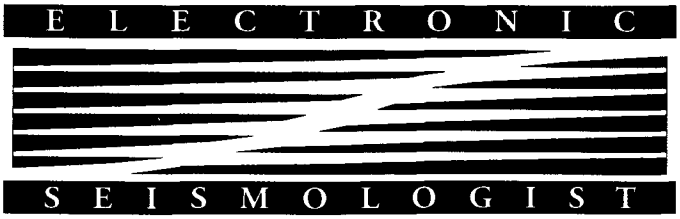


Steve Malone
E-mail: steve@geophys.washington.edu
Geophyics AK-50
University of Washington
Seattle, WA 98195
Telephone: (206) 685-3811
Fax: (206) 543-0489

FOREWORD

Anyone who reads any of the computer trade rags, or even the *Wall Street Journal*, has seen all sorts of blather regarding Java, the computer language, platform, virtual machine, do-all and be-all for everything. Depending on who the author is and from what industry, company, or persuasion, Java is either "an interesting little development" or the "second coming." Big deals about Java are that it is machine-independent, it can run on everything from super computers to toasters, it is "object-oriented" (OO—which is oh so much better than "structured"), and it contains and embraces inheritance, polymorphism, clever exception handling, and even garbage collection. If you care about these, don't know what these are, or just wonder if Bill Gates really does consider Java a "scary thing", then read on.

The Electronic Seismologist (ES) has talked a couple of young turk seismologists into giving us the inside seismological perspective on all this Java hoopla. Does it have promise for us? Are there useful things that one can do without "starting in column 7" (FORTRAN)? Why, or even should, us oldsters try to learn this new trick? Thus far there are few seismological applications (programs) written in Java. Many think of Java as only a way to do moving graphics on a Web page. Indeed, one of the first seismological uses of Java the ES is familiar with is the real-time seismogram display "applet" running at SCEC at URL http://www.scecdc.scec.org/seismocam. It's cute and works, but is it useful?

Other seismology applications in Java are starting to appear. Usually the driving force for these early programs is "I wanted to learn Java and here is an excuse to do so." This approach is inevitably that of students, but what better way to learn something than to try to do something useful as one learns? Do the applications end up being as efficient and robust as they would if written by experienced Java programmers? Probably not, but then that could be said about many useful programs written for seismological purposes. Just in the past couple of months the ES has played with several programs written in Java to get a sense of what might make them better or different than other programs. One, the *Quick Data Distribution System* (*QDDS*), is to be used for rapid distribution of earthquake parameter data from seismic networks to many clients. It takes advantage of Java's rich set of Internet support routines and its ability to run on any computer platform, but it is still in the early testing stages. Another application, called *WebWEED*, is designed to provide a friendly and easily modified and maintained user interface over the Internet into the IRIS Data Management System's data holdings. *WebWEED* should be available from IRIS for use over the Web by anyone by the time this article is published. Finally, the *TauP Toolkit* is a more traditional numerical application for calculating travel time and related information, but written in Java. Actually, an example of the relatively easy reusable nature of Java code is the use of the *TauP Toolkit* in the *WebWEED* application.

But enough of this. Let's hear from the aspiring Java programmers who can explain why we should all learn this language.

WebWEED and *TauP*: Java and Seismology

John Winchester
Geophysics Program, University of Washington
E-mail: winch@geophys.washington.edu

Philip Crotwell
Department of Geology, University of South Carolina
E-mail: crotwell@seis.sc.edu

Object-oriented (OO) programming and the Java platform are two technologies which show every sign of surviving their hype and becoming useful tools. In this article we will discuss the application of these technologies to seismic computing in the context of a generalized travel-time calculator and its use in a World Wide Web-based application for the selection of seismic data.

An Interesting Set of Problems to Solve

Seismologists are simultaneously blessed and cursed by the current abundance of seismic data. We can now do experiments using millions of seismograms with thousands of source-receiver geometries. The problem rapidly becomes how to mine this mother lode effectively. Nowhere is this problem more evident than at the Data Management Center (DMC) of the Incorporated Research Institutions for Seismology (IRIS). Making selections of these data easily available to the seismic community is one of the basic missions of the DMC.

