REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
1/25/05

3. REPORT TYPE AND DATES COVERED
final report, 27 Aug 2001 - 29 May 2004

4. TITLE AND SUBTITLE
Efficient Integration of Old and New Research Tools for Automating the Identification and Analysis of Seismic Reference Events

5. FUNDING NUMBERS
DE-FG02-01ER83218

6. AUTHOR(S)
D. Wilmer Rivers, Jr. and Robert A. Wagner

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Multimax, Inc
1441 McCormick Drive
Largo, MD 20774

8. PERFORMING ORGANIZATION REPORT NUMBER
MM-05-ISD-02

9. SPONSORING / MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
U.S. Department of Energy
Chicago Operations Office
9800 South Cass Avenue
Argonne, IL 60439

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Any single computer program for seismic data analysis will not have all the capabilities needed to study reference events, since these detailed studies will be highly specialized. It may be necessary to develop and test new algorithms, and then these special codes must be integrated with existing software to use their conventional data-processing routines. We have investigated two means of establishing communications between the legacy and new codes: CORBA and XML/SOAP Web services. We have investigated making new Java code communicate with a legacy C-language program, geotool, running under Linux. Both methods were successful, but both were difficult to implement. C programs on UNIX/Linux are poorly supported for Web services, compared with the Java and .NET languages and platforms. Easier-to-use middleware will be required for scientists to construct distributed applications as easily as stand-alone ones. Considerable difficulty was encountered in modifying geotool, and this problem shows the need to use component-based user interfaces instead of large C-language codes where changes to one part of the program may introduce side effects into other parts. We have nevertheless made bug fixes and enhancements to that legacy program, but it remains difficult to expand it through communications with external software.

14. SUBJECT TERMS
seismic analysis, inter-process communications, distributed applications, CORBA, Web services, XML/SOAP, metadata, *geotool* program

15. NUMBER OF PAGES
183

16. PRICE CODE

17. SECURITY CLASSIFICATION
UNCLASSIFIED

18. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

20. LIMITATION OF ABSTRACT

SUMMARY

The selection and study of reference events for inclusion in the National Nuclear Security Administration (NNSA) Knowledge Base (KB) requires the application of a much broader suite of seismic analysis software than does either the routine production of a seismic bulletin or the subsequent preliminary screening of those bulletin events to conduct nuclear monitoring. For either of the latter applications, a large program designed explicitly for that single purpose can perform all the tasks that must be applied to the many events that will be processed daily, whereas all the different measurements and algorithms that a scientist may wish to apply to a candidate reference event cannot be anticipated in advance and incorporated into a single large program. A scientist studying candidate reference events is therefore more likely to need the capabilities found in many separate specialized programs, and it may in fact be necessary to develop new software to perform specific analyses that are modified or designed just for the particular events under examination. All these applications, including the developmental software that is still in the testing phase, must then be made to work in tandem so that they can be applied to the data set under examination.

To link together the different applications (which may be running in separate execution threads on a single computer or on separate computers, perhaps even under different operating systems), it is necessary to implement a middleware layer and then modify the applications so that they can communicate through it. There are two principal approaches to establishing the inter-process communications, and we have examined both of them. One is the tight coupling of one application to another through socket-based Remote Procedure Calls that are implemented directly in code or within Common Object Request Broker Architecture (CORBA) or Java Remote Method Invocation (RMI) client-server software. The alternative approach is the loose coupling of a diffuse cluster of clients and servers employing a Service-Oriented Architecture (SOA) for communications, especially across an Intranet or Internet, such as "Web services" that exchange both processing requests and data as XML-formatted Simple Object Access Protocol (SOAP) messages transmitted by HTTP.

In this project we have assumed that the familiar program *geotool,* for the interactive display and processing of seismic data, will be the cornerstone program in a reference event analysis system. Because *geotool* is built using an early 1990s architecture consisting of a single large C-language program running within UNIX (or Linux) that invokes operations through callback functions from Motif widgets, it is a difficult program to modify without introducing unanticipated software "side effects" into parts of the code that were not themselves directly modified. We have met with mixed results in attempting to make *geotool* flexible enough to allow it to request services from other programs by acting as a so-called "fat client" (*i.e.*, a program that does most of the data processing itself and relies on server programs only for specialized tasks). A preferable approach would be to employ a "thin client" program that serves mainly as a user interface (like a graphics terminal), but it would be better still to retain much of the client-side functionality of *geotool* by breaking it up into individual components that could be modified independently of one another, thereby minimizing the side effects introduced by those modifications. Modern software architectures as such J2EE and .NET rely on client and

server programs that are developed using this approach. A promising future platform for building a reference event analysis system comprising individual software components for different seismic data processing tasks is the next version of the Windows operating system, which is built upon a middleware layer called “Indigo” that blurs the distinction between a tight client-server coupling and a loose coupling of Web services by employing SOAP messages for both types of inter-process communications.

ACKNOWLEDGEMENTS

We wish to thank our collaborators at Lawrence Livermore National Laboratory, specifically Craig Schultz, Douglas Dodge, and Gregory Pope, for producing a requirements document showing software that will be needed for a reference event analysis system and showing bug fixes and enhancements to the program *geotool* that are needed not just for a reference event system but for the application of *geotool* per se to seismic data analysis, as it will be necessary to do to commercialize the results of this Small Business Innovation Research investigation. We are especially indebted to Floriana Ryall of Johnson Controls for her diligence and thoroughness in discovering shortcomings of *geotool.* We are also indebted to the software development staff at Multimax for their hard work and resourcefulness in their efforts to address those shortcomings.

TABLE OF CONTENTS

1. BACKGROUND

Monitoring the globe to detect, locate, and identify possible nuclear explosions requires an extensive database of information about known seismic events (*i.e.*, those which have been located reliably and for which their nature, whether a particular type of earthquake or chemical explosion, is presumed known either by detailed studies of locally recorded seismograms or by ancillary information such as on-site geological investigations, satellite photography, or perhaps simply the examination of the industrial logs of blasting conducted at rock quarries) in as many different geographic and geologic settings as possible. This extensive database is needed because the characteristics displayed by seismograms are strongly influenced not only by the nature of the seismic event but also by the event's geologic setting and the seismic waves' propagation path from the source to the receiver. It is therefore important to have a reference database containing information characterizing different types of events (*e.g.*, chemical explosion with various geometries of ripple-fire patterns relative to the spatial distribution of recording stations, shallow strike-slip earthquakes, deep dip-slip earthquakes, etc.) in as many different geologic settings (rock type, anelastic attenuation, proximity to local scatterers of seismic waves, etc.) for as many different source-to-station wave propagation paths as possible. These seismograms should then be studied in detail so that seismograms from newly detected (and hence unidentified) events can be compared with them. Such a database of reference events is therefore an important component of the Department of Energy National Nuclear Security Administration (NNSA) Knowledge Base (viz. Young et al., 2003). For instance, if the Knowledge Base records that for a particular type of event in a particular region as detected at a particular station the ratio of the spectral amplitude of the *Pg* wave in the 6-10 Hz band to that of the *Lg* wave in the 2-4 Hz band exceeds some threshold, that characteristic will aid in the identification of other events of that same type in that same region as detected at that same station. What is thus required is a powerful and set of tools for studying seismic data in sufficient detail that they may then be used as reference events in the analysis of new data.

Unfortunately, no single piece of software is adequate for this purpose, and furthermore none ever can be, since it is impossible to foresee all the possible scientific investigations than can be performed on a seismogram (which, as we have pointed out, will be highly variable from case to case) and since new research results will always eventually lead to new techniques for studying reference events. This is an important difference between a reference event analysis system and the existing software systems that are employed in an operational setting for the routine analysis of new data to produce a daily bulletin of seismic activity as a means for monitoring the test ban: an operational system can (and perhaps should) be limited to employing a specific set of tools, often in a routine sequence of actions specified in advance and carried out on a large data set, whereas a reference event analysis system will of necessity be open-ended, since scientific research leads wherever it may lead, often in unforeseen directions that require inventive new tools. The reference event research system will be therefore be designed to perform as many different operations as possible in whatever order is deemed most appropriate, although perhaps on a much smaller data base (consisting of only a few events from the same region and/or exhibiting some particular characteristic) than the one that is processed for routine seismic event bulletin production.

Large and powerful seismic analysis programs such as geotool, SAC, ARS, etc., all offer a variety of useful tools, but each one offers certain capabilities that the others lack, and all fail to provide the capabilities that are offered by specialized programs that perform only a specific type of analysis in great detail. These specialized tools are frequently developed by investigators who would like to add those capabilities to the existing software programs (once they have been tested against them), but they do not want to have to re-write the large programs in order to add these specialized routines to the pre-existing suite of capabilities that those programs offer. What is therefore required is a system architecture that allows the existing seismic analysis programs as well as newly developed specialized ones to cooperate with one another, effectively creating a "super-program". This linking of the separate programs into an integrated whole is more useful than the sum of its constituent parts, since it is the interplay between the different modes of investigation, and not just results obtained by each technique separately, that will prove most useful to a scientist attempting to identify and characterize reference events.

2. OBJECTIVE OF THIS STUDY

As is explained above, scientists use a variety of stand-alone computer programs to analyze and identify seismic events for the monitoring of possible nuclear explosions, and an especially wide selection of software tools is needed for the detection and intensive study of reference events that will be included in the NNSA Knowledge Base for use in future event comparisons. Some of these stand-alone programs are large software packages that offer many tools for routine seismic analysis, but they are difficult to modify to include additional tools for specialized tasks. Others of these stand-alone programs offer the capability of performing only specific operations, and they must be used in conjunction with other programs such as interactive waveform graphics displays that offer more general analysis capabilities. In most cases neither the large software packages nor the specialized analysis programs can communicate adequately from one to another without the tedious creation and input of temporary data files and other awkward techniques. Since the stand-alone programs cannot exchange data easily, it is difficult to use them in a data-processing pipeline that could add new capabilities to those offered by the large software packages or that could allow the specialized analysis programs to rely on other software for tasks such as graphics displays.

A reference event analysis system should therefore be built by using a system architecture that facilitates data flow among these stand-alone programs, including any new programs that will be developed in the course of future research and that may be especially valuable for the identification and characterization of reference events. This new architecture should allow the results of one program to be sent easily to another one, as chosen on a case-by-case basis by the seismic analyst, without the creation of temporary files and database tables. Because many of the stand-alone seismic analysis programs that need to communicate with one another are written in different computer languages, and many are written for use under different operating systems, it will be important for this architecture to be as nearly platform-independent as possible. Furthermore, the communications among the separate programs should allow access to remote resources for data retrieval or specialized computations. The reference event analysis system should therefore be constructed as a distributed system of individual software components rather than as a single large software package. The first objective of our study was to examine software architectures and communications protocols that will allow existing and newly developed programs to be used as the components in this distributed system. Our next objective was to modify one of those programs, *geotool*, so that it can be integrated into a reference event analysis system using that distributed system architecture.

3. INITIAL SYSTEM ARCHITECTURE INVESTIGATION

The first phase of work was intended to explore the architectural underpinnings of a software system for analyzing seismic reference events, and it was not intended in this first stage to design and construct any of the graphics, signal processing, geophysical, communications and other modules that would be the constituent parts of such a system. Our initial study focused not on the construction of a new seismic analysis package that is intended to replace the ones currently in use; rather, it focused on the integration of separate modules, including both existing code and newly developed routines, into a flexible software system.

In examining possible architectures for a reference event analysis system, our choices were guided strongly by the differences between routine seismic analysis performed for constructing event bulletins and the more nearly *ad hoc* analysis that is required to identify and characterize reference events for inclusion in the Knowledge Base. As noted above, software packages for routine seismic analysis are designed to facilitate the repetitive performance of a pre-determined series of tasks. Although a data analyst engaged in constructing an event bulletin will have some flexibility in choosing to perform certain ones of those analysis tasks for every waveform under consideration while reserving other analysis tasks for use only in unusual circumstances (such as using a polarization filter to help separate the sources of mixed signal arrivals), by and large the processing of every new event will conform to set procedures. In particular, if the analyst makes any choice of data processing tools at all, the selection of those tools will be limited to only those modules that are included within the seismic analysis software package, so the designer of that package must therefore foresee all the tasks that the analyst may need to perform. For screening all seismograms recorded by a network and pre-processed by an automated system, and especially for making a specified set of measurements that are agreed upon for international data exchange, this approach is in fact a suitable one. A number of software packages such as Seismic Analysis Code (SAC2000; *viz.* Goldstein, 1998), Analyst Review Station (ARS; *viz.* Wang, 1996), *geotool* (Henson, 1993), and MatSeis (Young, 2001), among others, are commonly used for these analysis tasks. These same packages can also be used satisfactorily for the analysis of reference events, but we feel that the use of any single one of these packages, or for that matter the use of any other single large program that we could design and build *ab initio* as a product of our current study, may not be the best possible approach for this purpose. We shall now discuss some shortcomings of that approach and describe alternatives to it.

For identifying and characterizing possible reference events for use within the Knowledge Base, it is critical that the scientist have maximum flexibility in choosing what algorithms and data processing routines should be used in any particular case. The purpose of a reference event is to exemplify the seismograms that are to be expected from a particular type of event, in a particular region, as recorded at particular stations, and the tools needed to characterize those seismograms will be as varied as the waveforms themselves. A wavetrain from a small strike-slip earthquake in the upper mantle in a region of low anelastic attenuation will of course be markedly different from the waveform of a large thrust-fault earthquake at shallow depth in a tectonically active region, and a scientist may wish to choose to apply a different suite of tools depending on certain features that are exhibited by one of these events but not by the other. In fact, highlighting these differences in order to categorize patterns that can be used to distinguish one type of event from another is one of the goals of reference event analysis. Even for conventional applications such as picking arrival times, determining epicenters and confidence regions, measuring signal amplitudes and periods, etc., when studying a reference event a scientist may well wish to apply unusual tools that would not conventionally be applied during routine event screening. For instance, elaborate de-ghosting algorithms may be used to identify multiple small echoes within reference signals propagating along paths characterized by strong multipath arrivals, a procedure that is unlikely to be applied to signals not intended for inclusion in the Knowledge Base. It is even possible that a scientist may wish to develop special software for application to a particular signal, if no tool that is available seems appropriate for the purpose. Applying specialized tools would require making modifications or extensions to the seismic analysis software packages. Another reason that these packages could need to be modified for reference event analysis is that it can be the software itself, rather than the seismogram, that is the subject of the analysis. This would happen because a scientist who is developing a new analysis routine will likely want to test it by applying it to reference signals in conjunction with the existing tools that already exist in those packages. For instance, a scientist who has developed some variant on the conventional computation of the complex signal cepstrum will want to test only that algorithm and not have to write new graphics code for the display of the waveforms, spectra, cepstra, etc., capabilities that are already offered by the existing seismic analysis packages. For processing reference events, then, it will be necessary for a scientist to make modifications to SAC, ARS, *geotool*, etc., by incorporating unusual or newly developed analysis tools into the software package.

Although most of these seismic analysis software packages can be modified to some extent, the process is usually not an easy one. In some cases modifications can be made to the software through the resource files it processes as data, and in other cases new libraries can be linked in and operated from existing callbacks, but often it is necessary to modify (and thus have access to) the source code to make some desired changes. Even when it is possible to do this, it is dangerous to do so, since modifying code in the huge single programs that form the basis of most of these packages is liable to result in unanticipated "side effects" that cause existing functionality in the program to break or to behave in a different manner than before. Another major problem is that often a scientist who is working with a large seismic analysis program written in the C programming language, for example, will want to implement a processing routine that is written in Java or some other language. Although software "pragmas" (as they are called in some languages, such as Ada) can be constructed to invoke routines written in one language from main programs written in another, the process is not so straightforward that a scientist or data analyst could rig it together in short order as an improvised addition to the analysis of a specific event. Scientists who are running a program like *geotool* that is intrinsically hard-coded with calls to a particular graphics platform (in this case, Motif) are furthermore constrained to work within a particular operating system such as UNIX and therefore cannot take advantage of seismic analysis codes written to run on another platform such as Windows. To overcome these problems associated with adding functionality to large software systems for seismic analysis, we have decided to take a different approach to constructing the reference event Analysis System. Instead of augmenting an existing large program, or writing a new one that would be more nearly comprehensive but that would itself eventually become inadequate when future research results in new analysis algorithms and new software tools, we have chosen to implement a distributed architecture that will permit existing code as well as new code to be invoked remotely from the large programs currently used for seismic analysis.

4. USE OF CORBA FOR DATA COMMUNICATIONS

The distributed system architecture that we investigated during the first phase of this project is one that is based on CORBA (Common Object Request Broker Architecture), a technology that was developed in the mid-1990s for client/server data communications between desktop computers and corporate mainframes. Our scheme for implementing CORBA as the backbone for a distributed seismic analysis software system is illustrated in Figure 1. We are using the C-language program *geotool* as the server software, since it manages the system of ASCII files (and/or the Oracle database) and thus operates as a data server. Although *geotool* runs under the Linux operating system, for our proof of concept we used a simple Java program called "WaveformViewer" as a client application, running under the Windows 2000 operating system on a separate computer within the same LAN. As its name implies, WaveformViewer does little more than display waveforms using Java graphics, but it does offer a limited functionality such as digital filtering that will suffice to demonstrate how the reference event Analysis System should work. WaveformViewer is in fact assembled from only a few of the many Java classes that make up a much more comprehensive seismic analysis system (Henson *et al.*, 2000), but rather than use that full system, thereby linking together two large programs that offer many duplicated capabilities, we wish to show how a single server can drive data analysis by communicating with a number of small client programs each of which perform only a small number of particular tasks, perhaps only a single function. This is the system design that we consider the best candidate for use in the reference event Analysis System. Examining the configuration shown in Figure 1 points out a significant difficulty with our initial architecture, however: more properly *geotool* should be a client application, since it acts as a Graphical User Interface, and it should call WaveformViewer by accessing a Java application server, since it delivers a service (in this case, filtering a waveform) to the *geotool* client. This reversal in roles is a problem to which we shall return later.

