EFFICIENT INTEGRATION OF OLD AND NEW RESEARCH TOOLS FOR AUTOMATING THE IDENTIFICATION AND ANALYSIS OF SEISMIC REFERENCE EVENTS

Wilmer Rivers[1], Robert A. Wagner[1], Joey Chen[1], Eric Siu[1], Craig A. Schultz[2], Douglas A. Dodge[2], and Gregory Pope[2]

Multimax, Inc.[1]; Lawrence Livermore National Laboratory[2]

Sponsored by National Nuclear Security Administration
Office of Nonproliferation Research and Engineering
Office of Defense Nuclear Nonproliferation

Contract No. DE-FG02-01ER83218[1] and W-7405-ENG-48[2]

ABSTRACT

The selection and study of reference events for inclusion in the National Nuclear Security Administration (NNSA) Knowledge Base (KB) require the application of a much broader suite of seismic analysis software than do 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 interprocess 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 interprocess communications.

OBJECTIVE

Scientists use a variety of stand-alone computer programs to analyze and identify seismic events for the monitoring of the nuclear test ban, 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 KB 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. Ideally, 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. We have investigated the use of this architecture for the processing of seismic data, by using the large C-language program *geotool* (Henson, 1993; Coyne and Henson, 1995) as the primary tool for user interaction and by determining how difficult it is to add new functionality to *geotool* by using software architecture that is based on distributed processing.

RESEARCH ACCOMPLISHED

Use of CORBA for extending *geotool*

Figure 1 shows one manner in which a distributed architecture can be used in practice, and it is the method that we used in the first phase of our investigation (Rivers et al., 2002). In that phase we used CORBA (a client-server architecture commonly used in the 1990s for large-scale business systems) as the means for data communications between the seismic analysis program *geotool*, operating as a data server running under Linux, and a Java program called *WaveformViewer,* operating as a client application on a Windows computer. (*WaveformViewer* is a program that displays a waveform on the screen using Java "Swing" graphics but that also allows the operator to perform some signal-processing operations such as digital filtering on the data.) CORBA has a particular advantage for our purposes, in that because much business software in the 1990s was written in C under UNIX, CORBA can be made to interface with a program like *geotool*. As will be discussed below, much modern software for data communications is not suitable for that environment, which remains the most common one for software performing nuclear monitoring data processing, especially for legacy applications.

We wrote software called CORBA Center to specify that *geotool* on a Linux computer will operate as a data server and that the Java program *WaveformViewer* on a Windows computer will be a client. As is shown in the diagram in Figure 1, we modified *geotool* by adding new callback functions that allow it to communicate with an Object Request Broker (ORB). (Since *geotool* is running under Red Hat's distribution of Linux, we can make use of the open-source ORB known as ORBit that is bundled as a part of the distribution. For *geotool* running instead under Solaris, we could use 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 *WaveformViewer* by invoking CORBA Center. The CORBA Center utility expedites the process of setting up the data connection by effectively acting 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 (such as FTP and HTTP) that allows an ORB on one platform to communicate with one on another platform. Note that it is irrelevant that *geotool* is written in C and *WaveformViewer* is written in Java, since IIOP is platform-independent. We modified the *WaveformViewer* 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 release JDK 1.2. 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 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 reuse that same interface for as many different *geotool* client-server applications as possible.

The architecture shown in Figure 1 may be a bit misleading, since *geotool* is shown operating as a data server and *WaveformViewer* as a client application. The goal of this investigation, however, is to use *geotool* as a graphical interface for user interaction (i.e., as a client), and to use external software as application servers delivering requested information and processing results to that client. For the CORBA architecture in Figure 1 this distinction is not critical, since CORBA establishes a socket connection that is actually being used in more nearly a peer-to-peer mode, with a scientist performing graphical operations on both the *geotool* and *WaveformViewer* sides by switching back and forth between working on the Linux workstation and the Windows PC (both of which must therefore be physically accessible to the scientist). In practice, this design would be implemented by running both *geotool* and the Java *WaveformViewer* within separate windows on the same Linux workstation, instead of running the Java *WaveformViewer* on a separate Windows computer. In the next section we shall examine a more typical scenario, wherein the distinction between client and server is sharper, and *geotool* does indeed operate as a graphical client of remote server applications.