One data selection tool currently available is *WEED* (*Windows Extracted from Event Data*). *WEED* is an X-Windows program which the user can download from the DMC along with a set of station and event files and use with its built in travel-time tables for the selection windows of data to be extracted. It generates data request messages which can then be e-mailed to the DMC or to any institution which

supports the request format. *WEED* has proven to be a valuable and useful tool. There are, however, several problems inherent in the distribution of stand-alone programs and databases. Some of the problems which affect *WEED* are:

- Distribution and maintenance: When new versions of the application are released, users must be notified and then must download the new versions. In addition, the station and event files at the DMC are updated daily, so the user must regularly download the updated files if he wishes to have access to the latest results.
- Inflexibility: Distribution of the program and its database locks the DMC into a fixed format for the cataloguing of stations and events.
- Platform dependence: The current implementation of *WEED* runs only under X-Windows on Sun computers.
- Fixed parameters: The travel-time tables are prescribed only for the most common phases and popular velocity models.

Enter the World Wide Web and Java

There now exists a platform which solves all of these problems. A Java applet running in a ubiquitously distributed Web browser such as Netscape or Internet Explorer suffers from none of the above problems and offers other strengths as well.

Java is a computation platform which originated at Sun Microsystems, Inc. Java is usually referred to as a platform rather than as a programming language. This is because Java has two basic parts, the development environment and the execution environment. The development environment is composed of the Java programming language, a fully object-oriented language and various development tools—for instance, Sun's Java compiler, *javac*, which compiles source code files (.java) into standardized binary files (.class).

The execution environment consists of the Java Virtual Machine (JVM), which runs the class files. JVM's exist for all major computer platforms. For browser-based Java applications (also known as applets) the JVM runs within the browser. Therefore, if you have recent versions of Netscape or Internet Explorer, you also have a JVM. It is this two-pronged approach which makes Java applications platform-independent. A class file compiled on any platform using *javac* will run on any JVM, whether the JVM is running on a Cray supercomputer, a Sun workstation, a Macintosh, or an IBM-compatible PC.

There are other aspects of the details of coding in Java which may be reason enough for using it as a computer language on its own merits. As one starts coding in Java, one soon comes to appreciate the cleanness of the language and the features of the system that make life as a developer easier.

For instance, Java does not have pointers as C does, so you no longer have to remember the difference between &, *, and ->. It does, however, retain all of the good things about pointers, such as dynamic memory allocation, while discarding the bad, such as moving a pointer off the end of an array (overflowing a dimensioned array for FORTRANers). In addition, you never need to worry about memory leaks (deallocating memory when finished), as the Java garbage collector takes care of this for you. Java also has exception handlers which provide an alternate means of dealing with error conditions (such as divide by zero) without using cryptic return codes. When used properly, exception handling can greatly aid in tracking down bugs. In addition, Java is more than just a programming language; it also includes a rich set of objects ready for our use. They range from graphical windows with buttons to hash tables to specialized IO streams. The fact that there are so many of these, and that they are a standard part of any Java installation, means that many common tasks are already done. Also, this set of standard objects, or API's, is growing larger and more sophisticated with every new version of Java.