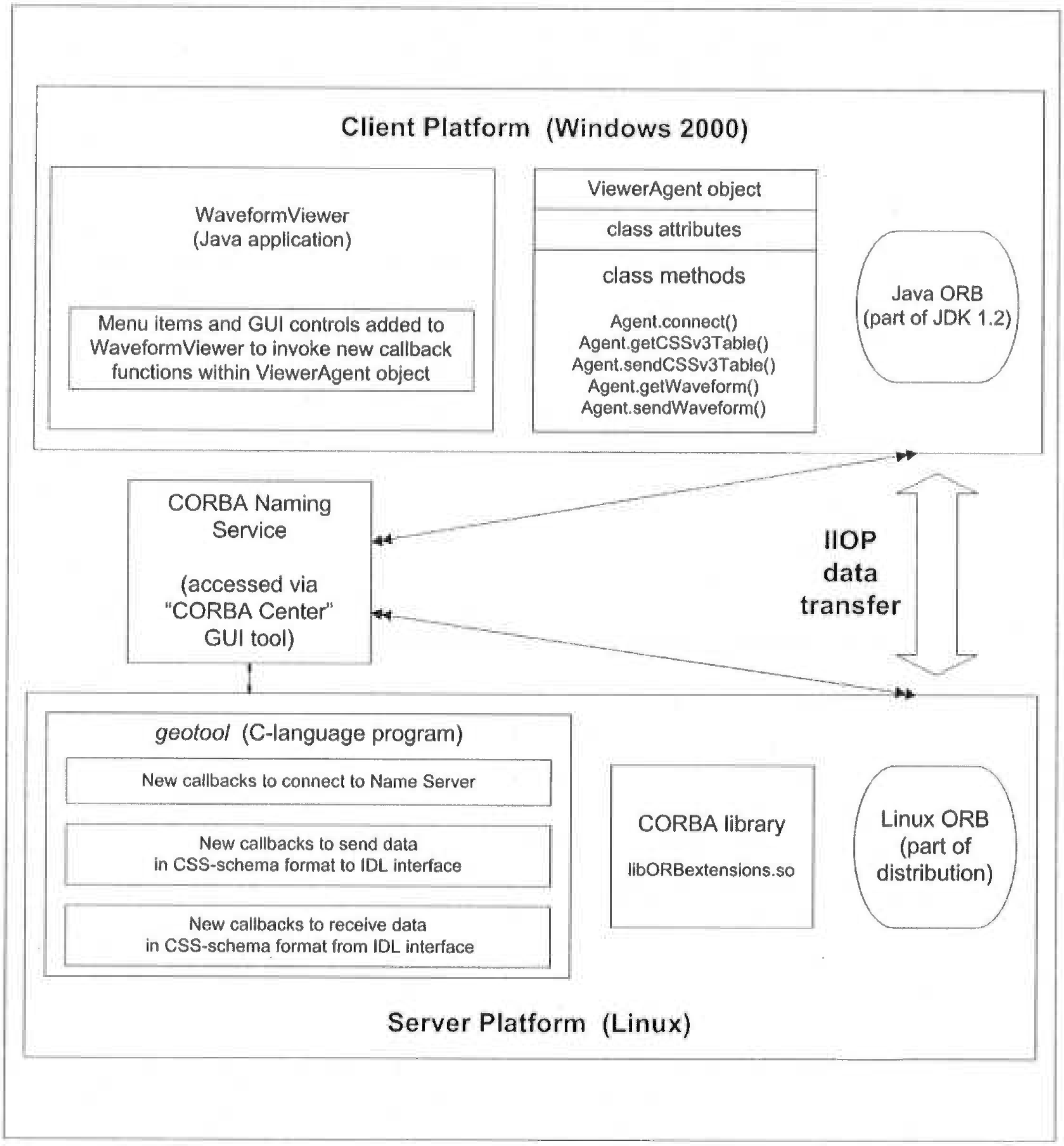

Client Platform (Windows 2000)
WaveformViewer
(Java application)
Menu items and GUI controls added to WaveformViewer to invoke new callback functions within ViewerAgent object
ViewerAgent object
class attributes
class methods
Agent.connect()
Agent.getCSSv3Table()
Agent.sendCSSv3Table()
Agent.getWaveform()
Agent.sendWaveform()
Java ORB
(part of JDK 1.2)
CORBA Naming Service
(accessed via "CORBA Center" GUI tool)
IIOP data transfer
geotool (C-language program)
New callbacks to connect to Name Server
New callbacks to send data in CSS-schema format to IDL interface
New callbacks to receive data in CSS-schema format from IDL interface
CORBA library
libORBextensions.so
Linux ORB
(part of distribution)
Server Platform (Linux)

Figure 1. Using CORBA to enable client/server data communications between the C-language program *geotool*, running as a data server under Linux, and the Java program WaveformViewer, running as a client application under Windows 2000. (Ideally, however, *geotool* should be a client application, since it acts as a Graphical User Interface, and it should call WaveformViewer by accessing a Java application server, since it delivers a service to *geotool*.) The data communication takes place between the Object Request Brokers on each platform via the Internet Inter-ORB Protocol, which is an Internet standard. We have modified the client code and the server code so that they link to CORBA, and that link can now be used to permit data communications between the sever and additional client applications that are invoked by the user through items added to the *geotool* pull-down menus. The data that are exchanged between the client and server programs are translated from the standard CSS database table schema to CORBA Interface Definition Language. We have written a utility called "CORBA Center" that makes it easier for the user to set up the client/server link (which requires the *geotool* server to register itself with the CORBA Name Server).

As is shown in the diagram in Figure 1, we have modified *geotool* by adding new callback functions that allow it to communicate with an ORB (Object Request Broker), a commercial software program that operates as "middleware" in the CORBA data communications design. Since *geotool* is running under Red Hat's distribution of Linux, we are able to make use of the open-source ORB known as "ORBit" that is bundled as a part of the distribution. For a *geotool* server running instead under Solaris, we could use as an alternative any one of a number of commercial ORBs for UNIX. Using the X Resources file for *geotool* we can then add a new pull-down menu item that will allow the user to send data to our client program, WaveformViewer. To expedite the process (external to *geotool*) of setting up the data connection, we wrote a utility called "CORBA Center" that effectively acts as a switchboard for choosing a server side and a client side of the data flow. This utility registers the IP addresses with the CORBA Naming Service (to associate object references with symbolic names) so that the two programs can exchange data across the LAN as easily as if they were both running on the same computer. CORBA's data communication across the LAN takes place using the Internet Inter-ORB Protocol (IIOP). This is an official Internet protocol (like FTP and HTTP) that allows an ORB on one platform to communicate with one on another platform. Note that it is irrelevant that the server-side software is in C and the client-side software is in Java, since IIOP is platform-independent. We had to modify the pre-existing Waveform-Viewer client application by adding callback functions that allow it to communicate with a new "Agent" Java class, which in turn handles the communication with the ORB that is included within the Java platform under releases of the Java Development Kit (JDK) with version numbers 1.2 and higher. The data messages that are exchanged through this client/server communication consist of seismograms and the metadata that describe them, in conformance with the standard Center for Seismic Studies (CSS) data schema (Carter *et al.*, 2001). To allow this communication, we have translated the CSS-schema data structures such as *.wfdisc*, *.arrival*, *.origin*, etc., into CORBA's Interface Definition Language (IDL). An IDL routine does not know whether the application with which it is communicating is written in C or Java (or one of several other languages). All it knows is that the application on the other side of the data stream expects to receive a message conforming to a particular IDL argument list, and we have therefore translated all the standard CSS data tables into a single IDL interface so that we can re-use that same interface for as many different *geotool* client/server applications as possible.

5. IMPLEMENTING CORBA IN *GEOTOOL*

We have written a graphical interface to the CORBA Name Service and made it accessible by remote Java or C applications. The Name Service makes it possible for the client applications to identify the remote data providers. Here, we have called these applications that provide data that can be accessed by other client applications as "server applications". You can have one or more server applications in one network or even in a single machine. The Name Service application must be running on one machine in the network all the time so that it can be used by these server applications or client applications. This architecture is illustrated schematically in Figure 1, where the CORBA Name Service is show as "middleware" between the server and client applications.

In addition to developing the IDL description of the data to be transferred among applications, we have added a software library called *libORBExtensions.so* to facilitate the use of CORBA in the server-side application. The coding in the server is then less complex than inter-process communication ordinarily is. Most of what the users need to know is how to use the callback functions in *geotool* and what are the data structures in the CSS Schema v3.0. Here we list the functions that are the ones most often used in *libORBExtensions.so*:

```
void Init_Schemav3( void );

void Schemav3_create_service( char *host,
                              int portnum,
                              char *requested_name );

void Schemav3_SetCallback( void *func,
                           int which,
                           int callback_type );

void Schemav3_thread_run_service( void );
```

First, the function Init_Schemav3 must be called before the program uses this library. Then the function Schemav3_create_service can be called to create a name and set it to the Name Server, as is shown in the second function call that is listed above. The parameters used in this function call are:

> *host*: The name or IP address of the machine on which the name server is running. For example, if you have a name server running in a machine with IP address 63.119.210.101 in your network, this parameter will be "63.119.210.101".
>
> *portnum*: The port number in name server machine. The TCP/IP protocol uses different port numbers to listen to the network and establish different network connections. When the name server starts, you will have to give it a port number for network connection. This number can be obtained from the person who administers the name server.
>
> *requested_name*: This name will be stored in the name server and will be requested by the client applications. It is used to identify the server applications. If a client application wants to establish a connection with a server application, it has to request this name from name server, and the name server will then guide it to connect to the proper server application.

Next, the program should make the function call:

```
void Schemav3_SetCallback( void *func, int which, int callback_type );
```

This function is used to set different callback function for different data tables. If the client applications place the requests, the system will call this corresponding callback function automatically. For example, we define a callback function and set it to respond to a data request about the data table *Wfidsc*. When the client application requests the waveform description table *Wfdisc*, the server application will automatically call the function void impl_CALLBACK_get_Wfdisc_vector( ):

```
static void impl_CALLBACK_get_Wfdisc_vector(Schemav3_WfdiscVector * w)
{
        //collect data for the table Wfdisc in Geotool and fill in the structure w.
        // the client application will get back all the data in structure w.
}
Schemav3_SetCallback((void
)&impl_CALLBACK_get_Wfdisc_vector,F_Wfdisc,T
Y_CALLBACK_VECTOR);
```

For the other data tables in CSS Schema v3.0, the user can perform a similar procedure in the same way.

```
void Schemav3_thread_run_service(void);
```

This function tells the system to begin the application data service. The service will start as an individual thread. The user therefore does not have to worry about the main message loop in the system.

Here is the sample code in *geotool* that shows how to response to the "get Wfidsc table data" requests. When *geotool* starts, it will first call the function "StartSchemav3ServiceCB", and this function call will make *geotool* work as the CORBA data provider.

```
// This is function orbCB.c

static long impl_CALLBACK_get_Wfdisc_total( void) ;
static void impl_CALLBACK_get_Wfdisc_vector( Schemav3_WfdiscVector *w );

void StartSchemav3ServiceCB( Widget widget, XtPointer client_data,
            XtPointer callback_data )
{
    Init_Schemav3( ); //initialize before using the lib

// Set the callback function so that client can get the total waves in Geotool
    Schemav3_SetCallback( ( void * ) &impl_CALLBACK_get_Wfdisc_total,
            F_Wfdisc, TY_CALLBACK_TOTAL );

// Set the callback function to response to the "Get Wfdisc" table request from client application.
    Schemav3_SetCallback( ( void * ) &impl_CALLBACK_get_Wfdisc_vector, F_Wfdisc,
            TY_CALLBACK_VECTOR );

// A name server is running in machine with IP address 63.119.210.101 and listening to the port 1077.
// Here, the name "Geotool" is set to the name server
    Schemav3_create_service( "63.119.210.101", 1077, "Geotool" );
    Schemav3_thread_run_service( ); //begin the service and begin to response to the client's requests
    printf( "starting Geotool Corba service, waiting for request....\n" );
}

/* This function returns how many waveforms are displayed in Geotool */
static long impl_CALLBACK_get_Wfdisc_total (void )
{
  Geotool *g;

  if ( ( g = getGeotool( g_widget ) ) == NULL ) {
        printf( "TableListCB: Can't get geotool parent" );
        return 0;
   }
  return ( long ) g->num_wavs;
}

/* This function reads these data in Wfdisc table in Geotool and fills in the structure w.
  The structure w will be returned to the client's application. */
static void impl_CALLBACK_get_Wfdisc_vector( Schemav3_WfdiscVector *w )
{
  Geotool * g;
  int i;
  Schemav3_Wfdisc * wl;
  WFDISC30 * wf = NULL;
  Boolean valid_wf = 0;

  if ( (g = getGeotool( g_widget ) ) == NULL ) {
        printf( "TableListCB: Can't get geotool parent" );
        return;
  }

  if ( g->num_wavs < 1 ) return;
  mallocWarn( &wf, g->num_wavs*sizeof( WFDISC30 ) );
  valid_wf = GetWfdisc30Data( g_widget, wf );
  for ( i=0; i< g->num_wavs; i++ )
  {
// get from the wf
    wl = &w->_buffer[i];
    if ( valid_wf ) {
        if ( wf[i].sta[0] ) strncpy( wl->sta,wf[i].sta, strlen( wf[i].sta ) );
        if ( wf[i].chan[0] ) strncpy(wl->chan, wf[i].chan, strlen( wf[i].chan ) );

        wl->wfid = wf[i].wfid;
        wl->time = wf[i].time;
        wl->chanid = wf[i].chanid;
        wl->endtime = wf[i].endtime;
        wl->nsamp = wf[i].nsamp;
        wl->samprate = wf[i].samprate;
        wl->jdate = wf[i].jdate;
        wl->calib = wf[i].calib;
        wl->calper = wf[i].calper;
        if ( wf[i].instype[0] ) strncpy ( wl->intype, wf[i].instype, strlen( wf[i].instype ) );
        if ( wf[i].segtype[0] ) strncpy ( wl->segtype, wf[i].segtype, strlen( wf[i].segtype ) );
        if ( wf[i].datatype[0] ) strncpy ( wl->datatype, wf[i].datatype, strlen( wf[i].datatype )  );
        if ( wf[i].clip[0] ) strncpy ( wl->clip, wf[i].clip, strlen( wf[i].clip ) );
        if ( wf[i].dir[0] ) strncpy ( wl->dir, wf[i].dir, strlen( wf[i].dir ) );
```

```
            if ( wf[i].dfile[0] ) strncpy ( wl->dfile, wf[i].dfile, strlen( wf[i].dfile ) );
            wl->foff = wf[i].foff;
            wl->commid = wf[i].commid;
            if ( wf[i].lddate[0] ) strncpy( wl->lddate, wf[i].lddate, strlen(wf[i].lddate ) );
        }
    }
    free( &wf );
    return;
}
```

Here is an illustration showing the logic of this sample code.

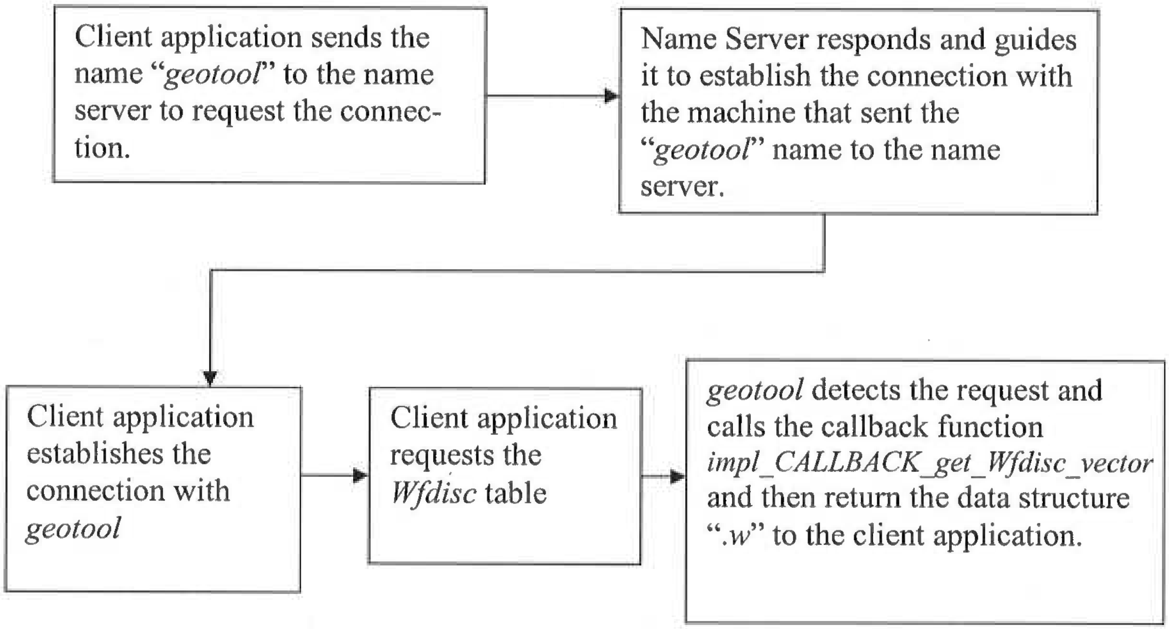

Client application sends the name "geotool" to the name server to request the connection.
Name Server responds and guides it to establish the connection with the machine that sent the "geotool" name to the name server.
Client application establishes the connection with geotool
Client application requests the Wfdisc table
geotool detects the request and calls the callback function impl_CALLBACK_get_Wfdisc_vector and then return the data structure ".w" to the client application.

Figure 2. Flow of logic implementing the data communications between a geotool data server and a client application, using the CORBA interface developed during this project.

In order to do this same procedure with the other data tables in CSS Schema v3.0, one can simply copy the code and add the callback functions for those tables. In this example, there is another callback function impl_CALLBACK_get_Wfdisc_total that returns how many waveforms are shown in *geotool.* It will be called if the client applications make a request for the "total number of waves". The theory underlying this request is the same as that for the *Wfdisc* table.

6. IMPLEMENTING CORBA IN THE APPLICATION "WAVEFORMVIEWER"

For the client application also, the users do not need to be knowledgeable about how to perform coding with CORBA. We have provided a Java class "GRBAgent" (which is labeled "ViewerAgent" for illustration purposes in the schematic diagram of the system architecture in Figure 1), and client applications can use this class to query the name server, establish the connection with *geotool* or other server applications and fetch data from *geotool* in the CSS v3.0 schema.