Use of Web Services for Extending *geotool*

In the second phase of our investigation, we replaced the tight coupling of applications with a loose coupling architecture, namely Web services. In this configuration, application programs are treated as URLs that are accessed over an intranet or the Internet by sending XML messages over simple HTTP connections. The format of these XML messages includes the necessary data for the service to be performed and a remote procedure call, expressed in ASCII text rather than binary code, as specified by the SOAP standard. After the Web service application performs the data-processing (and/or data-retrieval) service for which it was invoked, it returns its results to the client application through another SOAP-formatted XML message over HTTP. The tight binary coupling of the client and server tiers that characterize CORBA is thus replaced by an ASCII-based messaging system. The concept is analogous to that of the World Wide Web: through the transmission of XML messages, the Internet can become not only a Web of information but also a Web of services. A particular benefit of using XML and SOAP to transmit messages between a client and (multiple) servers is that because the messages are ASCII, and the wire protocol is HTTP, they can pass through firewalls that would block attempts by remote systems to transmit binary messages and invoke operating system actions through CORBA. The trade-off for the flexibility of the loose coupling is that, by operating at a higher level of abstraction in the TCP/IP software stack, Web services are less efficient than a tight coupling architecture like CORBA. In particular, since the data transmitted to and received from the Web services are ASCII encoded, and since they are augmented with tags for XML markup, the messages that are sent over the intranet or Internet are considerably larger than the corresponding binary data would be. An example of a SOAP message for performing a signal-processing operation on a time-series array (such as a seismogram) is shown in Figure 2.

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. 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, 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 work we therefore used the academic software *gsoap* to enable *geotool* to communicate using SOAP, 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 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 waited 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 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.

Problems with Incorporating *geotool* into a Reference Event Analysis System

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 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 reprogramming, 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 and 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 (illustrated by the addition of the Oracle interface shown in Figure 3) 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 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, 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* (Hart, 2004), 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 to continually 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 than *geotool* for use within a seismic reference event system.

Recent Developments in Service-Oriented Architecture for Commercial Software Systems

The need to integrate separate business processes, such as shipping products and submitting invoices, has long been a requirement in commerce, and large, complex management information systems 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 use socket connections directly or underneath remote procedure calls, CORBA, or Java remote method invocations (RMI), which is a Java-specific implementation of CORBA.

Currently, however, most enterprises are starting to implement, or at least to experiment with, a loosely coupled architecture for integrating separate processes, namely 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 nonintrusive data communications model is required for an 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. 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." Corporations are actively addressing methods to expose their business processes as Web services 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 Phase II 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 opensource, 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 for 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, 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 application programming interface [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 in a fashion that requires no recoding by the programmer. This processing is becoming easier still, because on July 1, 2004, Microsoft released the first beta distribution of "Whibdey", its new integrated development environment for .NET programming (which will be released officially as *Visual Studio 2005* next year). "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 more efficient within the next year or so than it has been heretofore.

Future Developments in System Architecture for Desktop Clients

As we have noted, most modern desktop applications, unlike legacy programs 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*. 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 interprocess 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 API that is based on XML. Moving to XML-based displays and screen graphics from those built using Motif and Xlib on Linux and UNIX or using COM widgets or .NET Forms and GDI+ or Win32 low-level graphics on Windows, will be a significant change in the construction of seismic analysis software and is a potentially 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 and not 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."

CONCLUSIONS AND RECOMMENDATIONS

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 KB. 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 properly handle 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 functionality of large monolithic programs like *geotool* into object-based programs, preferably "thin clients" that handle user interaction but that do not carry the overhead of extensive computational code.