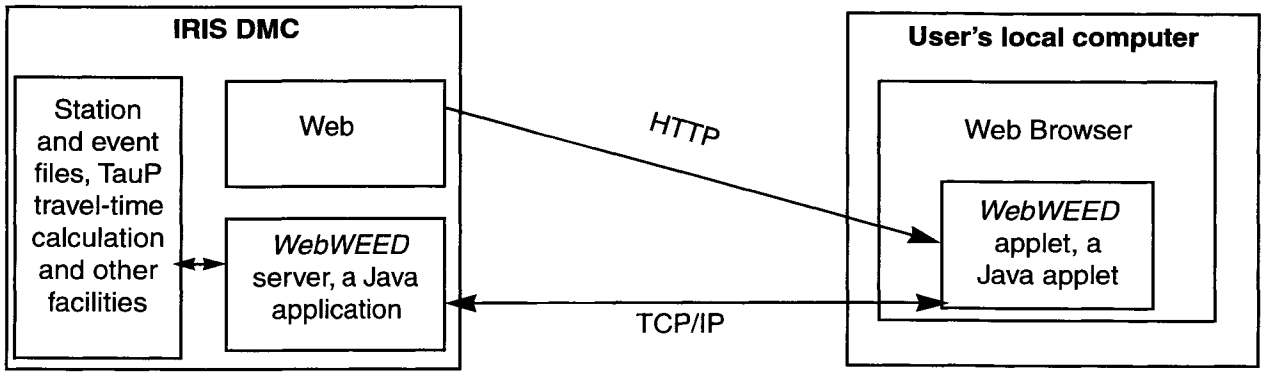
At a Higher Level

A basic attribute of Java is its embrace of object-oriented programming. A complete introduction to object-oriented analysis, design, and programming is far beyond the scope of this article, but it's possible to give the flavor of the OO approach. OO proramming enables a more natural and understandable way to write programs, although there is a learning curve involved in changing from structured programming techniques as used in the C programming language.

A quick analog from the real world may help convey some fundamentals. Objects have attributes and functionality. For example, a dog has a name and he has functionality. A basic functionality might be that he can bark. A dog also produces milk for its young, as does a cat. We can avoid duplication by creating a new object, mammal, that has the functionality, among other things, of producing milk. Dogs and cats then inherit from mammal and thus automatically have the functionality of producing milk. Inheritance provides for the reuse of common functionality among like objects.

Of course, the functionality is not always exactly the same. Mammals can create noise with their vocal cords, but dogs bark and cats meow. We can create the nature of a particular functionality within a particular mammal. Thus, one does not have to know the particular implementation of the functionality to know that the functionality exists. You can get a mammal to "speak", but whether you get a bark or a meow depends on which type of mammal you have. Polymorphism frees you from caring which particular subtype you have, since you need only know how to request "speaking" in general.

Java, by making these simple ideas workable within an elegant, clean, and feature-rich language, combined with its platform independence and rich set of API's, has given developers a powerful base from which to build ever more complicated and useful systems. The commercial world has embraced Java and will provide continued improvements that science in general and seismology in particular can leverage for its own needs. Best of all, Java is free!



▲ **Figure 1.** The physical architecture of *WebWEED*.

TauP and *WebWEED*

The *TauP Toolkit* is a Java implementation of the method of Buland and Chapman (1983) for the computation of seismic travel times and other related parameters. It extends the popular "IASPEI times" package to allow for arbitrary velocity models and a phase code parser to recognize and calculate travel times for virtually any type of seismic ray. Its modular design allows for the easy extraction of derivative information such as ray paths and for easy adaptation to other Java applications needing its functionality. A detailed description of this "toolkit" will appear as an article in a future issue of *SRL*. As a stand-alone application, the *TauP Toolkit* doesn't really address the distribution and database access solutions mentioned above. It does, however, benefit from the platform independence and object-oriented nature of Java.

WebWEED is a Web-based version of *WEED* which uses the Java/Web browser platform to solve all of the problems listed above. The physical architecture of *WebWEED* is shown in Figure 1. The user starts his Web browser and visits the *WebWEED* page at the IRIS DMC's Web site (http://www.iris.washington.edu). The *WebWEED* applet is automatically downloaded and runs within the JVM in the user's Web browser. The applet opens a TCP/IP socket back to the IRIS DMC and becomes a client of the *WebWEED* server, a Java application running at the DMC. The applet provides forms for the collection of various user information and data selection criteria and transmits this information back to the *WebWEED* server at the DMC. The server provides station and event information and calculates travel times of seismic phases using the *TauP Toolkit*. The server will formulate a data request using this information and can then e-mail the request to the user and, if desired, to the IRIS DMC or any other institution which honors the request format.