Here is the code in WaveformViewer to get the "*.wfdisc*" table from *geotool*:

```
        ...
        Vector v[];      //define variable stores the return data from Geotool
        GRBAgent c = new GRBAgent();     //create class instance.
// connect to the name server machine, IP address 63.119.210.101:1077
// and connect to the server application "Geotool"
        c.connect("63.119.210.101", 1077, "Geotool");
//get table wfdisc from server application "Geotool"
        v = c.getTable("Wfdisc");
        ...
```

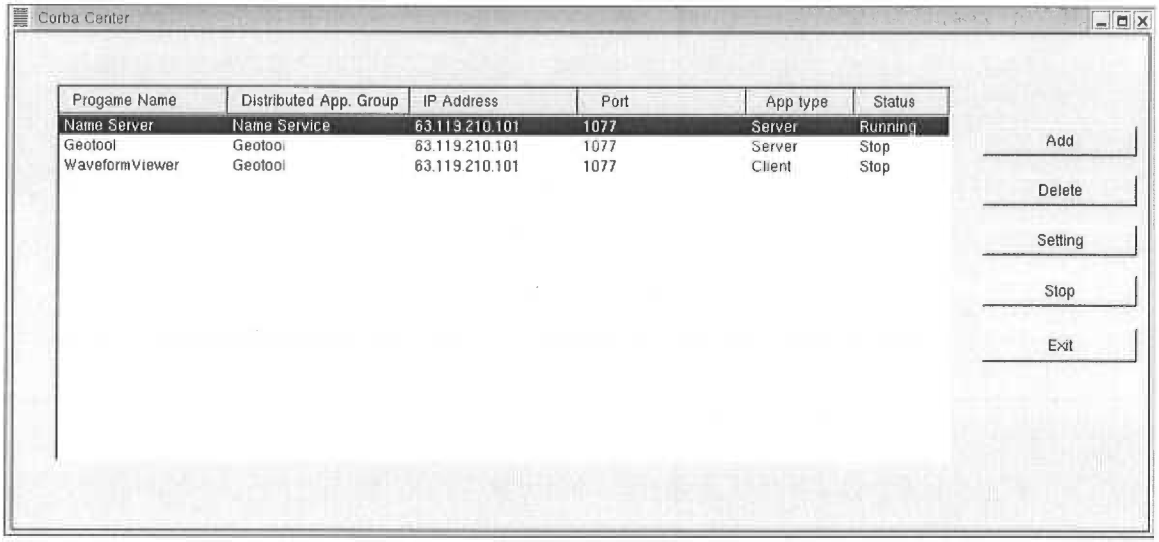
This is a description of the Java class GRBAgent:

> **public class GRBAgent** : class to implement the CORBA connection and other necessary functions of the CSS v3.0 data schema IDL.

Method Summary	
public	GRBAgent( ) Class instance construction, initialization.
public void	**Connect(String host, int portnum, String requested_name)** Connect to name server and establish connection with server application. Parameter: String host: the host name or IP address of the name server. int portnum: the listening port number of the name server. String requested_name: the server application name specified on the name server.
public Vector[]	getTable(String Suffix) Get table/data structure defined in Schema V.3 IDL from *geotool*. e.g. getTable("Wfdisc") - get data from wfdisc table,
Public sTimeSeries	**getTimeseries(int wfid)** In table "wfdisc", each wave has a unique *wfid* identifier. This function returns the TimeSeries of the *wfid* from *geotool* .

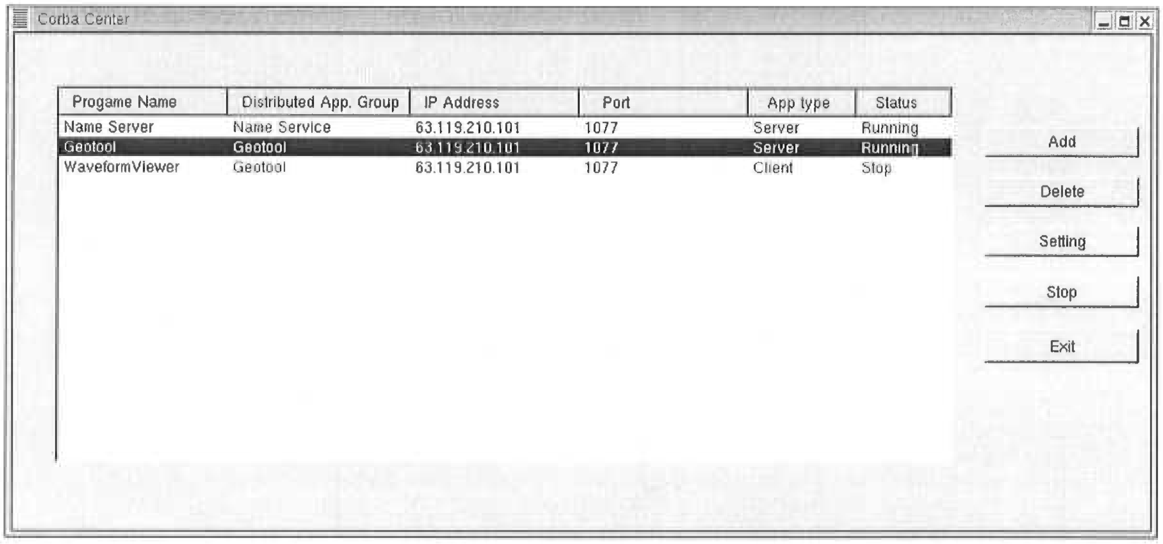
7. ON-SCREEN DEMONSTRATION OF CORBA ARCHITECTURE

The following figures show a user interface view of how this system would be used in practice. This demonstration of the system architecture is self-explanatory, since the figure captions serve as a step-by-step narrative of how CORBA can be used to enable data communications between *geotool* and a separate application (WaveformViewer).



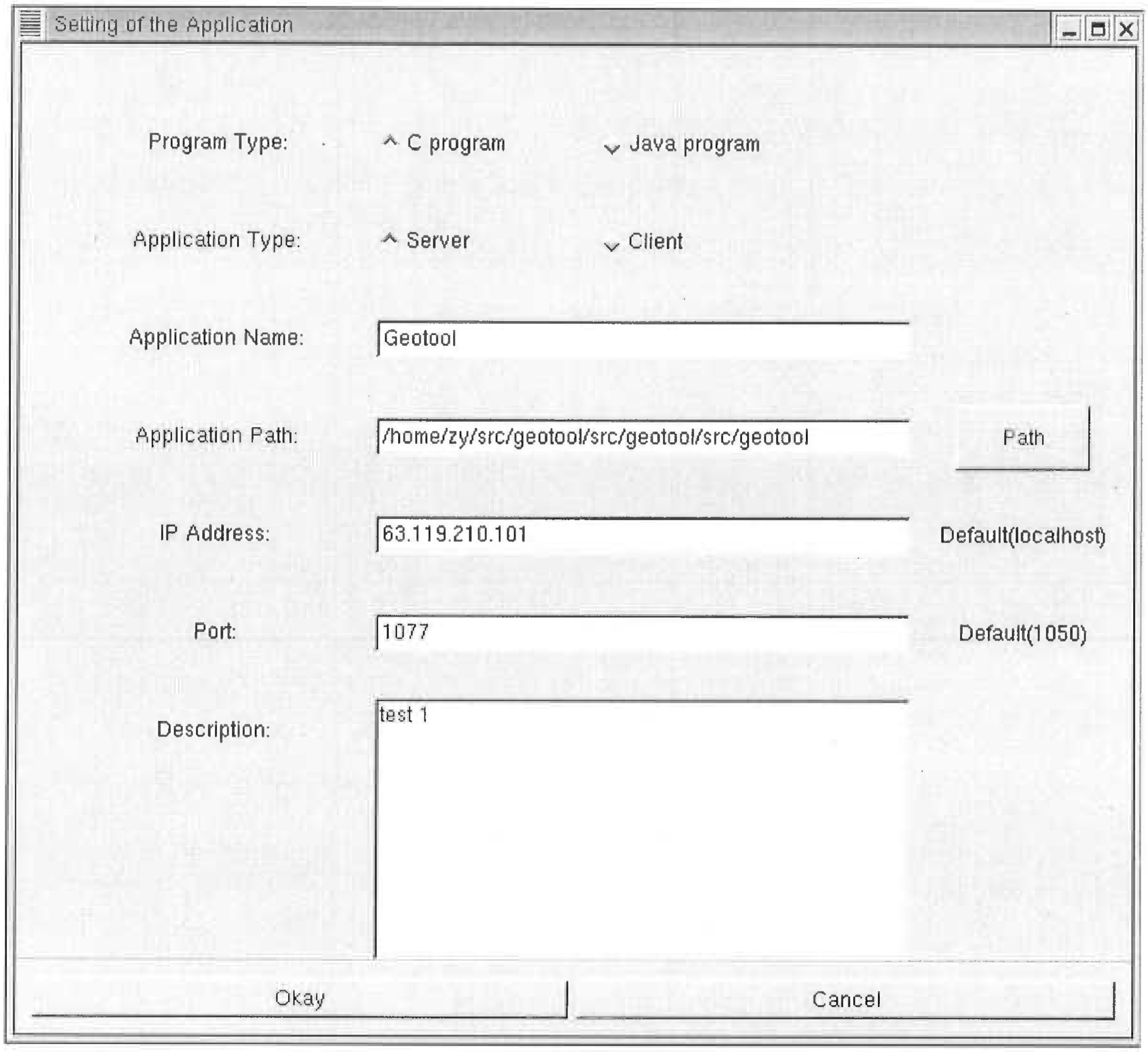
Progame Name	Distributed App. Group	IP Address	Port	App type	Status
Name Server	Name Service	63.119.210.101	1077	Server	Running
Geotool	Geotool	63.119.210.101	1077	Server	Stop
WaveformViewer	Geotool	63.119.210.101	1077	Client	Stop

Figure 3. Using the "CORBA Center" tool that was developed in the first phase of this project to make the use of CORBA simpler, start the CORBA name server to permit components to locate one another within the distributed processing environment.



Progame Name	Distributed App. Group	IP Address	Port	App type	Status
Name Server	Name Service	63.119.210.101	1077	Server	Running
Geotool	Geotool	63.119.210.101	1077	Server	Running
WaveformViewer	Geotool	63.119.210.101	1077	Client	Stop

Figure 4. Now use "CORBA Center" to start up the program *geotool* as a server within this distributed processing environment. Next use this same window to start up WaveformViewer, an application developed using Java classes, as a client within this distributed processing environment. Finally, select "Name Service" in "CORBA Center" to make settings to connect the server and the client.



Program Type: ^ C program ⌄ Java program

Application Type: ^ Server ⌄ Client

Application Name: Geotool

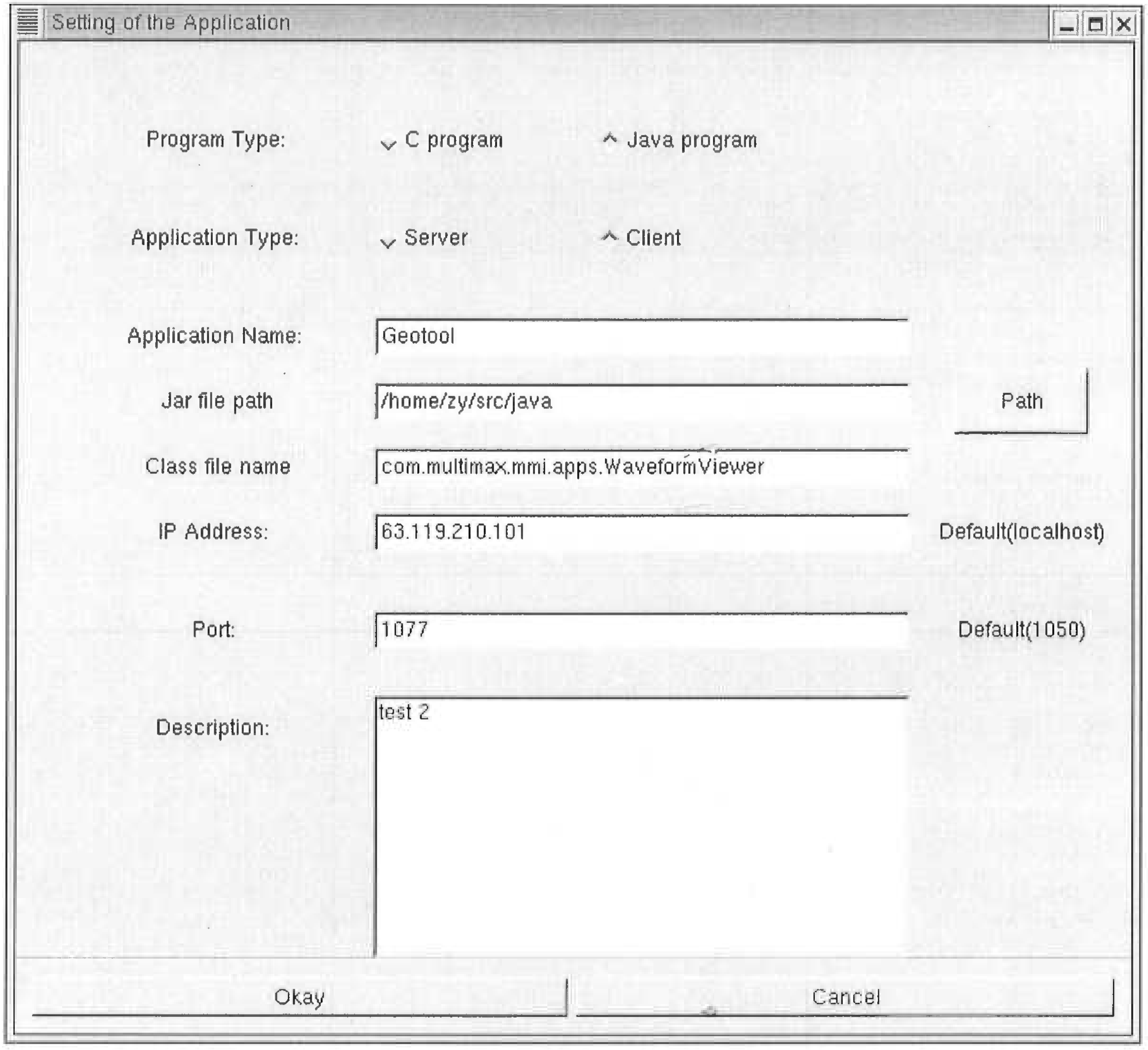
Application Path: /home/zy/src/geotool/src/geotool/src/geotool

IP Address: 63.119.210.101 Default(localhost)

Port: 1077 Default(1050)

Description: test 1

Figure 5. Now, a pop-up window is generated that enables the user to make settings in the CORBA connection to the server program. This form of the pop-up is the one for a C-language program like *geotool* (and like most of the legacy code that will be incorporated into the Reference Event analysis system using "wrappers" to make those standalone programs function as distributed components). This pop-up eliminates the need for detailed CORBA programming knowledge on the part of the user of the distributed processing system.



Program Type: C program Java program

Application Type: Server Client

Application Name: Geotool

Jar file path /home/zy/src/java

Class file name com.multimax.mmi.apps.WaveformViewer

IP Address: 63.119.210.101 Default(localhost)

Port: 1077 Default(1050)

Description: test 2

Figure 6. Next, a pop-up window is generated that enables the user to make settings in the CORBA connection to the client program. This form of the pop-up is the one for a Java application like WaveformViewer. In the future, new components could be written using C++ and Java, since these two object-oriented languages have CORBA interfaces, and they can then be incorporated into the reference event analysis system to create a true distributed processing environment of software objects communicating data and requests for processing services from one to another. WaveformViewer is a prototype of a Java component that is intended for use within that distributed processing environment.

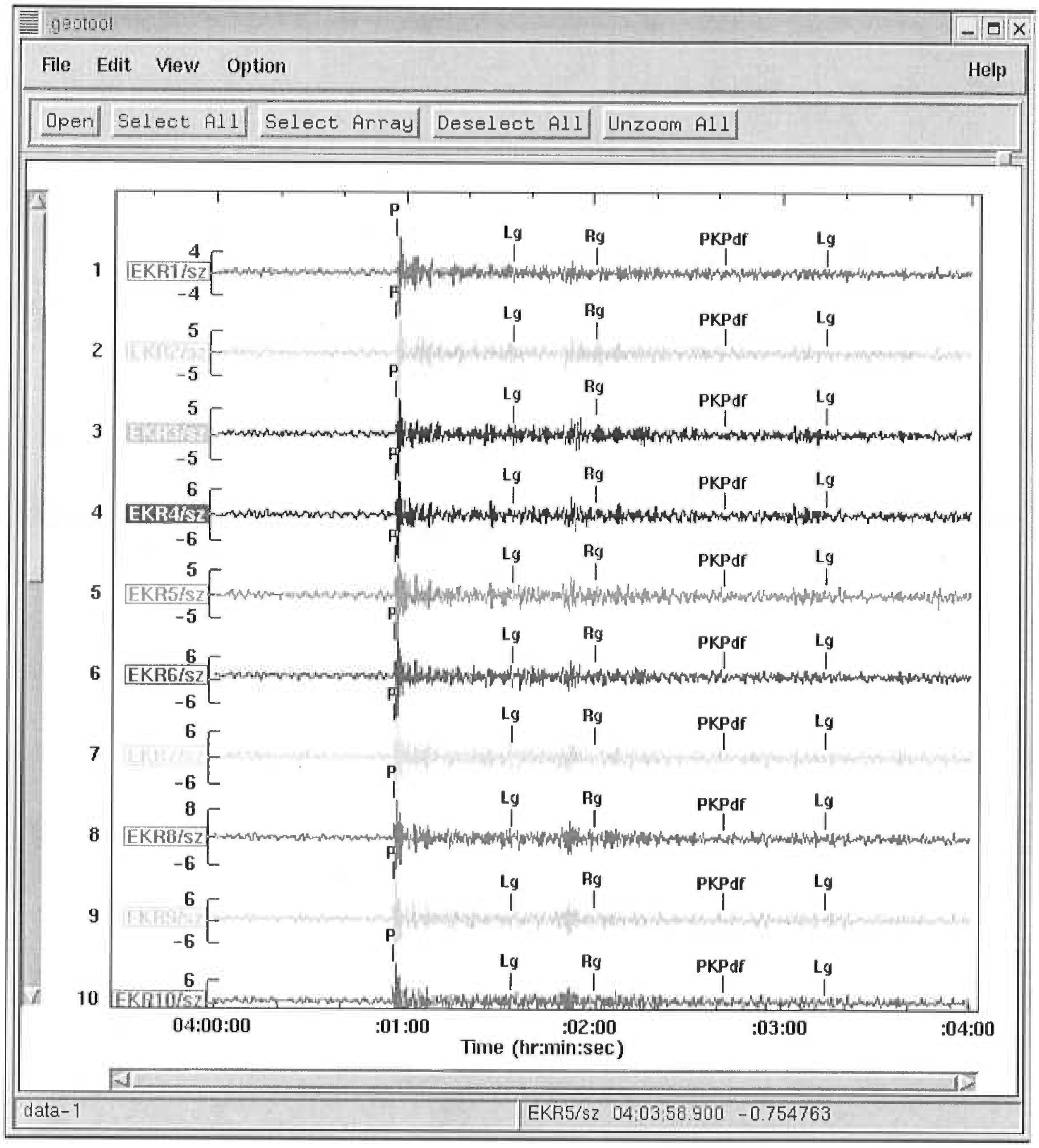
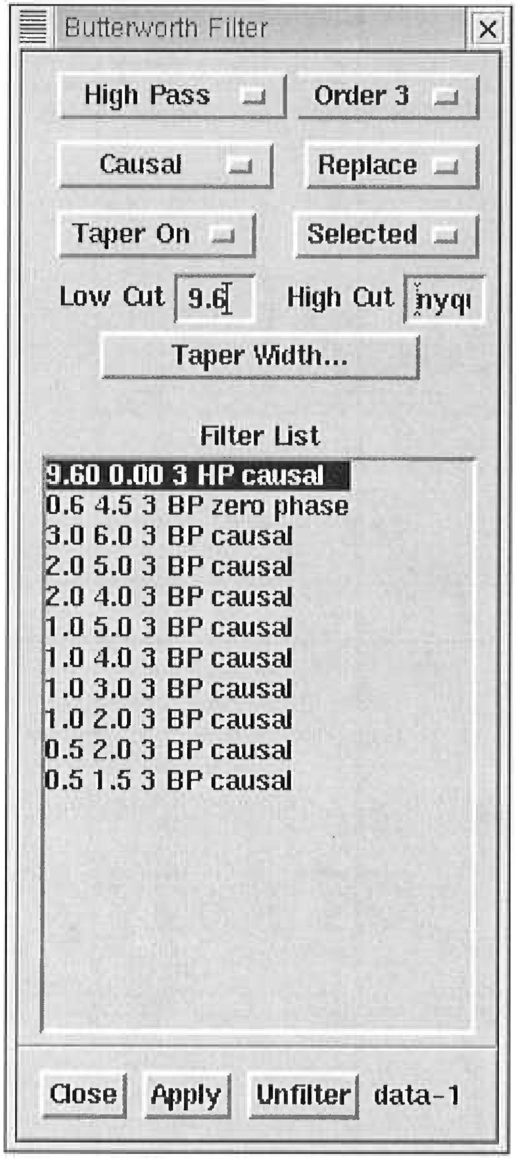


Figure 7. In the *geotool* server window, the user is displaying an event as recorded at the short-period vertical-component channels of the seismic array EKA in Scotland. The user has selected channels 3 and 4 so that bandpass filtering can be performed on them, and so geotool highlights them in blue. Since that filtering has not yet taken place, however, the waveforms as displayed on the screen are still the same as they are in the "*.w*" data files that reside on the hard disk.