ACKNOWLEDGEMENTS

We are indebted to Floriana Ryall for her many extensive and insightful contributions to the evaluation of problems with the operation of *geotool* and the identification of useful and important extensions to its current capabilities.

REFERENCES

Carter, J., R. Bowman, K. Biegalski, J. Bohlin, M. Fisk, R. Carlson, W. Farrell, B. MacRitchie, and H. Magyar (2001), IDC Documentation Database Schema Revision 3, Center for Monitoring Research User Guide on-line document (available at http://www.pidc.org/librarybox/idcdocs/downloads/511r3ab.pdf).

Coyne, J. M. and I. Henson (1995), *Geotool* Sourcebook: User's Manual, Phillips Laboratory report PL-TR-96-2021.

Hart, D. (2004), *MatSeis* User's Manual, version 1.8, Sandia National Laboratories on-line document (available at https://www.nemre.nnsa.doe.gov/prod/nemre/fileshare/matseis-1.8_manual.pdf).

Henson, I. (1993), "The *Geotool* Seismic Analysis System," in Proceedings of the 15th Annual Seismic Research Symposium, September 8–10, 1993, Phillips Laboratory report PL-TR-93-2160.

Rivers, D. W., C. A. Schultz, and D.A. Dodge (2002) "Efficient Integration of Old and New Research Tools for Automating the Identification and Analysis of Seismic Reference Events", in *Proceedings of 24th Seismic Research Review - Nuclear Explosion Monitoring: Innovation and Integration*, LA-UR-02-5048, Vol. 2, pp. 882-891 (available at http://www.nemre.nnsa.doe.gov/srr/2002/screen/07-05.pdf).

Rivers, D. W., R. A. Wagner, E. Siu, C. Chen, C. A. Schultz, D. A. Dodge, and G. Pope (2003), "Efficient Integration of Old and New Research Tools for Automating the Identification and Analysis of Seismic Reference Events", in *25th Seismic Research Review - Nuclear Explosion Monitoring: Building the Knowledge Base*, LA-UR-03-6029, pp. 738-747 (available at http://www.nemre.nnsa.doe.gov/srr/2003/papers/08-10.pdf).