It may be instructive for understanding object-oriented design to follow the path used in developing *WebWEED*. The design of robust and flexible software systems has been greatly eased by the use of design patterns, a technique developed by a group of seasoned software designers usually referred to as the "Gang of Four" (Gamma *et al.*, 1994). They studied patterns common to successful software architectures, classified them according to their strengths and their requirements, and then catalogued these patterns. A software designer can compare the requirements of his project with these patterns and choose an architecture which has been proven to provide a reliable solution for similar projects.

WebWEED is easily divided into the user interface/client and the server. The client applet must have components which allow the user to log in, select station and event files, set geographical selection criteria, set window definitions for the desired seismic phases, and initiate a data selection request. The server must have components to fetch user profiles, fetch lists of station and event files, use the user-supplied criteria to winnow the stations and events, determine the travel times of the seismic phases specified in the window definitions, create summary files of data which fit all of the criteria, and create and e-mail the actual data requests. There must also be communication modules which control the transmission and reception of messages between the client and the server. Note that the components of the client do not need to know each other. The log-in process, the selection of files, and the determination of the user's other selection criteria may all proceed independently of each other. They all, however, share the need to communicate with the components of the server. A similar state of affairs exists on the server side.

This situation fits a design pattern called the "Mediator-Colleague" pattern. In *WebWEED*, the client and the server each have mediator components. The mediator modules own and manage the communication pipeline between the client and the server. The various modules on each end can register dynamically as colleagues with their local mediator and request that the mediator send a message to a remote colleague. For example, when the user selects a new directory of event files on the applet, the client-side file browser colleague tells its mediator to send a request message to the server-side file server colleague. The server-side mediator receives the message and passes it on to the file server, which then retrieves the desired directory listing and sends it back in a return message.

One advantage of this design pattern is that it is very easy to add functionality to the system, even after the system is operational. As an example, we decided to add the log-in

process to *WebWEED* after the application was fully operational. This required writing a log-in dialog for the client side and a user profile fetcher for the server side, and then plugging these modules into their respective mediators. This style of design is not new to OO programming, but the high degree of modularity inherent in OO programming under Java makes using these design patterns much easier.

There are many benefits to using the design pattern approach, Java, and a client/server architecture. First, the distribution problem is solved. There is only one *WebWEED* applet. It resides at the DMC and automatically downloads to the user's browser on demand. Upgrades and maintenance to *WebWEED* are thus completely transparent to the user. Secondly, all database access is encapsulated on the server side of the architecture. There is no longer any need to distribute the station and event files. Their format can change to adapt to future needs. Thirdly, the guts of the selection routine are the *TauP Toolkit*, written by someone else but with a well defined interface for *WebWEED* use. An unlimited set of velocity models and seismic phases could be used for defining selection windows. Finally, *WebWEED* can evolve quite freely. For example, it could be integrated with other applications, such as a data viewer for examining waveforms.

The reader is invited to try both of these applications, but one should be warned that, as with any web-based software, using a recent release of either the Web browser (*WebWEED*) or the Java Virtual Machine (*TauP*) may be necessary.

References:

Buland, R. and C.H. Chapman (1983). The computation of seismic travel times, *Bull. Seism. Soc. Am.* **73**, 1271-1302.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1994). *Design Patterns: Elements of Reusable Object-oriented Software*, 395 pp., Addison-Wesley, Reading.

Relevant Web URL's:

The IRIS Data Management Center: http://www.iris.washington.edu

JavaSoft: http://www.javasoft.com

The Java Tutorial: http://java.sun.com/docs/books/tutorial/index.html

Java Developer Connection: http://developer.javasoft.com/developer

TauP Toolkit software at University of South Carolina: http://www.geol.sc.edu/seis/software

---

SRL encourages guest columnists to contribute to the "Electronic Seismologist." Please contact Steve Malone with your ideas. His e-mail address is steve@geophys.washington.edu.