Butterworth Filter

High Pass
Order 3
Causal
Replace
Taper On
Selected

Low Cut 9.6 High Cut nyq

Filter List

```
9.60 0.00 3 HP causal
0.6 4.5 3 BP zero phase
3.0 6.0 3 BP causal
2.0 5.0 3 BP causal
2.0 4.0 3 BP causal
1.0 5.0 3 BP causal
1.0 4.0 3 BP causal
1.0 3.0 3 BP causal
1.0 2.0 3 BP causal
0.5 2.0 3 BP causal
0.5 1.5 3 BP causal
```

data-1

Figure 8. The user creates a high-pass filter using the pop-up window that appears in response to selecting an item from a *geotool* menu. This is the type of waveform processing operation that is conventionally performed using programs such as *geotool* or SAC. Although here it is being performed within *geotool*, it is the goal of the reference event analysis system that scientists can implement their own processing tools, for instance a high-pass filter that uses a different algorithm from *geotool*'s, by means of distributed processing. This figure shows the conventional approach, namely using a callback function that is part of *geotool* itself. The following figures show the distributed-processing approach, whereby data are transmitted from *geotool* via CORBA to a separate Java application, namely WaveformViewer, that will apply its own filtering routine and then send the filtered data back to *geotool*. However, we first apply the conventional approach of conducting filtering within *geotool* so that we can show that data modified within *geotool* are sent dynamically to the distributed components directly from memory rather than from disk.

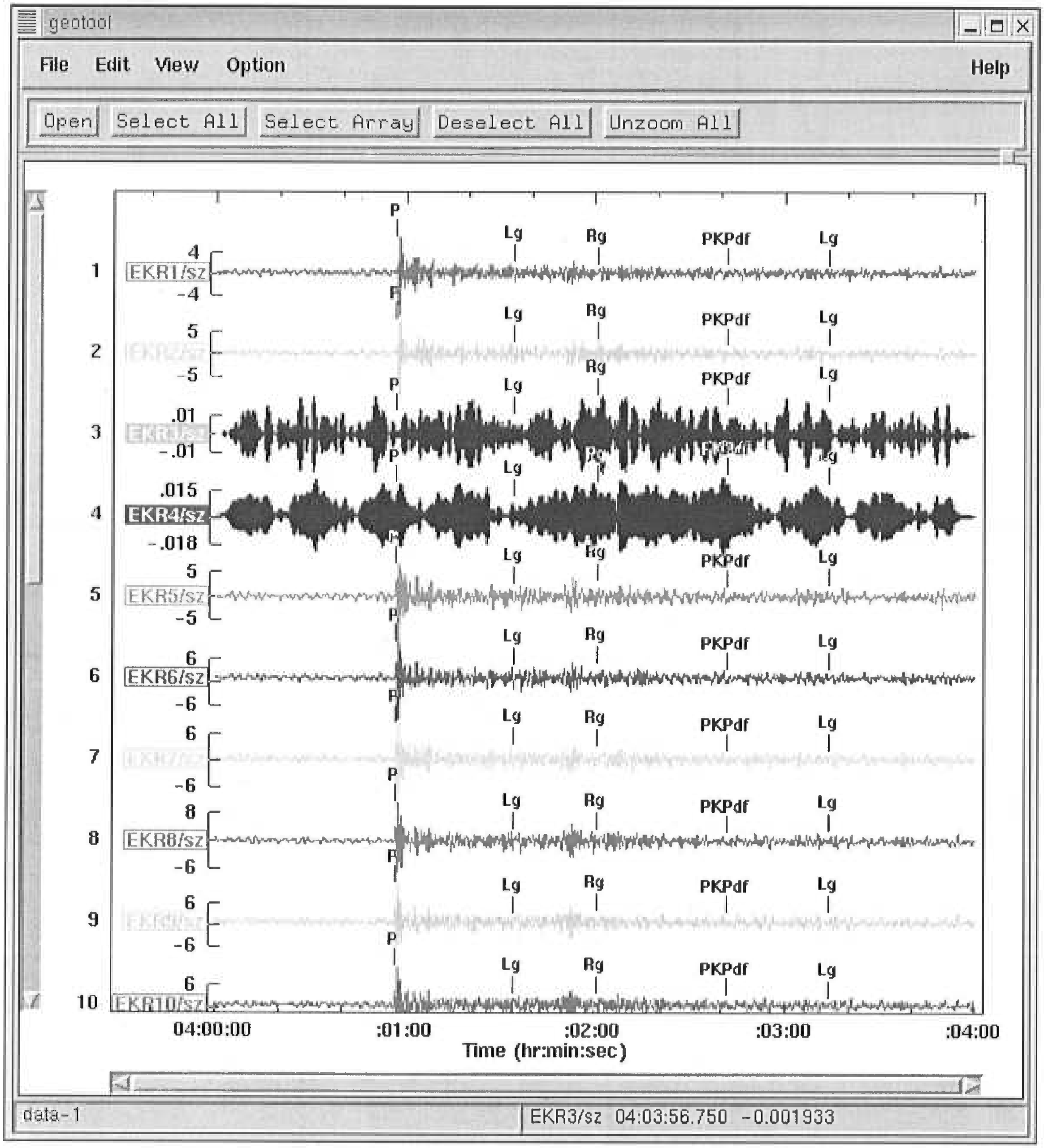
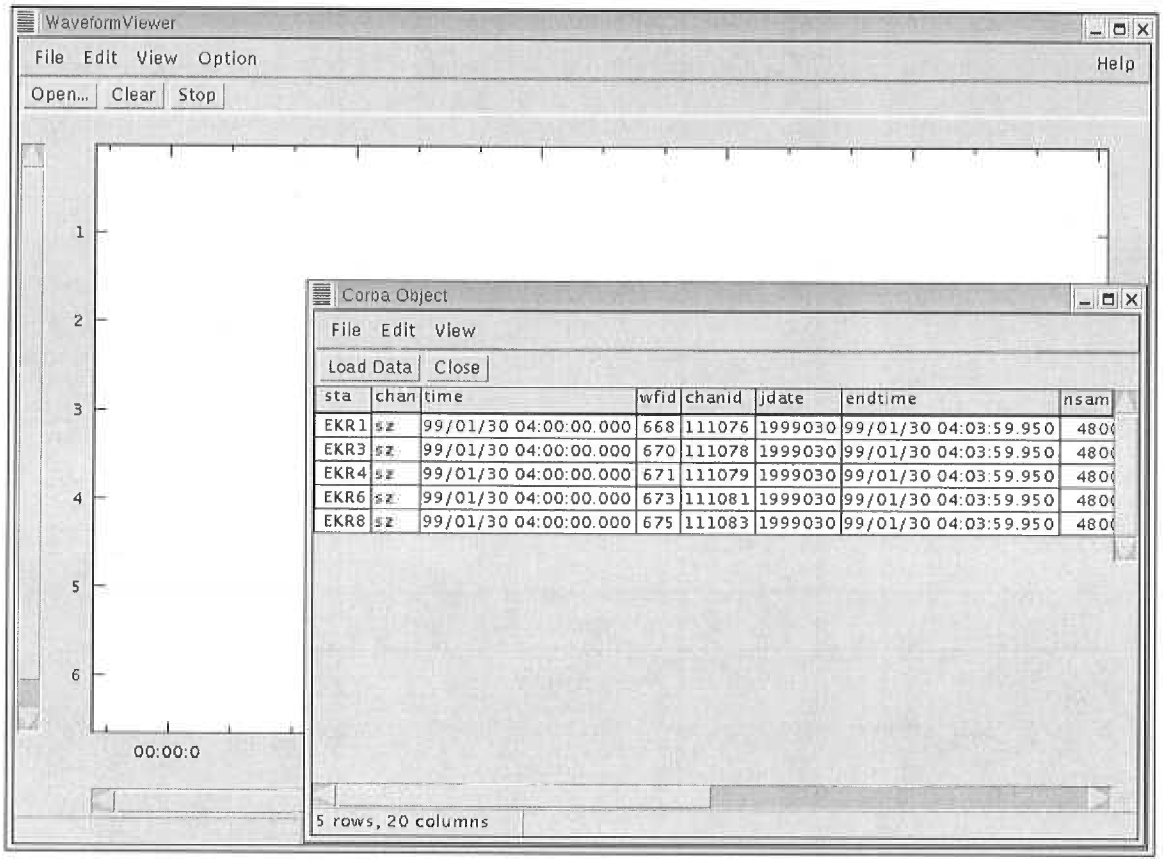


Figure 9. The selected filter has been applied to channels 3 and 4. Now the waveforms as displayed on the screen are different from the disk-resident data. In a conventional data-processing pipeline consisting of stand-alone applications that cannot communicate with one another, it would be necessary to save onto disk the modified waveforms (and probably the modified ".wfdisc" database table that describes them), and then the next program in the data stream would read those new files from disk as its input. The work we have done in the first phase of this project permits the memory-resident waveforms to be transmitted to the next program (in this particular case, WaveformViewer) dynamically.



sta	chan	time	wfid	chanid	jdate	endtime	nsam
EKR1	sz	99/01/30 04:00:00.000	668	111076	1999030	99/01/30 04:03:59.950	480
EKR3	sz	99/01/30 04:00:00.000	670	111078	1999030	99/01/30 04:03:59.950	480
EKR4	sz	99/01/30 04:00:00.000	671	111079	1999030	99/01/30 04:03:59.950	480
EKR6	sz	99/01/30 04:00:00.000	673	111081	1999030	99/01/30 04:03:59.950	480
EKR8	sz	99/01/30 04:00:00.000	675	111083	1999030	99/01/30 04:03:59.950	480

5 rows, 20 columns

Figure 10. In WaveformViewer, a Java application, the user sees a ".wfdisc" display of the data channels that were selected within geotool and were then transmitted via CORBA to WaveformViewer. The ".wfdisc" information was translated to CORBA IDL on the *geotool* side of the connection, and those IDL data were then interpreted properly on the WaveformViewer side of the connection so that they can be displayed as the seismogram description table that appears in the pop-up window. The user will then select certain of those channels for display within WaveformViewer. In this particular case we shall choose to display all 5 of the channels that were transmitted from *geotool.* Note that these include channels 3 and 4, to which a high-pass filter had been applied by *geotool*, as well as 3 waveforms that were unmodified by *geotool* from their disk-resident files.

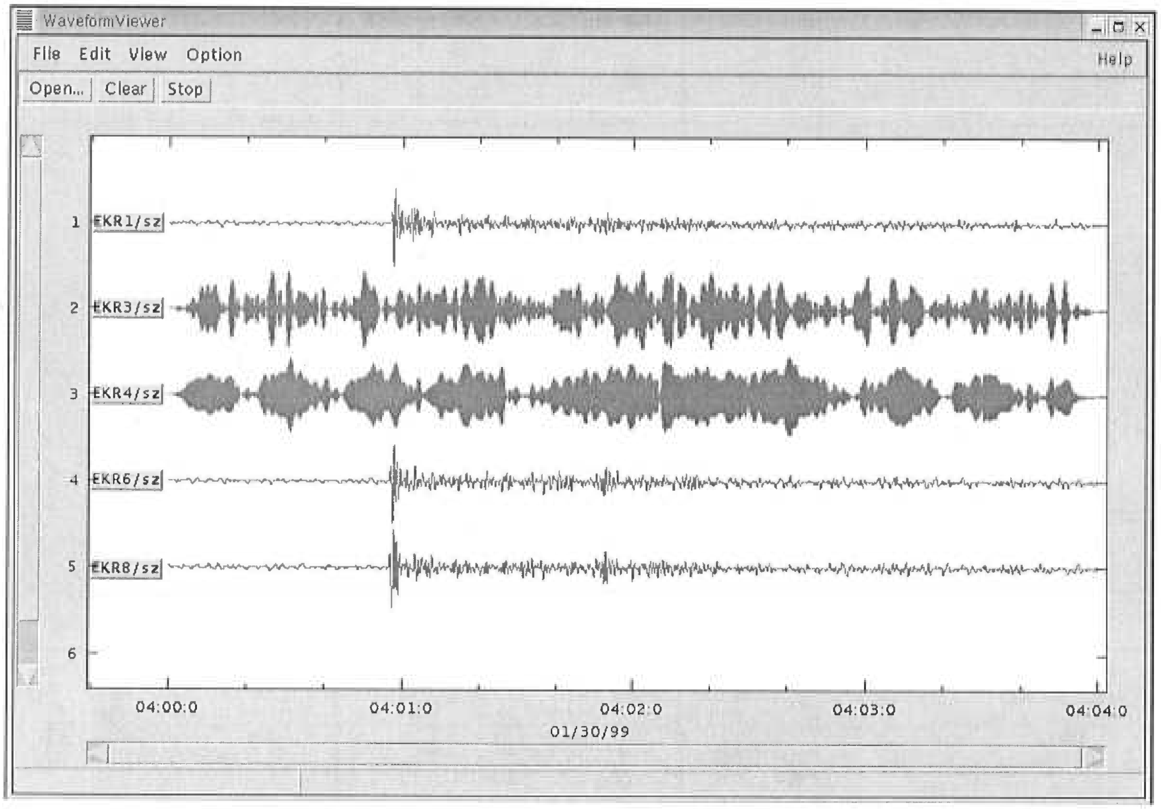
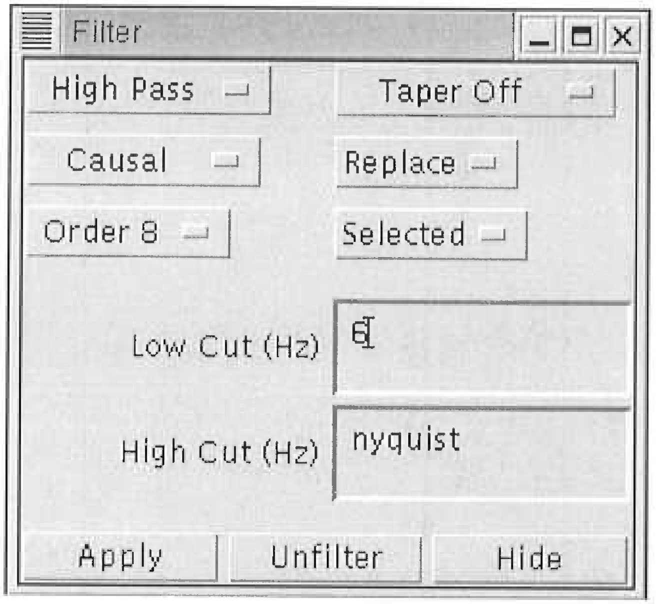


Figure 11. Since the user selected all five channels that were listed in the pop-up window, WaveformViewer then displays them all. Note that the waveforms for channels EKR3 and EKR4 are the filtered ones that were resident in memory within *geotool* and not the original ones that reside on disk. The data have been transmitted from one program to another dynamically rather than via intermediate disk files. Note also the similarity of the WaveformDisplay screen (which employs Java graphics) to the *geotool* screen (which employs X Windows graphics). This demonstrates the feasibility of using Java classes to duplicate, in a platform-independent application, the functionality of the Motif widgets that were written as the basis of *geotool*. Although WaveformViewer, as its name suggests, displays here only the waveforms and not the phase arrivals that were shown in the geotool display, the arrival information too (as well as ".origin", ".assoc", ".sitechan", etc., information) was transmitted from geotool to WaveformViewer via CORBA through the IDL code that we have developed in the first phase of this project to hold the contents of the CSS v3.0 schema for seismic data tables. These data can then be used by other Java classes for additional data processing beyond the waveform filtering that will be shown here. From this display, the user now selects the last 2 channels on the screen (EKR6 and EKR8) to be filtered using one of the Java classes. As in the previously shown *geotool* screen, the selected channels are then highlighted in blue. On account of the similarity between the *geotool* and WaveformViewer graphics displays, we shall not show herein that process of channel selection.