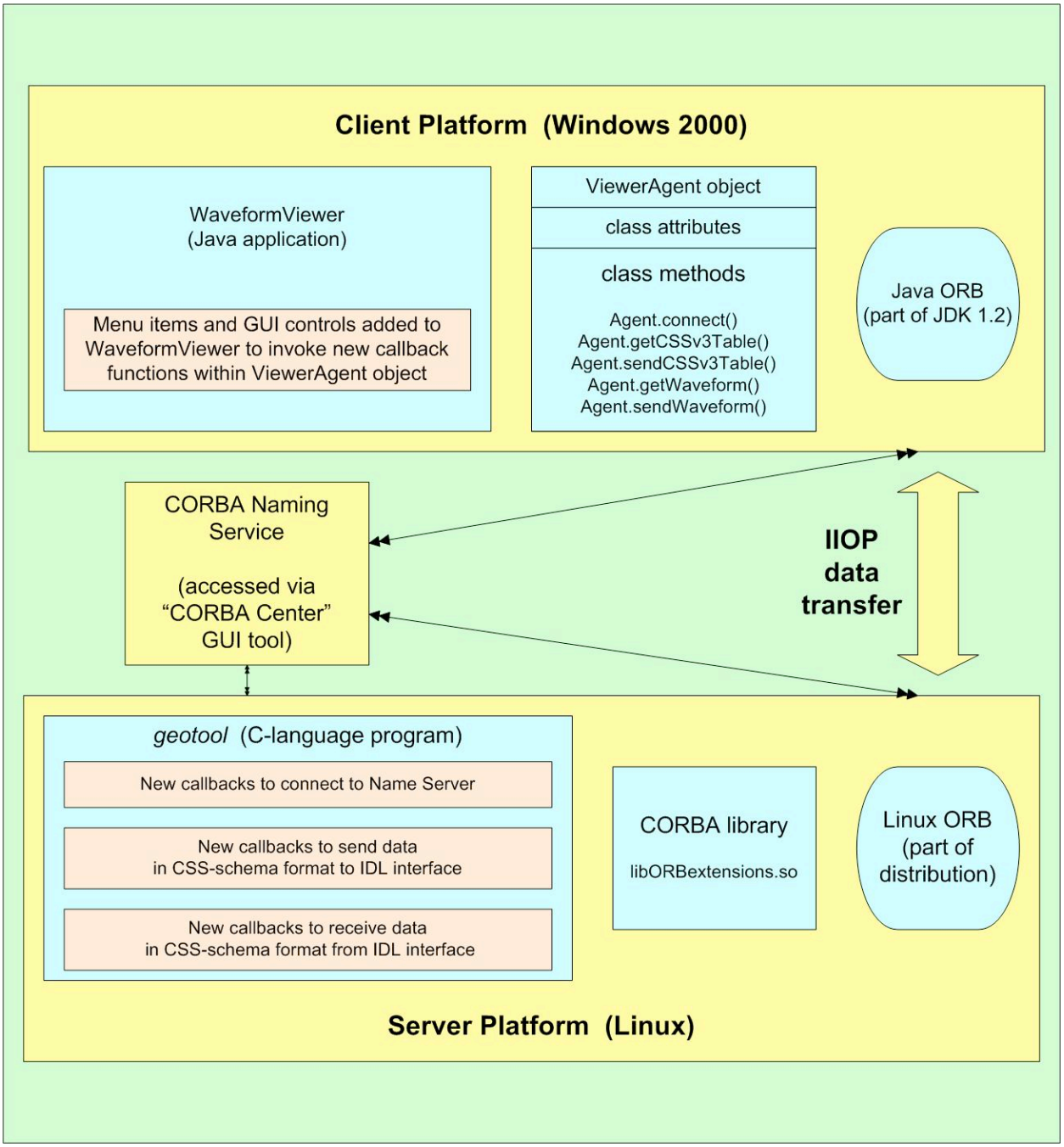

Figure 1 (From Rivers et al., 2002). Using CORBA to enable client/server data communications between the C-language program *geotool*, running as a data server under Linux, and the Java program *WaveformViewer*, running as a client application under Windows 2000. The data communication takes place between the ORBs on each platform via the IIOP, 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 server 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 IDL. 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).

