEMRD colleagues at SNL for their helpful feedback on WavePro throughout the design and testing process.

REFERENCES

Java Plug-in Framework . Retrieved from http://jpf.sourceforge.net/

Java Parallel Processing Framework. Retrieved from http://www.jppf.org/