High Pass

Taper Off

Causal

Replace

Order 8

Selected

Low Cut (Hz) 6

High Cut (Hz) nyquist

Figure 12. Just as with *geotool*, the user is presented with a pop-up window that allows a filter to be created. The difference is that this window and the filter it applies are written in Java, rather than C and Motif. The filter is then applied to the selected channels, namely EKR6 and EKR8

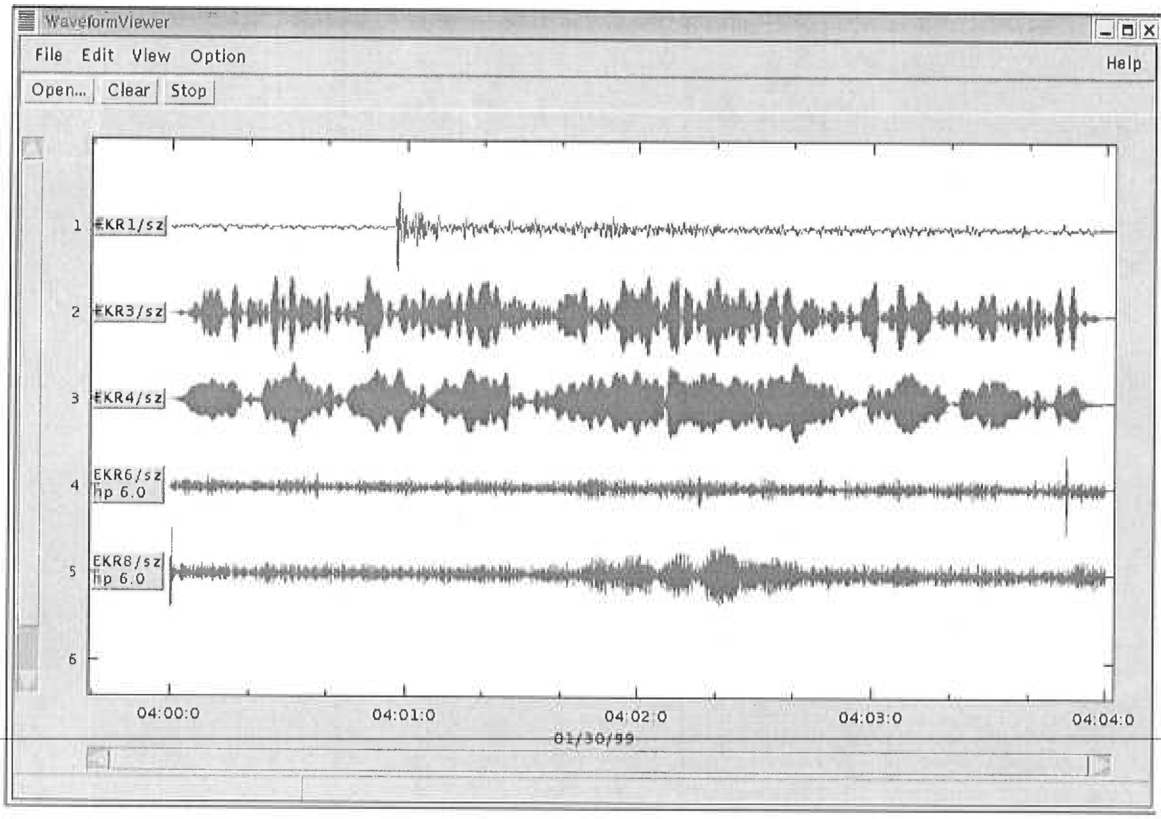
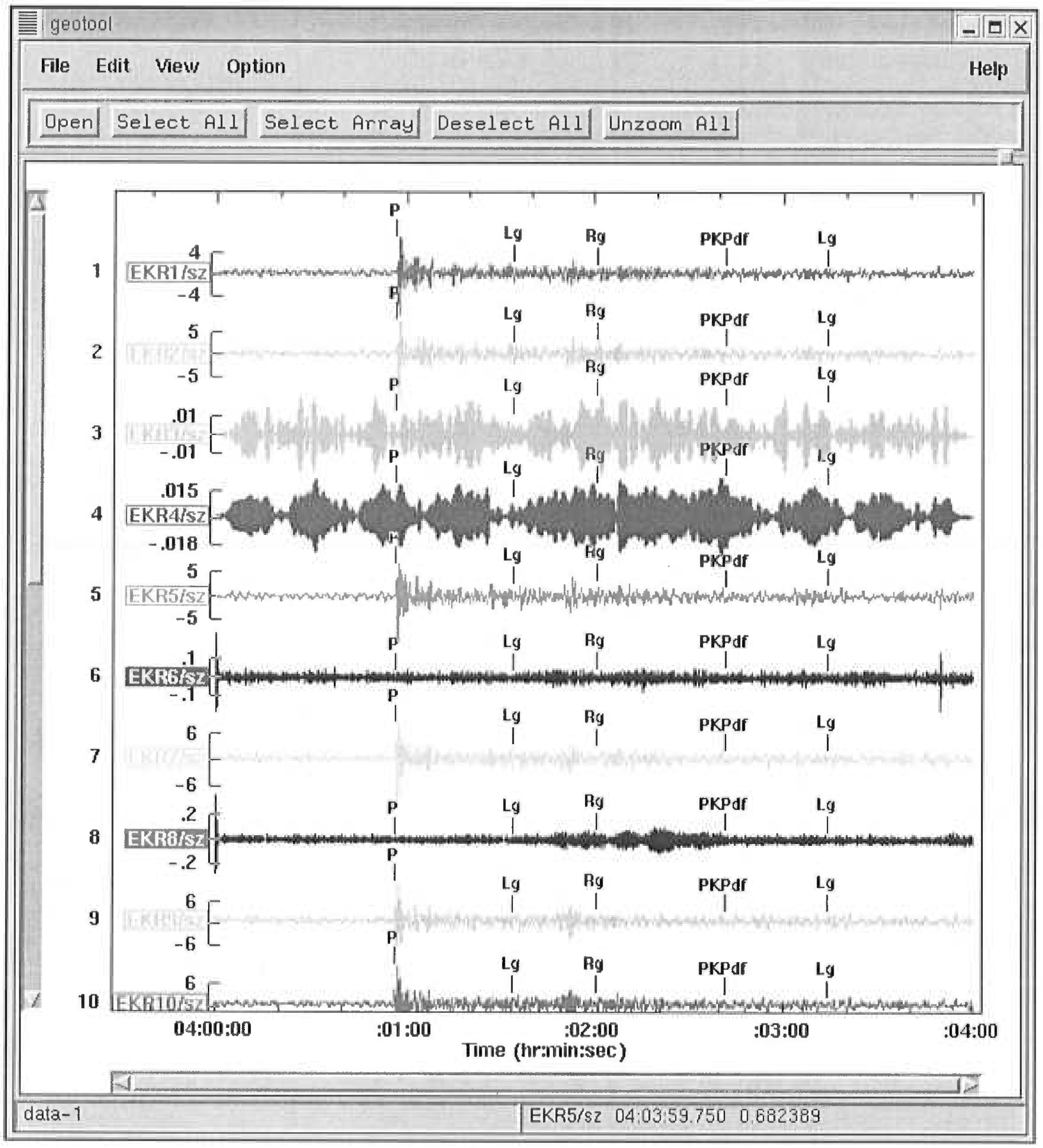


Figure 13. The user next selects these newly filtered channels with the mouse, and then they are both sent back to *geotool* via the CORBA bus.



data-1

EKR5/sz 04:03:59.750 0.682389

Figure 14. The waveform display that has been running within the *geotool* window is updated with the data for channels EKR6 and EKR8 that were filtered within the WaveformViewer Java application. Just as with the data transmission from *geotool* to WaveformViewer, the data that were transmitted from WaveformViewer to *geotool* were sent dynamically rather than being written out as disk files. In addition to the binary waveforms themselves, the data tables such as ".wfdisc" and ".arrival" were transmitted in both directions via CORBA through the code that we have written to translate the CSS v3.0 data table schema to IDL. The whole process shows how a filter would be applied in the reference event analysis system: the user can apply a filter using a routine that is part of *geotool* or can apply a filter (in this case, one written in Java) that is part of a separate application (in this case, WaveformViewer). We have thus demonstrated the technical feasibility of using CORBA to link stand-alone seismic data analysis programs.

8. DISCUSSION OF DISTRIBUTED PROCESSING

The proof of concept shown in Figures 3 – 14 demonstrates that a distributed software architecture built using CORBA can form the basis of a reference event analysis system. Instead of having as the tool to use for identifying and characterizing reference events a single huge program that is difficult and/or dangerous to modify, it is possible to modify *geotool* (or ARS, or SAC, etc.) to be used as a data server and then to perform the actual seismic analysis externally to *geotool* by using a host of separate applications, including both existing code and programs yet to be written, running on whatever platform is most appropriate. In particular, the analyst should be able to use seismic data centers and supercomputers located at remote facilities accessible via the Internet, just as if those capabilities were offered by the server platform itself. The data communications should be performed as easily as possible without requiring the analyst to go through intermediate steps of creating, exporting, and importing temporary files and database records. This flexibility will be required for the intensive study of reference events, where new algorithms not ordinarily used for routine seismic analysis may be required in order to perform specialized investigations in individual cases. The analyst should be able to select which programs to run, and in what order to run them, and in many cases it may well be necessary to construct new programs. The analysis programs that run as client applications should perform only one or two specific tasks, and they can be written to perform only those particular functions without reproducing all the overhead capabilities for data management that are required by the server program.

The example that was illustrated in these figures should serve to demonstrate the need for certain features in a distributed processing architecture that are not conventionally part of a software system like *geotool* that runs as a single program on one computer (albeit within multiple windows). In Figure 9 data channels 6 and 8 are unfiltered, in Figure 13 they have been filtered by the Java application, and in Figure 14 they have been returned to the *geotool* server in their filtered form. What, then, should the *geotool* server program do with the unfiltered versions of these channels, which of course are still resident in the program's RAM, and are still displayed on the Linux computer screen, once the client program is ready to send the filtered versions back to the server? Should the process of returning the filtered data from the client automatically cause the server to delete the unfiltered data? Should the *geotool* window instead now display the presence of two different versions of these waveforms, and if so, should it change the waveform identification label *.wfid* to reflect that there are now two different waveforms for the same time windows on these particular data channels? Should the return of the filtered data from the server be blocked until the analyst has hit a "receive" button in *geotool* indicating that the original data should now be overwritten? (Blocking the transmission of a data message from the client until the server specifies that it is ready to receive it makes the data stream function in many ways like an Instant Messenger service such as AOL IM or ICQ.) What happens if the data analyst sends the unfiltered data to one client application for filtering and then chooses to send those same data to a different client program for some other processing, such as polarity reversal, *before* the first client program has returned its results? Clearly, the distributed processing architecture presents a number of issues related to the data's referential integrity, versioning, and configuration management. Establishing and enforcing policies for handling these issues will be an important part of the design of a reference event analysis system.

It is also clear that the system must use some sort of metadata audit trail for tracking the distributed processing. Regardless of whether the unfiltered data are deleted from the server's memory and screen display when the filtered data are returned from the client, how will the system distinguish between the filtered waveforms and the original ones that reside on the disk? In the particular case of the interactive WaveformViewer client application, the analyst was required to input the filter parameters by using a GUI, so it is known (at least by the analyst, if not by *geotool*) what sort of filter was applied. In general, however, it is far more likely that the client code will run automatically, and the analyst will have no knowledge of the exact parameters that govern the algorithm implemented by the client program, which after all is likely to be running on a different computer, perhaps even at a different facility. The client must therefore return not only the processed data but also a metadata message describing just what operations it performed and the parameters it used. The server must construct a database archive of these metadata records as a tool to use for the data versioning and configuration management so that any one or all of the data processing routines executed by the distributed client applications can later be repeated or undone. We shall present a formulation of metadata in a subsequent section, after we discuss the use of XML as a means for expressing data and for invoking remote processing.

9. SHORTCOMINGS IN THE USE OF CORBA

In the last few years there has been in the software industry a move away from 1990s client/server architecture based on CORBA. This move has been motivated by the need for business-to-business (B2B) software interoperability. The software of one company needs to interface with that of another, but security requirements rule out the use of a tight coupling model such as CORBA, since binary protocols like CORBA cannot be used to transmit data among machines that use firewalls that regard external data transmissions and requests as viruses or hacker intrusions. A company's proprietary data must not be accessible by another company using a binary Remote Procedure Call, since the content of that message cannot be analyzed and vetted by the system that receives it. Another major problem with the use of CORBA in a commercial environment is that CORBA is limited to the UNIX (or Linux) and Java platforms; Microsoft Windows has used its own data communications protocol, DCOM, which is based on a different object model. Building software bridges between CORBA and DCOM systems is quite difficult, but in most commercial applications, system architectures make extensive use of both UNIX and Windows systems, and code running on both of them must be able to exchange data between them. It should of course be noted that in Figures 3 - 14 we did in fact exchange data between Linux and Windows 2000 using CORBA, but this was possible only because the code that ran on Windows was in fact running within the Java platform rather than as native Windows code. We feel that this scenario will become increasingly unrealistic in coming years, as more and more scientific code is written that takes advantage of the ever-increasing power and low costs of desktop and laptop computers by writing C/C++ programs that run as Windows native code. Because this project is a Small Business Innovation Research (SBIR) project, it has as its ultimate goal the development of software that can be marketed commercially, and we therefore need to be able to use native Windows code. This cannot be done using CORBA. In the second phase of this project, we have therefore investigated using the new B2B standard for data communications, and that is the technology of Web services.

10. WEB SERVICES

The term "Web services" is often misunderstood, since it has become associated (through marketing initiatives) with the concept of selling what were formerly desktop applications as "software as a service" through a subscription model. This is in fact one use for Web services, but it is far better to think of this technology as a Web *of* services, in the same manner that the WWW is a Web of information. The shelves of the computer sections of most bookshops nowadays are filled with textbooks describing how to build Web services, since it is a technology that is rapidly becoming a necessity for businesses that wish to stay competitive in a B2B marketplace. (The number of CORBA programming books, by comparison, is miniscule.) Web services are built around the HTTP protocol, so they run on every platform. They use XML messages, so they are text-based and thus are platform independent. Web service software written on Windows can communicate perfectly with Web services written in Java, and this is not at all true of CORBA. Every major software vendor has released toolkits to enable the development of Web services, including software running under the UNIX, Windows, and Java platforms (e.g., Sun Microsystems' "Sun ONE", Microsoft's ASP.NET, and the Java Web Services Developer Pack). It is especially important for B2B purposes that Web services operate by transmitting ASCII messages (written using XML) over HTTP, since this text-based messaging over a standard industry protocol can work through firewalls, whereas a binary technology such as CORBA cannot.

Web services are built upon the Simple Object Access Protocol (SOAP) for network communication between software components. This protocol is simpler and thus easier to use than is CORBA. SOAP containers are available in Java, C++, Perl, Python, C#, and many other languages. SOAP provides a mechanism for performing Remote Procedure Calls more easily than does CORBA. No ORB is required to invoke methods from distributed objects, since the Web server itself performs that operation. In particular, the open-source Web server from the Apache project makes Web services possible. Just as the ORB is replaced with object method calls made via SOAP, the CORBA IDL is replaced by Web Services Description Language (WDSL), a standard that is based on XML. Finally, the CORBA Naming Service is replaced by the Universal Description, Discovery, and Integration (UDDI) software standard that allows a Web service to be published on the Internet or Intranet and then found by clients that need it.

Moving from sockets to CORBA to SOAP results in a gain of flexibility by moving to increasingly higher levels of software abstraction within the TCP/IP stack, but it comes at the cost of decreased efficiency, as direct binary communications are replaced by ASCII messages. We nevertheless believe this loss in efficiency is offset by the decrease in development time, and especially by the decrease in software maintenance time, required for the implementation of such a loosely coupled system of independently developed components rather than a large single program or a tightly coupled client/server system.

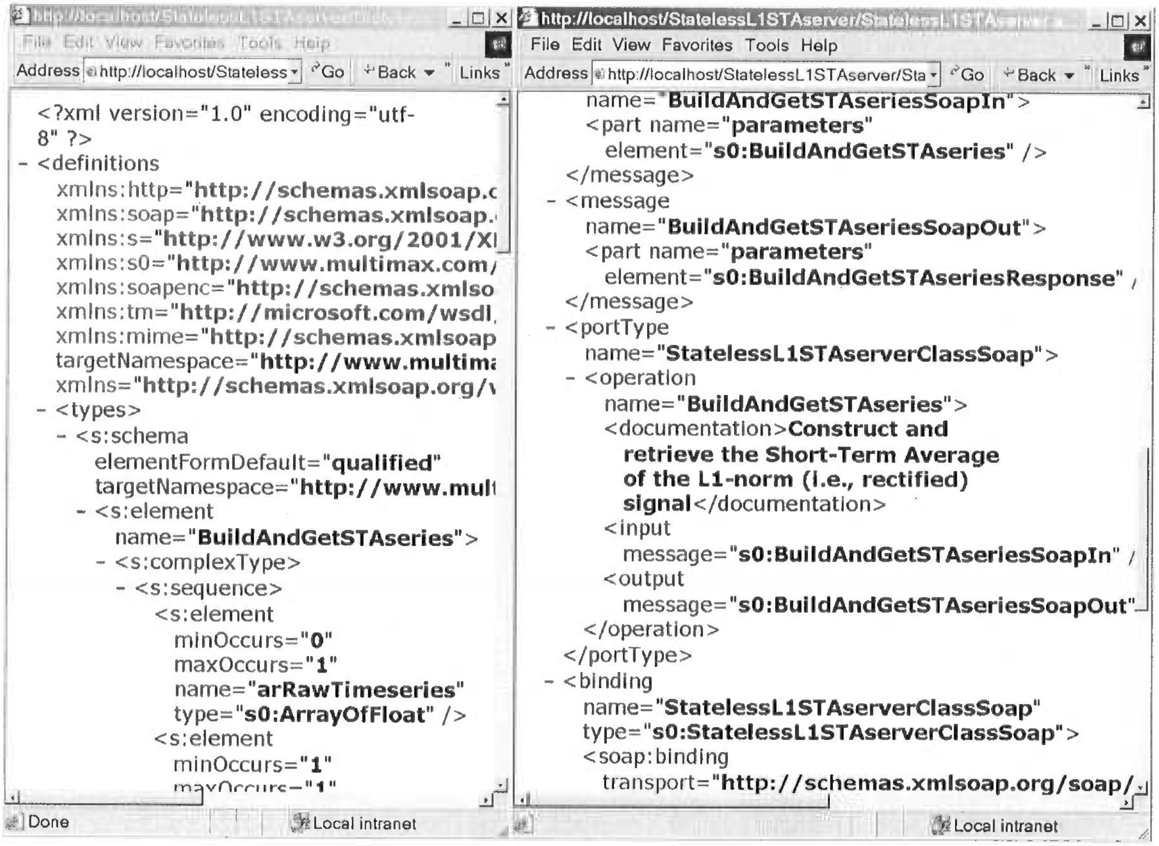
As we have noted above, a particular advantage of Web services is that whereas CORBA permits communication only among the UNIX, Linux, and Java platforms, SOAP Web services permit these platforms to communicate directly with the Windows platform. We feel the commercialization of seismic data analysis software will require the ability for the software to operate within such a heterogeneous environment.

11. USE OF XML/SOAP FOR DISTRIBUTED SEISMIC DATA PROCESSING

Our investigations of the use of XML SOAP for distributed seismic processing will be illustrated by a Web service that computes the short-term average (STA) of a time series. This routine is a C++ program running within the Microsoft .NET software platform under Windows 2000 (not as a program running on the Java platform within Windows). Figure 15 shows the Web Services Description Language (WSDL) interface that this program presents to the Internet. WSDL is an XML markup language that is used for SOAP Web services in a manner analogous to the use of the IDL software that enables data communications under CORBA. In this case the WSDL file was generated automatically by the Microsoft Visual Studio software development tool when the C++ program was installed within the *Inetpub/wwwroot* directory of the Web server, so the programmer does not need to write the WSDL description. Figure 16 shows the format of the SOAP message that should be used by a client program such as *geotool* that has need of this particular Web service. This SOAP format too is generated automatically by the Microsoft .NET development environment, and SOAP messages in that format can then be used by a program running under Linux (like *geotool*) to communicate with this Windows program.