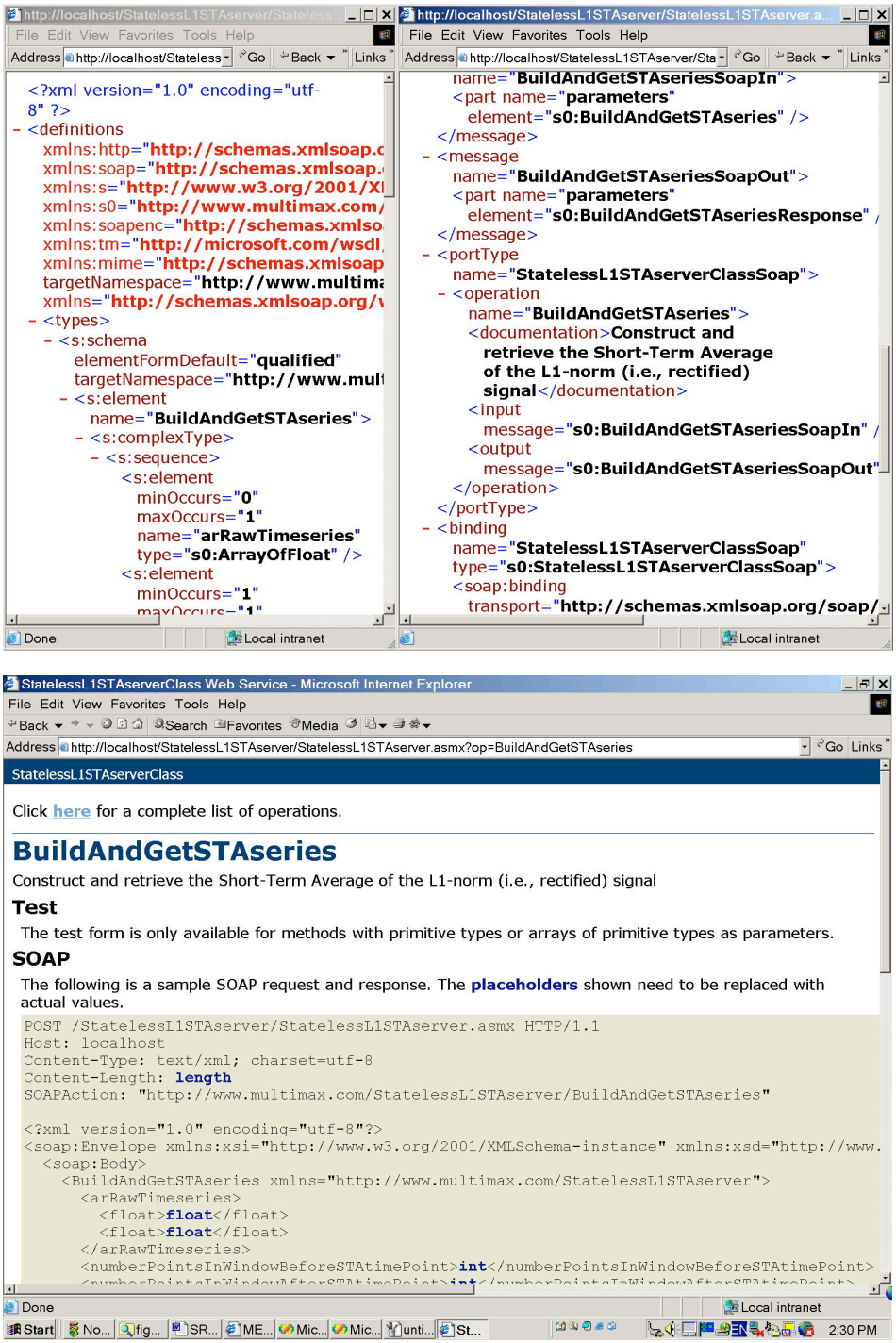

Figure 2 (After Rivers et al., 2003). (Top) Web services description language (WSDL) interface to a service application that accepts a waveform as an array of floating-point numbers and computes the short-term average of the rectified values. (Bottom) Sample code template (compiler generated) showing the format of a SOAP message that invokes this service. A client program such as *geotool* would dispatch a message in this format either through a manually written function or through machine-generated proxy code, and it will satisfy the WSDL interface shown in the top figure. The template also shows the format of the SOAP response message that is transmitting from the Web service to the client containing the XML-encoded processing results, and *geotool* would be modified to await that message (as the return value from a proxy function call) and then to parse it.