In our description of the work performed in the first phase of this project when we used CORBA as the middleware platform for distributed processing, we presented a number of screenshots showing the transmission of waveforms between the C-language program *geotool* running under Linux and the Java program WaveformViewer running within Windows 2000. We shall not show a similar sequence of screenshots herein, since externally the distributed processing appears the same whether CORBA or Web services are used as the architecture. We do show in Figure 17 how to establish a routing to that same Java program, now accepting SOAP messages rather than CORBA transmissions to invoke its operations, and at the bottom of the figure we show that a *geotool* menu item is being used to receive the results of the processing performed by that remote application. Note that we have replaced the CORBA Center code that was developed in the first part of this project (*cf.* Figures 3 – 6) with a new "SOAP Center" application that establishes the data routing in an analogous manner.

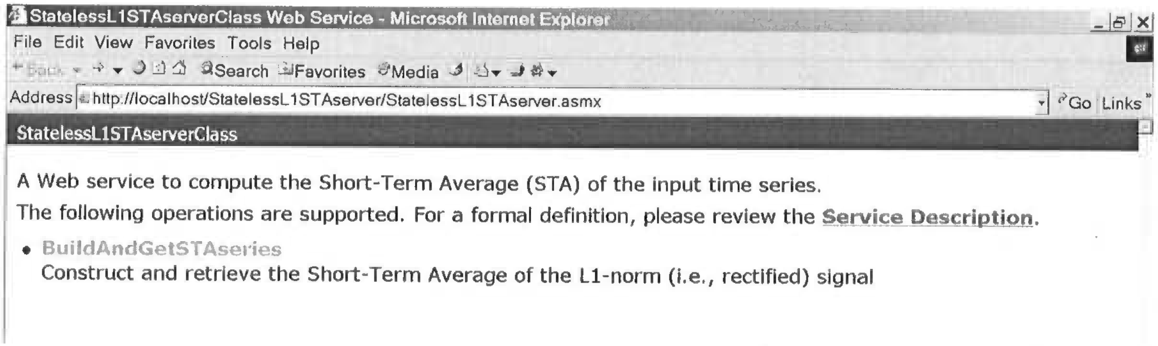
We mentioned earlier that a problem with the architecture shown in Figure 1 is that *geotool*, even though it operates as a data server, should more properly be considered a client, since it also fills the role of being the graphical user interface. A user of *geotool* will then use CORBA, or SOAP, to invoke a remote application that performs a service (such as the bandpass filtering that was shown earlier) on behalf of *geotool*. The Web services architecture makes this distinction explicit, so we shall henceforth refer to *geotool* as a client of services that are provided by other programs such as the Java WaveformViewer code, which is thus more properly considered an applications server.


File Edit View Favorites Tools Help
Address http://localhost/Stateless
Go
Back
Links
Done
Local intranet
http://localhost/StatelessL1STAserver/StatelessL1STAserver
File Edit View Favorites Tools Help
Address http://localhost/StatelessL1STAserver/Sta
Go
Back
Links
Local intranet

```
  <?xml version="1.0" encoding="utf-
  8" ?>
- <definitions
    xmlns:http="http://schemas.xmlsoap.c
    xmlns:soap="http://schemas.xmlsoap.
    xmlns:s="http://www.w3.org/2001/XI
    xmlns:s0="http://www.multimax.com/
    xmlns:soapenc="http://schemas.xmlso
    xmlns:tm="http://microsoft.com/wsdl,
    xmlns:mime="http://schemas.xmlsoap
    targetNamespace="http://www.multim
    xmlns="http://schemas.xmlsoap.org/\
  - <types>
    - <s:schema
        elementFormDefault="qualified"
        targetNamespace="http://www.mult
      - <s:element
          name="BuildAndGetSTAseries">
        - <s:complexType>
          - <s:sequence>
              <s:element
                minOccurs="0"
                maxOccurs="1"
                name="arRawTimeseries"
                type="s0:ArrayOfFloat" />
              <s:element
                minOccurs="1"
                maxOccurs="1"
```

```
      name="BuildAndGetSTAseriesSoapIn">
      <part name="parameters"
        element="s0:BuildAndGetSTAseries" />
    </message>
  - <message
      name="BuildAndGetSTAseriesSoapOut">
      <part name="parameters"
        element="s0:BuildAndGetSTAseriesResponse" /
    </message>
  - <portType
      name="StatelessL1STAserverClassSoap">
    - <operation
        name="BuildAndGetSTAseries">
        <documentation>Construct and
          retrieve the Short-Term Average
          of the L1-norm (i.e., rectified)
          signal</documentation>
        <input
          message="s0:BuildAndGetSTAseriesSoapIn" /
        <output
          message="s0:BuildAndGetSTAseriesSoapOut"
      </operation>
    </portType>
  - <binding
      name="StatelessL1STAserverClassSoap"
      type="s0:StatelessL1STAserverClassSoap">
      <soap:binding
        transport="http://schemas.xmlsoap.org/soap/
```

Figure 15. Screenshot of two browser windows displaying portions of the WSDL (Web Services Description Language) interface to the Web service *StatelessL1STAserver*, which computes the short-term average (STA) of a time series, using the L1 (*i.e.*, absolute value) norm to rectify the data. *StatelessL1STAserver* is a C++ class operating as a Web service within the Microsoft .NET software platform under the Windows 2000 operating system. The WSDL interface shows the input values (namely the array of data points that make up the time series, along with a couple of integers describing the desired STA time window) that can be used by *geotool* (or any other program needing the service of this STA algorithm) to access this Web service, as well as the output values (namely, a new array of data points) that this service will return to the calling program. The application that needs this service will call a local proxy object that has the same interface as is shown in this WSDL file, and the proxy will transmit the request via the Web server (within the local host, or across the Intranet or Internet, as directed by the URL for the Web service) and will receive the response. Because WSDL and SOAP are XML vocabularies, it does not matter that *geotool* and *StatelessL1STAserver* are written in different programming languages and run on different operating systems.

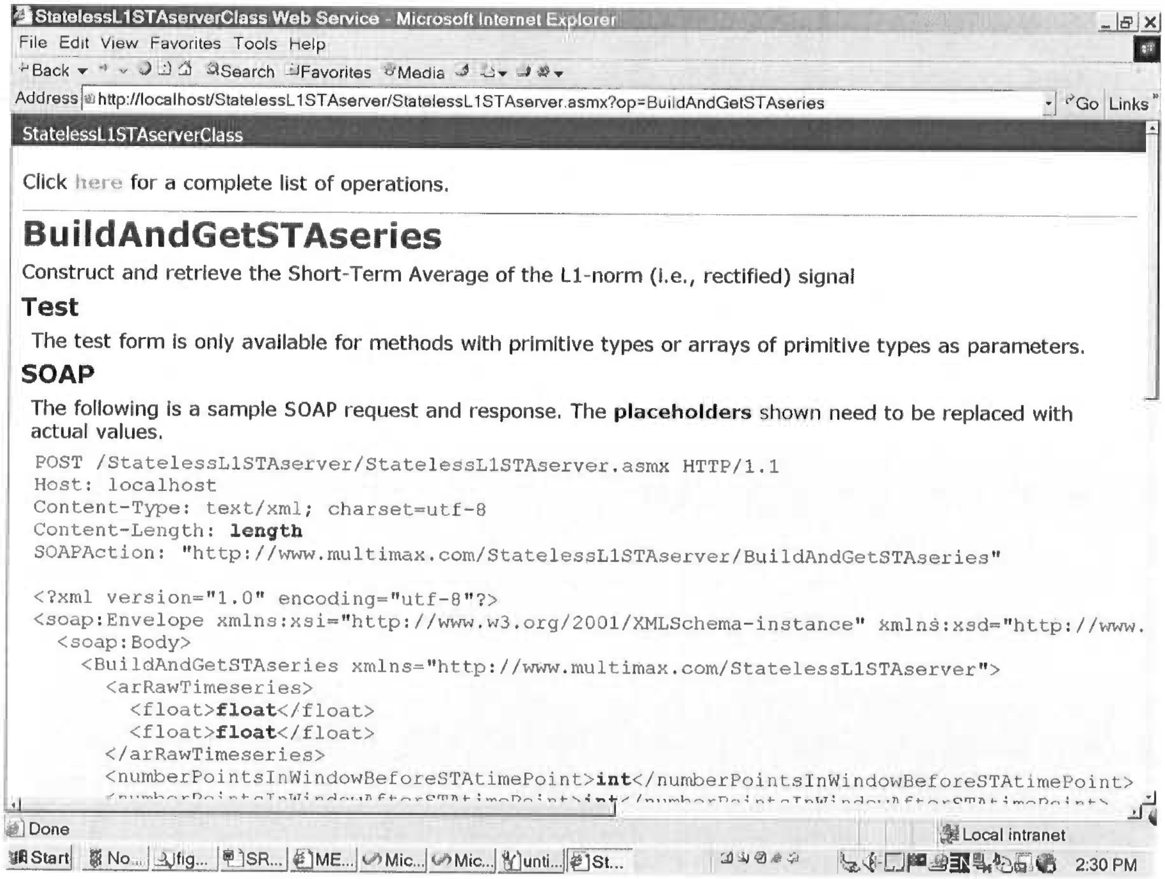

StatelessL1STAserverClass Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/StatelessL1STAserver/StatelessL1STAserver.asmx
Go Links

StatelessL1STAserverClass

A Web service to compute the Short-Term Average (STA) of the input time series.

The following operations are supported. For a formal definition, please review the Service Description.

- BuildAndGetSTAseries
  Construct and retrieve the Short-Term Average of the L1-norm (i.e., rectified) signal


StatelessL1STAserverClass Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/StatelessL1STAserver/StatelessL1STAserver.asmx?op=BuildAndGetSTAseries
Go Links
Done
Local intranet
Start No... fig... SR... ME... Mic... Mic... unti... St...
2:30 PM

StatelessL1STAserverClass

Click here for a complete list of operations.

BuildAndGetSTAseries

Construct and retrieve the Short-Term Average of the L1-norm (i.e., rectified) signal

Test

The test form is only available for methods with primitive types or arrays of primitive types as parameters.

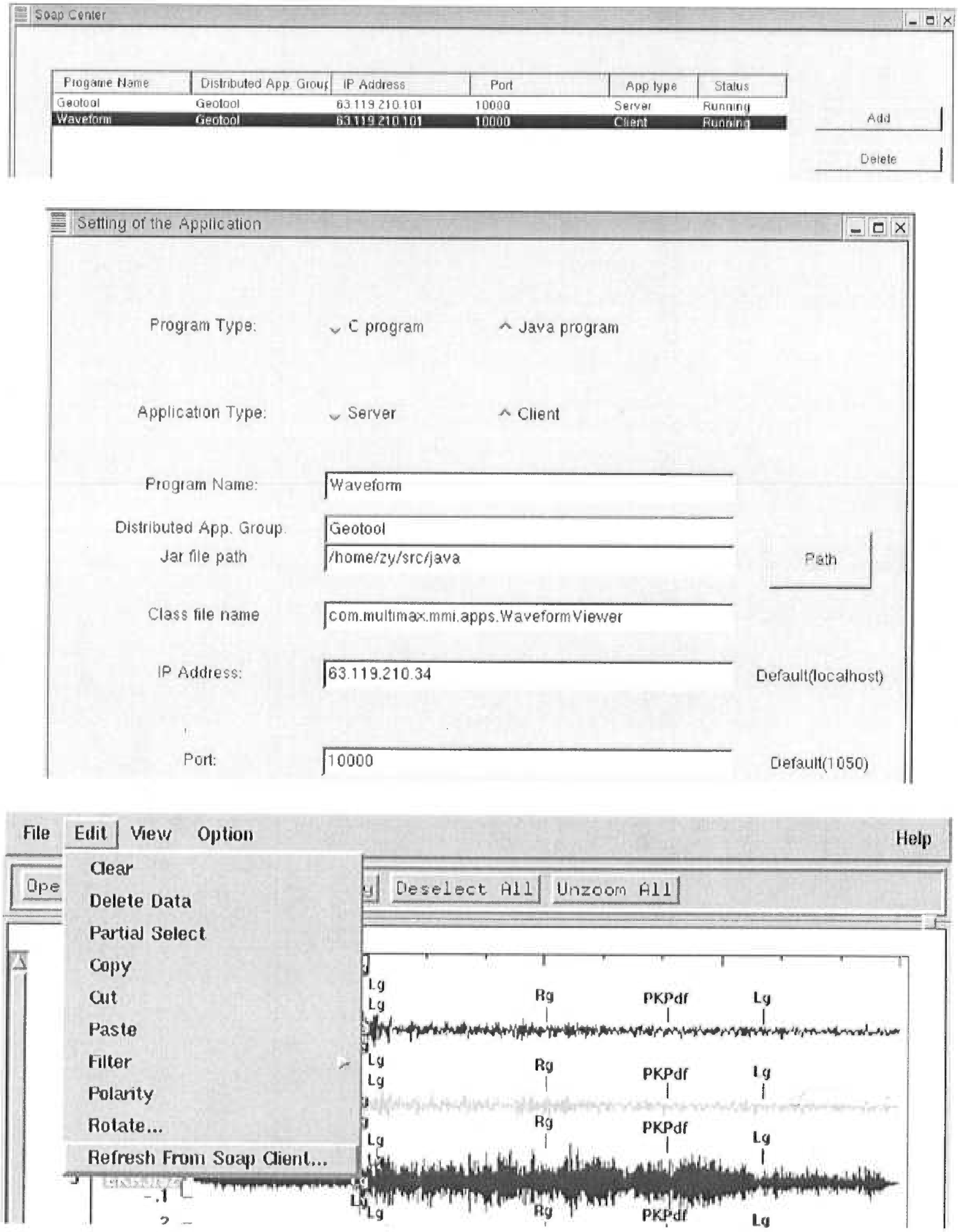
SOAP

The following is a sample SOAP request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /StatelessL1STAserver/StatelessL1STAserver.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.multimax.com/StatelessL1STAserver/BuildAndGetSTAseries"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.
  <soap:Body>
    <BuildAndGetSTAseries xmlns="http://www.multimax.com/StatelessL1STAserver">
      <arRawTimeseries>
        <float>float</float>
        <float>float</float>
      </arRawTimeseries>
      <numberPointsInWindowBeforeSTAtimePoint>int</numberPointsInWindowBeforeSTAtimePoint>
      <numberPointsInWindowAfterSTAtimePoint>int</numberPointsInWindowAfterSTAtimePoint>
```

Figure 16. *(Top)* Screenshot showing a browser window that exposes the *StatelessL1STAserver* Web service to examination by a potential user of this service. The hyperlink to the "Service Description" exposes the WSDL file that was shown in Figure 15 and that can be used to construct the interface to the proxy object. *(Bottom)* Screenshot showing that the hyperlink to the *BuildAndGetSTAseries* method of the C++ class exposes a template of the Simple Object Access Protocol (SOAP) message that will be sent to the Web service via HTTP when the calling program, such as *geotool*, makes a call to the *BuildAndSetSTAseries* method of the proxy object.



Progame Name	Distributed App. Grouฯ	IP Address	Port	App type	Status
Geotool	Geotool	63.119.210.101	10000	Server	Running
Waveform	Geotool	63.119.210.101	10000	Client	Running

Program Type: ○ C program ● Java program

Application Type: ○ Server ● Client

Program Name: Waveform

Distributed App. Group: Geotool

Jar file path: /home/zy/src/java

Class file name: com.multimax.mmi.apps.WaveformViewer

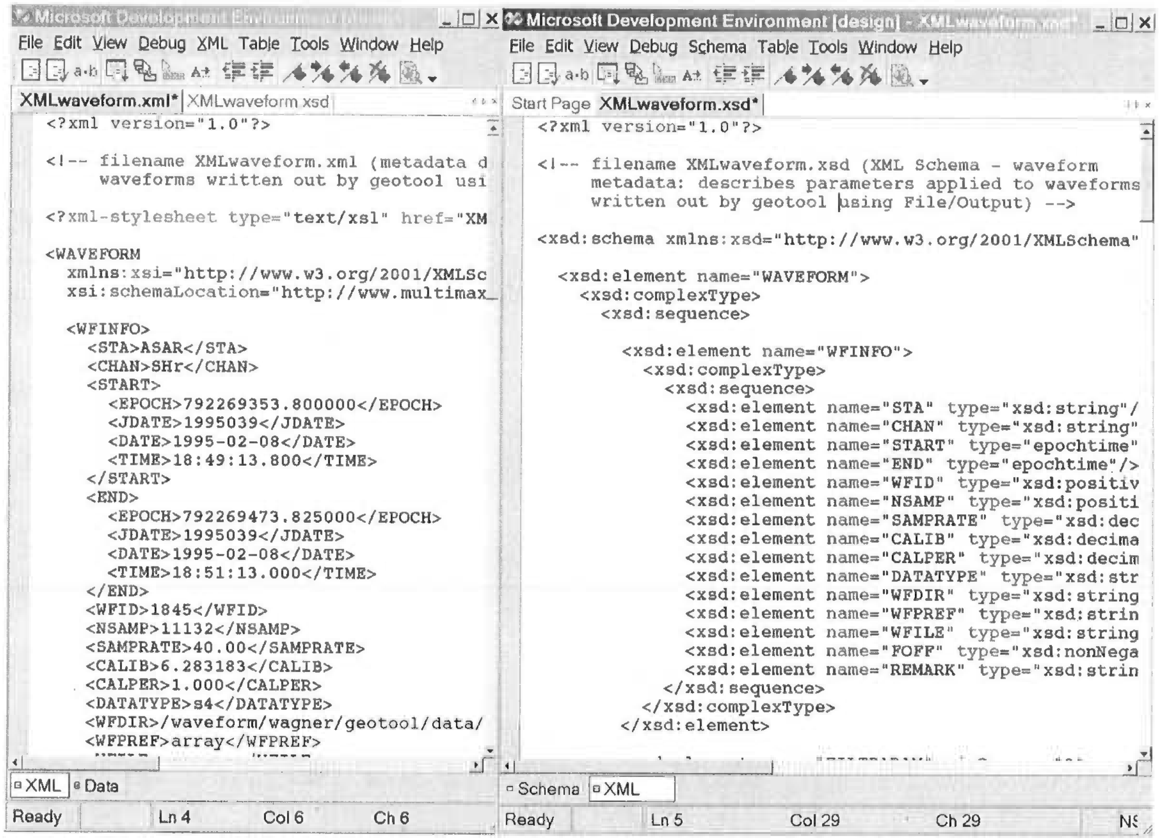
IP Address: 63.119.210.34 Default(localhost)

Port: 10000 Default(1050)

Figure 17. *(Top and middle)* Windows that set up a SOAP connection from the C-language program *geotool* to the Java program WaveformViewer. Although in this instance both programs are running on the same computer, this need not be the case. *(Bottom)* A window showing the menu item to refresh the selected waveform in the *geotool* display using data modified within the WaveformViewer program.

12. METADATA REQUIREMENTS

In seismic data analysis the issue of metadata is always an important one, since it is necessary to know how the raw data were obtained, how reliable were the measurements, what automatic and interactive processing were applied to the data, the thresholds beyond which the processing results are invalid, etc., before the data can be entered into the Knowledge Base and the events can be located, parameterized, and classified. As we have mentioned earlier, this need for a thorough description of the metadata is even more nearly critical for a distributed processing system than for a single large seismic analysis package, since in our reference event system the client program such as *geotool* will treat the independent analysis programs as black boxes. It will send data to other applicationns and receive results back, but the internal operations of these remote services must be regarded as unknown. It is therefore essential that the servers return to the clients not only the output of their algorithms but a thorough metadata description of all the values it returns and how they were derived. We have therefore undertaken the design of metadata records for all the processing routines within *geotool*, and we are extending that design to include external processing that will communicate with *geotool*. It is becoming the software industry standard to use XML for categorizing metadata, and this format is especially beneficial to us since it can be passed back and forth between clients and Web services via HTTP along with the XML SOAP RPC messages that instigate the data processing. Because Oracle and other RDBMS vendors are enabling their products to manage XML metadata records along with relational tables, we shall be able to store on the server the contents of the passed metadata files. Figure 18 shows the XML markup for a metadata file describing the waveform processing applied by *geotool*. Because the XML markup tags are by definition extensible, we must define the tags that will be used in all instances of waveform processing, and the schema for these tags (itself an XML document) is shown on the right-hand side of the figure. The design of this particular schema is illustrated in Figure 19. Although the XML markup is intended to be read principally by the data processing software, it can be presented in a form readily accessible by the *geotool* user, as is shown in Figure 20. The stylesheet that transforms the XML file into a presentation format is yet another XML document, and it is shown on the right-hand side of that figure.


Microsoft Development Environment
File Edit View Debug XML Table Tools Window Help
XMLwaveform.xml* XMLwaveform.xsd
XML Data
Ready Ln 4 Col 6 Ch 6
Microsoft Development Environment [design] - XMLwaveform.xsd*
File Edit View Debug Schema Table Tools Window Help
Start Page XMLwaveform.xsd*
Schema XML
Ready Ln 5 Col 29 Ch 29 NS

```
<?xml version="1.0"?>

<!-- filename XMLwaveform.xml (metadata d
     waveforms written out by geotool usi

<?xml-stylesheet type="text/xsl" href="XM

<WAVEFORM
  xmlns:xsi="http://www.w3.org/2001/XMLSc
  xsi:schemaLocation="http://www.multimax_

  <WFINFO>
    <STA>ASAR</STA>
    <CHAN>SHr</CHAN>
    <START>
      <EPOCH>792269353.800000</EPOCH>
      <JDATE>1995039</JDATE>
      <DATE>1995-02-08</DATE>
      <TIME>18:49:13.800</TIME>
    </START>
    <END>
      <EPOCH>792269473.825000</EPOCH>
      <JDATE>1995039</JDATE>
      <DATE>1995-02-08</DATE>
      <TIME>18:51:13.000</TIME>
    </END>
    <WFID>1845</WFID>
    <NSAMP>11132</NSAMP>
    <SAMPRATE>40.00</SAMPRATE>
    <CALIB>6.283183</CALIB>
    <CALPER>1.000</CALPER>
    <DATATYPE>s4</DATATYPE>
    <WFDIR>/waveform/wagner/geotool/data/
    <WFPREF>array</WFPREF>
```

```
<?xml version="1.0"?>

<!-- filename XMLwaveform.xsd (XML Schema - waveform
     metadata: describes parameters applied to waveforms
     written out by geotool using File/Output) -->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"

  <xsd:element name="WAVEFORM">
    <xsd:complexType>
      <xsd:sequence>

        <xsd:element name="WFINFO">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="STA" type="xsd:string"/
              <xsd:element name="CHAN" type="xsd:string"
              <xsd:element name="START" type="epochtime"
              <xsd:element name="END" type="epochtime"/>
              <xsd:element name="WFID" type="xsd:positiv
              <xsd:element name="NSAMP" type="xsd:positi
              <xsd:element name="SAMPRATE" type="xsd:dec
              <xsd:element name="CALIB" type="xsd:decima
              <xsd:element name="CALPER" type="xsd:decim
              <xsd:element name="DATATYPE" type="xsd:str
              <xsd:element name="WFDIR" type="xsd:string
              <xsd:element name="WFPREF" type="xsd:strin
              <xsd:element name="WFILE" type="xsd:string
              <xsd:element name="FOFF" type="xsd:nonNega
              <xsd:element name="REMARK" type="xsd:strin
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
```

Figure 18. The two windows on the left and right of the screenshot show excerpts from two separate XML files that are part of the characterization of the metadata describing the operations of the server application that performs the FK and bandwith filtering. On the left-hand side is the XML file itself that describes the waveform. The contents of the ".wfdisc" record in the CSS database schema are immediately obvious in this XML file, since the ".wfdisc" record is in fact metadata describing the waveform undergoing processing. An advantage of XML as a means for describing metadata is that it is an "eXtensible Markup Language", so the user can create whatever markup tags are appropriate to the vocabulary of the application being described, unlike a markup language such as HTML that is restricted to a pre-defined set of tags. Of course, extending the markup language by employing tags appropriate to seismology requires that these new tags, such as the "SAMPRATE", "START", "DATATYPE", etc., tags used in this XML file must be defined somewhere else. This is done in the schema file on the right-hand side, which is itself written in XML, that tells the software that "CALIB" is an XML-standard type (a decimal) and "END" is a user-defined type ("epochtime", which is further defined in this schema file). This schema corresponds to the design that is shown graphically in Figure 19.

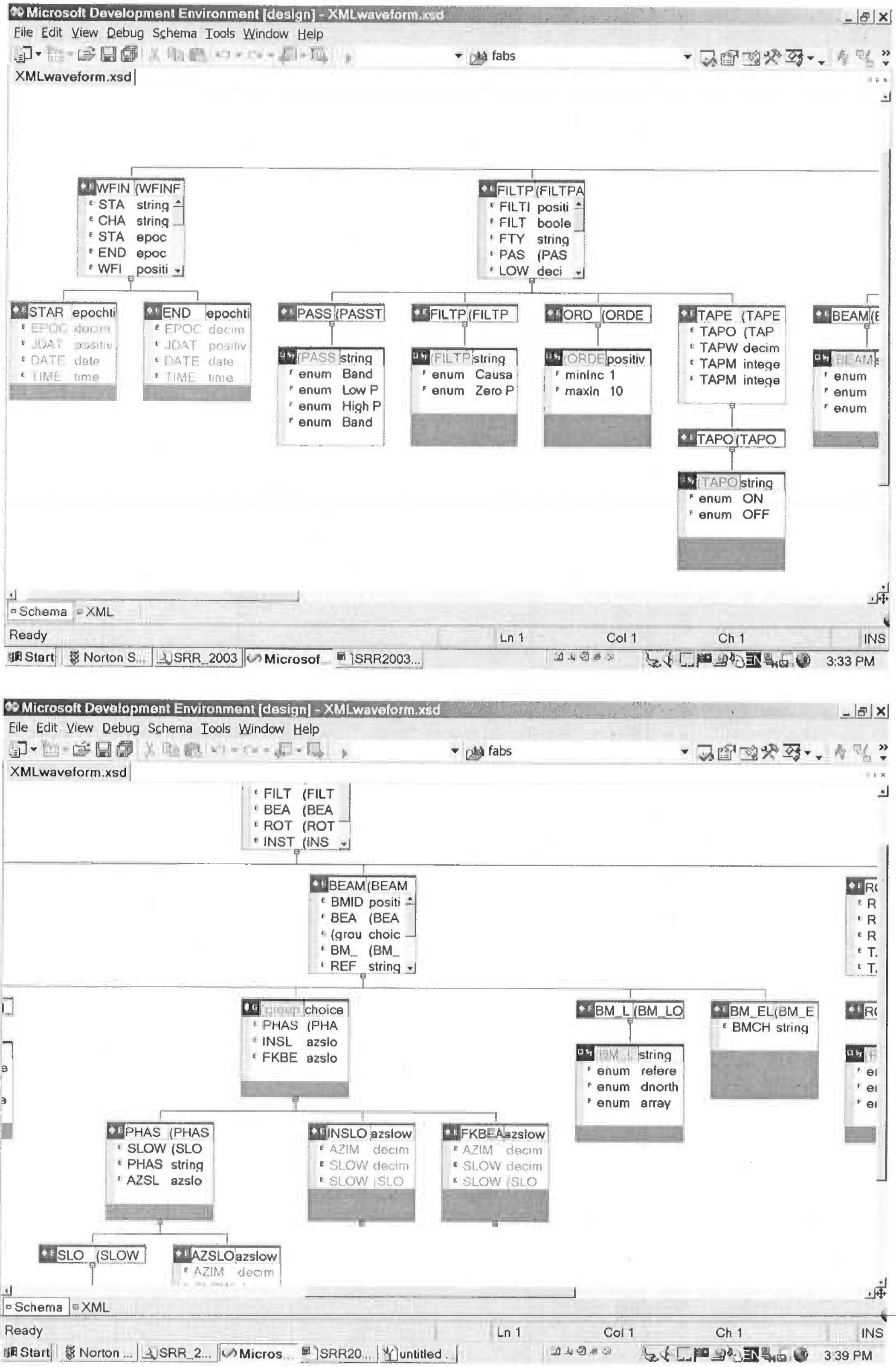
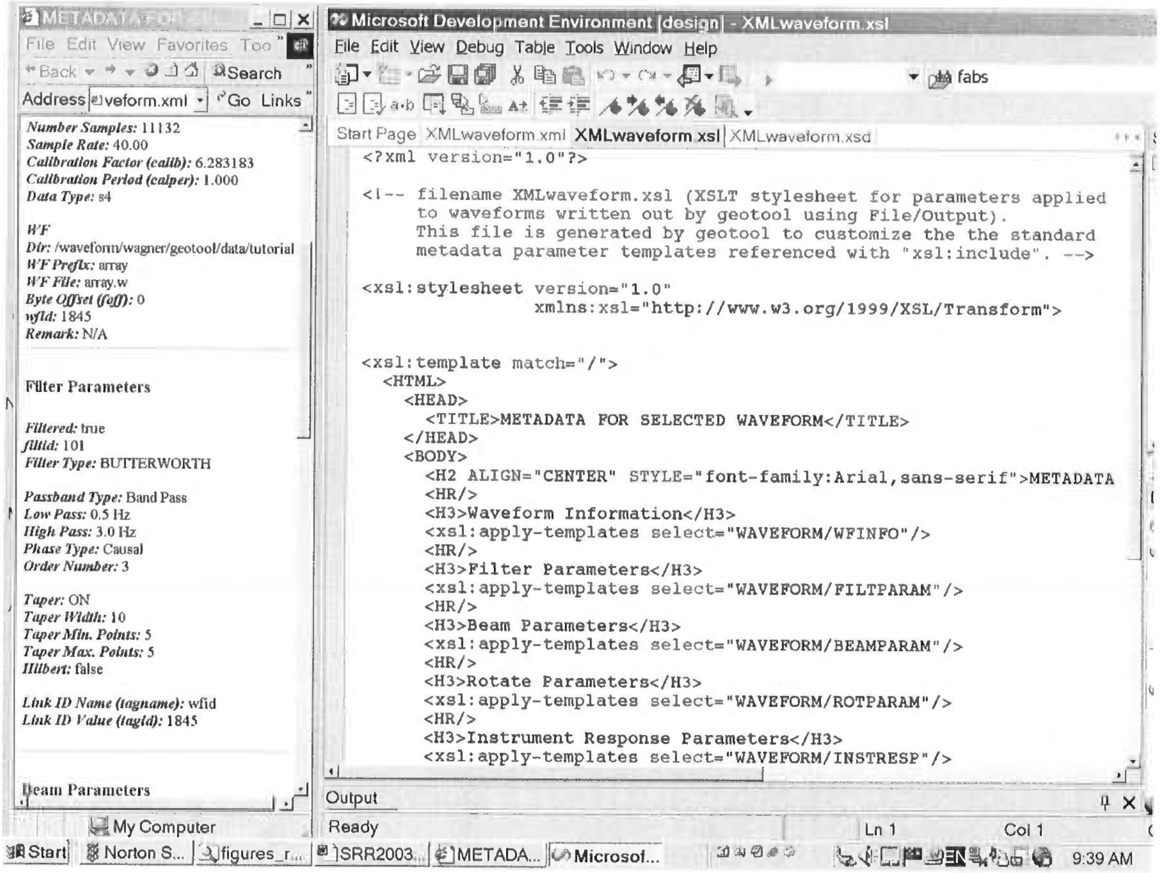


Figure 19. Two selected excerpts from the design layout of the waveform metadata XML schema that was shown on the right-hand side of the screenshot in Figure 18.


METADATA FOR ...
File Edit View Favorites Too
Back
Search
Address veform.xml
Go Links
My Computer
Microsoft Development Environment [design] - XMLwaveform.xsl
File Edit View Debug Table Tools Window Help
fabs
Start Page XMLwaveform.xml XMLwaveform.xsl XMLwaveform.xsd
Output
Ready
Ln 1
Col 1
Start
Norton S...
figures_r...
SRR2003...
METADA...
Microsof...
9:39 AM

Number Samples: 11132
Sample Rate: 40.00
Calibration Factor (calib): 6.283183
Calibration Period (calper): 1.000
Data Type: s4

WF
Dir: /waveform/wagner/geotool/data/tutorial
WF Prefix: array
WF File: array.w
Byte Offset (foff): 0
wfid: 1845
Remark: N/A

Filter Parameters

Filtered: true
filtid: 101
Filter Type: BUTTERWORTH

Passband Type: Band Pass
Low Pass: 0.5 Hz
High Pass: 3.0 Hz
Phase Type: Causal
Order Number: 3

Taper: ON
Taper Width: 10
Taper Min. Points: 5
Taper Max. Points: 5
Hilbert: false

Link ID Name (tagname): wfid
Link ID Value (tagid): 1845

Beam Parameters

```
<?xml version="1.0"?>

<!-- filename XMLwaveform.xsl (XSLT stylesheet for parameters applied
     to waveforms written out by geotool using File/Output).
     This file is generated by geotool to customize the the standard
     metadata parameter templates referenced with "xsl:include". -->