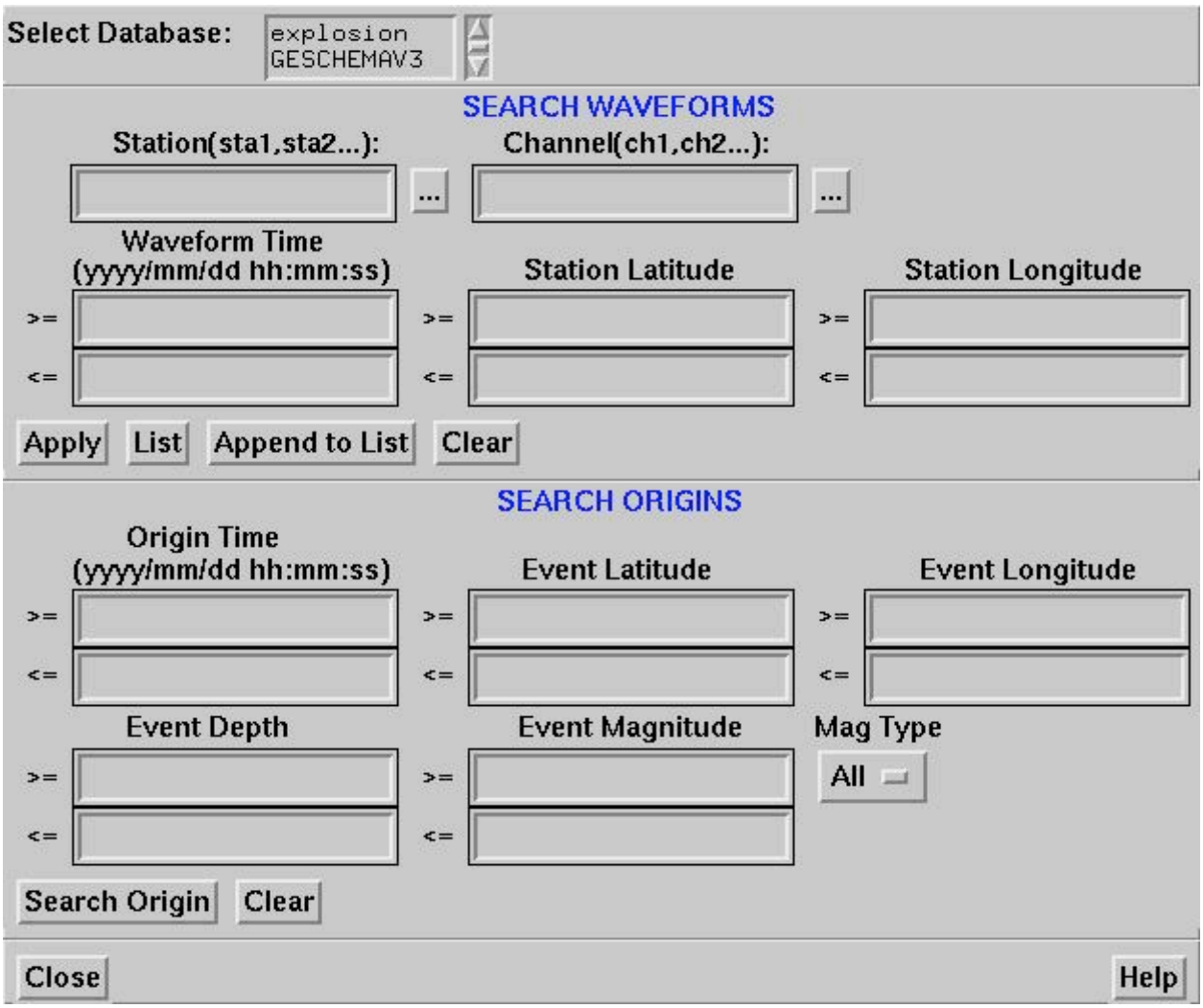

Figure 3. Among the bug fixes and enhancements that we have added to *geotool* during the last year is a graphical interface to the Oracle database that allows the user to query the database and retrieve waveforms of interest. This interface was added to *geotool* through conventional C-language Motif 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 wherein separate features are completely encapsulated and run within individual threads of execution. A distributed software design (even if all the components are local to a single workstation) will make programs such as *geotool* more flexible and safer to modify, but in its current state, *geotool* operates poorly as a client for distributed service applications. A more flexible system for analyzing potential seismic reference events would use a "thin client" for graphical interactions as a user interface and do much of the actual analysis by invoking self-contained services. Implementing such a design is becoming easier with the introduction of software tools that hide most of the low-level data communications within proxy code that is generated automatically by the development system. Under forthcoming platforms such as Windows 2006, the implementation of interprocess communications will be made considerably more nearly straightforward than it currently is, and we expect platforms such as J2EE to follow a similar trend. In these environments XML will be used as the data format for most applications, including database interactions and presentation graphics. The distinction between local and distributed applications will become indistinct, since the same data communications protocols will be used in both circumstances.