<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">


<xsl:template match="/">
  <HTML>
    <HEAD>
      <TITLE>METADATA FOR SELECTED WAVEFORM</TITLE>
    </HEAD>
    <BODY>
      <H2 ALIGN="CENTER" STYLE="font-family:Arial,sans-serif">METADATA
      <HR/>
      <H3>Waveform Information</H3>
      <xsl:apply-templates select="WAVEFORM/WFINFO"/>
      <HR/>
      <H3>Filter Parameters</H3>
      <xsl:apply-templates select="WAVEFORM/FILTPARAM"/>
      <HR/>
      <H3>Beam Parameters</H3>
      <xsl:apply-templates select="WAVEFORM/BEAMPARAM"/>
      <HR/>
      <H3>Rotate Parameters</H3>
      <xsl:apply-templates select="WAVEFORM/ROTPARAM"/>
      <HR/>
      <H3>Instrument Response Parameters</H3>
      <xsl:apply-templates select="WAVEFORM/INSTRESP"/>
```

Figure 20. The browser window on the left-hand side of the screenshot shows, in a format that can readily be understood by the *geotool* operator, the contents of the waveform metadata file that were shown in XML format in the window on the left-hand side of the screenshot in Figure 18. To the right of the browser window there is shown another window that displays the XSLT (Extensible Stylesheet Language – Transformation) file that was used to transform the XML metadata file into the visual presentation form that was shown within the browser window.

13. PROBLEMS ENCOUNTERED IN IMPLEMENTING WEB SERVICES

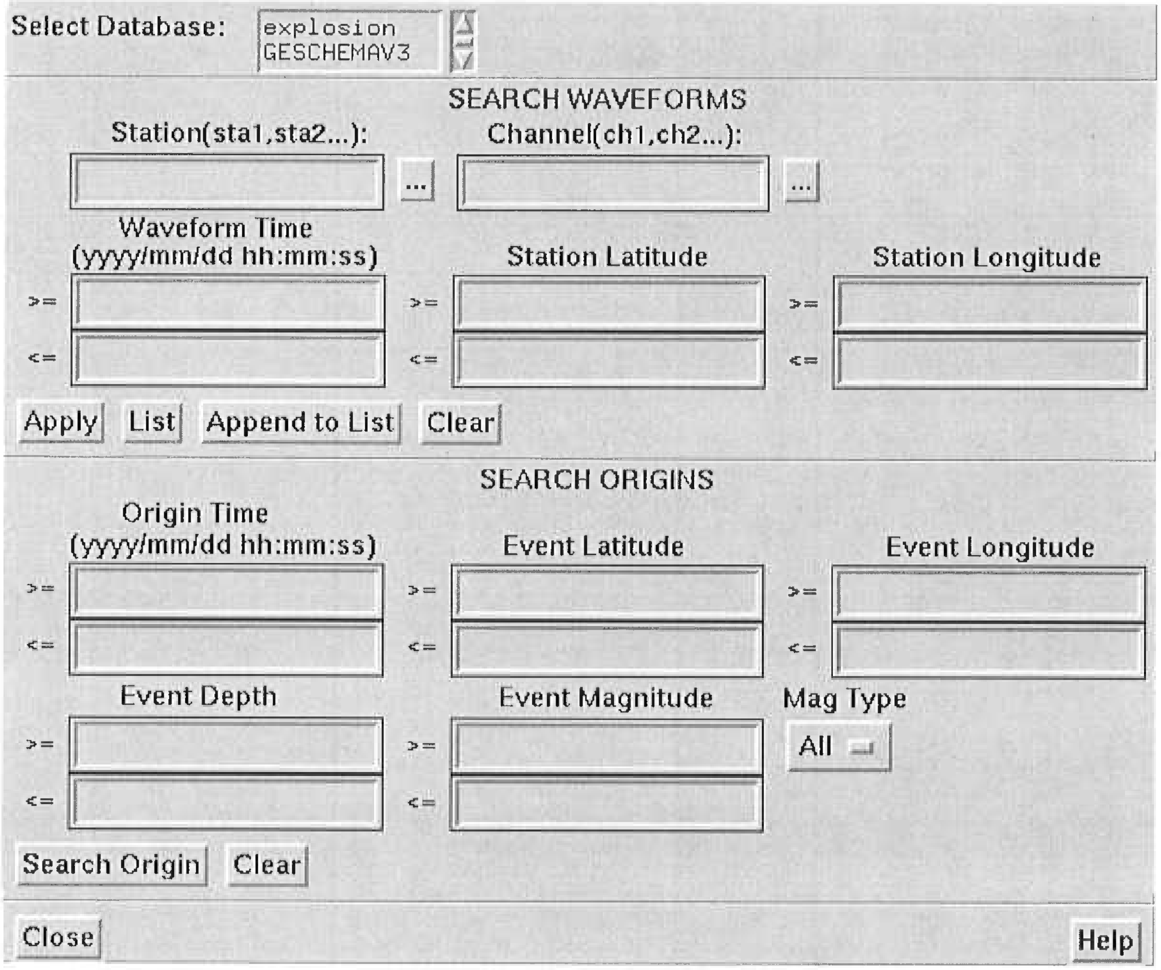
We have encountered difficulties in this project due to the paucity of satisfactory software for using SOAP with C or C++ programs on UNIX and Linux, and like much other geophysical analysis software (especially older code), *geotool* is in fact such a program. Although there are Java Web services packages that implement SOAP and that work with the commonly used open-source Apache Web server, we are restricted to using SOAP with C, and this significantly limits our options. The Microsoft SOAP toolkit makes it possible, albeit not especially easy, to use SOAP with C/C++ software on Windows (a task that now has been made considerably easier under the .NET platform by using ASP.NET and that will be made still easier in the next version of Windows by using a new technology called "Indigo", as we discuss below), but there is no corresponding commercially supported off-the-shelf technology for C/C++ on Unix or Linux other than certain expensive products that have been developed for enterprise-scale commercial applications. In our 2003 – 2004 work we therefore used the academic software *gSOAP* (van Engelen, 2005) to enable *geotool* to communicate using SOAP using the C language, but the process was not so simple as we would like it to be. This is a significant problem, since ease of use will be an important consideration by a scientist who wishes to integrate new and legacy research software. By far the most popular UNIX and Linux Web server tool for utilizing SOAP is the Java-based software *Axis*, which is developed and maintained by the open-source software organization Apache. Throughout the duration of this project we have awaited for Apache to deliver the promised C++ version of *Axis*, but it was not released until the spring of 2004. Our attempts to use that C++ version of *Axis* have been unsatisfactory, and we are not alone in that experience, so we feel this version 1.0 software release needs further improvements before we can interface it to *geotool*. Fortunately, a new version of *gSOAP* has now been released that generates header files that require less manual changing, so it is now a bit easier to use *gSOAP* to implement C/C++ Web services on UNIX and Linux than it was previously. However, the process is still far from transparent, and many or most scientists will not want to make use of it for routine purposes. We feel that we shall not be able to introduce a satisfactory commercial product based on this technology until the process of establishing data communications among applications "on the fly", rather than "hard wired" on a permanent or at least seldom changing basis, becomes considerably simpler. A scientist will want to concentrate on doing the programming for a new algorithm to analyze reference events instead of devoting significant time and energy to setting up the "plumbing" to make that new algorithm communicate with *geotool* (or any other program) so that the algorithm can then be applied within a visual data processing environment.

14. PROBLEMS ENCOUNTERED WITH MODIFYING *GEOTOOL*

As is described above, we found adding functionality to *geotool* by modifying the program to send and receive SOAP messages to be a more complicated process than most scientists would be willing to do for routine modifications, and so the process needs to be simplified. Perhaps an even bigger problem, however, is that not only is it cumbersome to modify *geotool* to communicate with other applications, it is difficult to modify *geotool* to make use of the information that those other applications would send back to it. This problem is a consequence of the program's architecture, namely an early 1990s-style large C-language program with many internal interdependencies. Thus, changing the code to modify a feature or add a new one is liable to cause unanticipated "side effects" in a different feature. In a reference event system, it will be necessary to modify *geotool* by adding widgets that permit the user to manipulate the displayed data and select seismogram segments, make measurements, enter parameter data, and then dispatch those values to another application which will in turn send its results back to *geotool* for display, a process that will require further modification to add new widgets (or expand existing ones) to incorporate those results. Each of these modifications to the code is liable to cause problems and introduce bugs. In many cases it is possible to accomplish changes by editing a table of X Windows resources, and this process is certainly preferable since it requires changing only an ASCII file outside of *geotool* instead of the source code. However, most modifications require some re-programming, and the architecture of *geotool* is not sufficiently robust for changes to be made easily.

Evidence of this problem can be seen in the host of changes we have made to *geotool* during this project simply to fix existing bugs in the code or to expand the functionality of many existing operations, since these changes were all internal to *geotool* and involved no communications with any other applications. An extensive list of bug fixes was generated by intensive testing of the existing code, and the mere fact that these bugs have persisted in a program that is well over a decade old shows how hard the code is to modify successfully. In spite of the problems in making changes to a large C program, we did correct those bugs, and we added many features that users have specified as being needed to enhance the utility of *geotool* and improve its ease of use. The process (whether a highly visible one such as the addition of the Oracle interface shown in Figure 21, or a minor and seemingly trivial one such as a bug fix that should be straightforward) has been a tedious one, however, and it bodes ill for our intended use of *geotool* as a central application in a reference event system, since we would like it to serve as a graphical user interface, and it would thus require frequent modification to perform new data manipulation and display operations. We acknowledge that a preferable approach would be to use, at the very least, a program that runs within a modern software "platform" like Java or .NET that handles the low-level systems operations for it, instead of a legacy program like *geotool* that must make UNIX kernel calls and other systems-level operations. It would be better still to use a program running within that platform that is far more flexible than is a monolithic C code like *geotool*, such as an object-oriented program organized into independent components (such as Enterprise Java Beans) that can be swapped in and out just as hardware components can be. In this regard, *geotool* is significantly inferior to a modern seismic analysis program like *MatSeis* (Young, 2001), which minimizes many of these difficulties because it is built upon a robust commercial software platform (MATLAB) that offers to the programmer an extensive suite of graphical and computational components that adhere to this architecture. In the long term it would be easier to build new "thin client" programs (*i.e.*, small object-oriented codes that rely on an underlying software platform for systems-level operations and that have little functionality other than to serve as a user interface for server-side components and applications) to act as intermediaries in a seismic reference event system than continually to make major modifications to a large, fragile program like *geotool* every time that new functionality is required. In the following section we discuss modern trends in software development that should be exploited to construct the type of programs that would be better suited for use within a seismic reference event system than is *geotool.*

Because during the course of this project we have made extensive changes to the *geotool* code base (unrelated to Web services or other communications with external applications but instead confined to *geotool* itself), we have revised the 1995 version of the documentation of the program to reflect the bug fixes and enhancements that have been made to the code in the last 10 years, especially the many changes that we made during this project. The updated documentation is included as an appendix to this report.



Select Database: explosion GESCHEMAV3

Mag Type: All

Figure 21. Among the bug fixes and enhancements that we have added to *geotool* during this project is a graphical interface to the Oracle database that allows the user to query the database and retrieve waveforms of interest. This interface was added to *geotool* through conventional C-language Motif widget callback functions, and considerable difficulty was experienced in its implementation due to unanticipated "side effects" in parts of the program unrelated to the changes that were made in the code. This behavior is typical of large monolithic C-language programs, and it points out the need for, at the very least, object-based software design or, preferably, a component-based architecture.

15. INTEGRATION OF TOOLS FOR USING THE KNOWLEDGE BASE

As we have described, our intention in this project was to develop a system architecture that could be used for analyzing reference events, especially for building the NNSA Knowledge Base, and then to use that architecture as the basis for a commercial seismic data analysis product. Difficulties in using SOAP with C software under UNIX or Linux (until more mature SOAP software becomes available for this language on this platform) has kept this goal from being realized with the necessary ease of use for the scientist. Under the current technology, users will not want to use C-language programs for Web services on UNIX/Linux, at least not for rapid development and testing of new algorithms. However, a system of inter-process communications has in fact been constructed specifically for the NNSA Knowledge Base that allows the integration of its software tools that conduct seismic analysis (Merchant *et al.*, 2004). This system takes the C-language code for seismic analysis and adds a Java "wrapper" to serve as the interface. These Java wrappers call the C code using Java Native Interface (JNI), a built-in feature of Java for invoking C functions, and the wrappers communicate with one another using Java Remote Method Invocation (RMI), another built-in feature of Java that implements a version of CORBA specifically for the Java platform. The C-language seismic programs thus communicate with one another through these Java intermediaries. In the current state of technology for utilizing Web services under UNIX/Linux using C, this Java RMI approach would in fact seem preferable, and it is appropriate that it be the architecture that is used for integration Knowledge Base tools. Perhaps this situation will change when the C-language version of Apache Axis becomes more mature, but that of course remains to be seen. The "Navigator" software designed and built by Merchant *et al.* (2004) thus makes our original goal of using Web services for the data communications in constructing a reference event analysis system for the Knowledge Base certainly less important. However, this Small Business Innovation Research (SBIR) project has as its ultimate goal the construction of seismic data analysis software that could be sold commercially, since commercialization of Government-sponsored research is the objective of the SBIR program. Bringing commercial software to market is still our intention, and we feel the Web services approach will be an important part of the architecture for a commercial product that will be used for seismic data analysis. However, the success of this approach depends on certain other developments in the software industry, as we shall now discuss.

16. RECENT DEVELOPMENTS IN SERVICE-ORIENTED ARCHITECTURE

The need to integrate separate business processes, such as shipping products and submitting invoices, has long been a requirement in commerce, and large and complex Management Information Systems (MIS) have been sold for many years to address that need. More recently, these systems have evolved to a higher level by unifying all the separate "information stovepipes" within a corporation, for instance by allowing the warehouse inventory system to initiate requisition processes in the purchasing system. Most of these Enterprise Resource Management software systems make use of data warehouse systems, a specialized type of database for Online Analysis Processing (OLAP) that used to be a separate technology but that is now being incorporated into conventional Transactional Processing relational database management systems, such as the latest versions of

Oracle and SQL Server. These commercial software systems for integrating separate processes are conventionally implemented using a three-tier client-server architecture wherein desktop computers function as "thin clients" for graphical user interaction, a database server allows all applications to have access to the same data (to the extent allowed by security controls), and in the middle tier the actual business rules are carried out by programs running on an application server. The communications between the desktop clients and the application server, and between the application server and the database server, are carried out through tight coupling of the tiers. The inter-tier communications uses socket connections directly or underneath Remote Procedure Calls, CORBA, or Java Remote Method Invocations (RMI), which is the Java-specific implementation of CORBA that is used in the "Navigator" tool developed by Merchant *et al.* (2004) for performing data communications among programs that access the NNSA Knowledge Base.

Currently, however, most enterprises are starting to implement, or at least to experiment with, the concept of replacing this tight inter-tier coupling by using instead a loosely coupled architecture for integrating separate processes, namely Service-Oriented Architecture (SOA). One of the principal motivations for using SOA is that it allows a still higher level of integration, not just within a department or throughout the enterprise but among separate enterprises. The concept underlying SOA is that a company can use a Web server tier to respond to requests for services from individual applications whether those applications are running locally or at another site. That way the purchasing system of one company can initiate a process by the order fulfillment system of another company, without the need for manually re-entering the data from the first company's purchase order (generated by that company's application server tier) into the back-end database of the other company so that the second company's application server can process it. Obviously, allowing data to flow directly from one company to another cannot be accomplished by a tight coupling of the two computer systems, due to security concerns. A non-intrusive data communications model is required for SOA that uses an open protocol such as HTTP for transmitting data in ASCII format instead of in a binary format that can contain viruses or spyware. As we have described in our discussion of Web services, such a model is provided by SOAP and XML, and in fact the acronym "SOAP" is sometimes now re-interpreted to mean "Service-Oriented Architecture Protocol" instead of its original meaning of "Simple Object Access Protocol". Corporations are actively addressing methods to expose their business processes through SOA so that they can be utilized by both local and remote applications, and a large software industry is emerging to meet that demand. Our work in the second phase of this project began at about the same time as SOA became popular in industry, and in future work we expect to be able to make use of forthcoming software products, both commercial and open-source, that will considerably expedite application integration using SOAP.

A large part of the problem in our use of SOAP has been that not only are programs such as *geotool* legacy software, but the fundamental architecture of developing software that runs at a level close to the operating system (in our case, Linux) is itself a legacy design. Modern software is for the most part written at a higher level, specifically one that targets a "platform" such as Java or .NET rather than the underlying operating system. The platform interposes a software layer that operates as a virtual machine to handle the low-level communications protocols. We hope that the available tools such as *gSOAP* and Apache's *Axis* for C/C++ will become easier to use on Linux and UNIX, but for now it is considerably simpler to implement Web services within the Java and .NET platforms where much of the data communications is handled by the platform software instead of by the applications.

Furthermore, the ease of use of the platforms is escalating rapidly. Because Java has its own CORBA-style RMI software for performing Java-to-Java communications, and since it also supports CORBA directly, Java software used for SOAP communication has heretofore been provided only by packages external to the core language. With the release in July 2004 of Java 1.5 (now re-labeled Java 5), however, JAX-RPC (Java API for XML-based Remote Procedure Calls) has been incorporated into the client-side platform, and this same change will be made to the next release of the server-side platform. Moreover, in 2005 Sun will release new software development tools that will make the use of Web services easier, and the open-source Eclipse organization is doing the same. On the .NET platform, the implementation of Web services has always been made relatively easy (although still not transparent) by the use of ASP.NET (a technology that replaced Microsoft's Active Server Pages for its earlier COM platform), since much of the low-level coding of both the proxy client and the proxy server is generated automatically by ASP.NET in a fashion that requires no re-coding by the programmer. This processing is becoming easier still, since in July 2004 Microsoft released the first beta distribution of "Whidbey", its new Integrated Development Environment for .NET programming (which will be released officially as "Visual Studio 2005" next year), and "Whidbey" contains a new tool with the code name "Whitehorse" that expedites the design and construction of SOA software. Unlike the Java platform, the .NET platform supports multiple languages, and Web service clients and services can be constructed using the version of C++ that runs within .NET. (The syntax for C++ under .NET, and the scope and utility of that language for the .NET platform, will also be considerably enhanced under "Whidbey".) Web services can be implemented in .NET even by using Fortran 95 with selected Fortran 2003 object-oriented extensions, and the commercially available Lahey-Fujitsu v7.1 compiler accomplishes this. We anticipate that the introduction of JAX-RPC into the Java 1.5 platform, the introduction of "Whidbey" for the .Net platform, the release next year of new Java software development tools by Sun and by Eclipse, and the continued evolution of the lower-level *gSOAP* and Apache *Axis* tools for C/C++ application on Unix and Linux will all make the implementation of Web services considerably easier within the next year or so than it has been heretofore.

17. FUTURE DEVELOPMENTS IN ARCHITECTURE FOR CLIENT CODE

As we have noted, most modern desktop applications, unlike legacy program such as *geotool*, are written using an object-oriented design that permits them to be altered with less danger of introducing unanticipated side effects than is possible in a large C language program. An important trend is that the platforms on which these modern programs run, such as .NET and Java, are themselves object-oriented, so modern application development software readily enables desktop client programs to be constructed that are more nearly suitable for easy modification than *geotool* is. This advantage is certainly demonstrated by the ease with which MatSeis (Young, 2001) can be modified, since it is built on a commercial object-oriented platform (namely MATLAB). We plan to continue development by migrating functionality from *geotool* to new client and server applications that will be sufficiently flexible for easy incorporation into a seismic analysis system that can be tailored for the needs of analyzing specific data sets.

A particular software platform that will likely be useful for constructing the sort of highly flexible distributed systems that will be needed for integrating seismic analysis applications is the next version of Microsoft Windows, which has the code name "Longhorn" and will likely be released as "Windows 2006". Although the software that is available to developers is still in its pre-alpha release, it is possible to begin investigating its utility for constructing new client and server applications that may eventually replace *geotool*, and we intend to pursue that investigation. "Longhorn" offers a particular benefit to the construction of distributed applications in that it incorporates a new data communications software system called "Indigo" that uses XML and SOAP messages not only for Web services but also for inter-process communication within the local environment. It will thus be transparent to the developer whether function calls are being made across threads or across computer systems. More importantly for the ease of use, and hence for the likelihood that scientists will write their code in a manner that allows its incorporation into a distributed system, the "Indigo" software will automate most of the coding required to perform data communications using SOAP. This will be an important change from the current situation with CORBA and Web services, and we feel it is important that work begin on building new programs using this new data communications software. Another significant feature of "Longhorn" is that the screen graphics for drawing windows, window controls (such as menus, buttons, etc.), and presentation graphics within windows (such as plots of seismograms, spectra, etc.) will be performed using a new Application Programming Interface (API) that is based on XML. Moving to XML-based displays and screen graphics from those built using Motif, Xt, and Xlib on Linux and UNIX (as is *geotool*) will be a significant change in the construction of seismic analysis software, and it is potentially a valuable feature. In particular, the use of XML graphics may make it possible for application servers to generate not only processed data, such as a seismogram that has been subjected to a particular filter, but also to generate presentation graphics displaying the results, and those graphics would be returned to the client program as an XML message. This would alleviate the client application from having to know how to display the processing results of the server application, so it could act as a true "thin client" for user interaction since it would not have to maintain the huge graphics and processing overhead that makes an application like *geotool* so difficult to modify. We intend to investigate this possibility actively before the anticipated 2006 release of "Longhorn".

18. CONCLUSIONS

We believe that a distributed system of individual applications for seismic data analysis, all communicating to a client program acting as a graphical user interface, is the appropriate design for a software system to identify and study seismic reference events that are candidates for inclusion in the Knowledge Base. However, the need for graphical interaction with each of the component programs means that they are perhaps better operated in a peer-to-peer mode than within a network of Web services. In either case, the requirement to use legacy C-language code in UNIX or Linux as major components of this system imposes significant limitations upon the ease with which components may be added to this system and modified. Whether these legacy components are wrapped within Java interfaces so that they can communicate through RMI or JAX-RPC, or whether they communicate directly through middleware such as CORBA or *gSOAP* (or, after it becomes more nearly stable, through Apache *Axis* for C/C++), establishing the data communications software for these legacy applications and modifying them to use that communications software and to handle properly the data and processing results that will be transmitted to them, remains a difficult task that most scientists would not undertake for routine software applications maintenance and modification. Newer data communications software such as ASP.NET and the inclusion of JAX-RPC in the core Java platform make the production of data communications software easier (for programs other than C-language applications running under Linux, which remain problematic), and future systems such as the "Indigo" software that will be part of the next version of Microsoft Windows will make much of this programming transparent to the developer.

The problem will still remain, however, of modifying the component applications such as *geotool* to handle new data and perform new applications without having those modifications break existing features in the code. That problem can be solved only by migrating the graphical display functionality of large monolithic programs like *geotool* into component-based programs where one component can be modified without changing others. This component-based graphical display program should then use Service-Oriented Architecture to operate as a "thin client" that handles user interaction but that does not carry the overhead of extensive computational code, since those computations would be performed by application server codes that could be modified separately. It would then be far easier to modify the component-based display software to handle the new data returned from the application servers than it is now, where changing *geotool* to handle new data is a difficult task.