A spectral estimation toolkit for Java applications

Vijay Shah\textsuperscript{a}, Nicholas Younan\textsuperscript{a,}\\textsuperscript{*}, Torey Alford\textsuperscript{b},
Anthony Skjellum\textsuperscript{b}

\textsuperscript{a}Department of Electrical and Computer Engineering, Mississippi State University, Mississippi State, MS 39762, USA
\textsuperscript{b}MPI Software Technology, 110 12th Street North, Suite D103, Birmingham, AL 35203, USA

Received 16 January 2004; received in revised form 28 April 2004; accepted 20 May 2004
Available online 20 July 2004

Abstract

This paper examines the capability, performance, and relevance of a high-performance advanced signal processing toolkit in Java, a programming language for Web-based applications. To demonstrate the simplicity, ease, and application use of the toolkit, a spectral estimation applet has been developed in the Java environment using advanced Internet technologies such as Remote Method Invocation (RMI). This application provides an interactive and visual approach in understanding theoretical concepts of advanced signal processing methods and shows the need to create more application applets to better understand additional concepts in signal and image processing. Furthermore, a toolkit with limited functionality and different framework has been developed for embedded and handheld devices such as cellular phones and palm pilots. This toolkit is also shown to be useful in developing applications MIDlets on those devices.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Java; Algorithms; VSIPL; Spectral estimation

1. Introduction

The introduction of Java by Sun Microsystems has brought a revolution into the computing world. Java is the most effective programming language for web-based applications since Java technology is not limited to workstations and servers, but it also...
targets a wide range of platforms such as pagers, hand-held devices, cellular phones, screen phones, digital set-top boxes, and car navigation systems as well.

Standard signal processing libraries, such as Vector signal and image processing library (VSIPL) in C [15], Signal Processing in C++ (SPUC) [9], and the IEEE DSP program (McMaster) in Fortran [4] are available. With these libraries, why should a toolkit be developed in Java? Although Java is considered to be relatively slow for computational type applications, it has several advantages such as simplicity, object-oriented programming, portability, robustness, security, exception handling capability, language for Internet applications, and simplifying the multiprocessing tasks of a signal processing system using Remote Method Invocation (RMI). **Fig. 1** shows a real enterprise scenario, which requires distributed applications that could be easily developed using Java [16].

Current available packages in Java are customized, distinct, and do not allow an easy creation of specific new applications for signal processing [11]. Moreover the computation for these applications was performed on the client’s machine. Having a standard Java toolkit simplifies the problems of developing new applications as well as improving the current applications. In this paper, an advanced signal processing toolkit in the Java environment has been developed using the advanced Internet technologies such as Remote Method Invocation (RMI). This makes it possible to perform computation on the server side. Additionally, this toolkit provides an interactive and visual approach in understanding the various theoretical concepts of spectral estimation and shows the need to create more application applets to better understand the various concepts of signal and image processing. A performance comparison of the Java package with other standard packages using other high-level languages is also made.
Section 2, Java, discusses the Java technology used for developing applications. Section 3, inAspect library, discusses the programming model and data manipulation for the library. Section 4, Spectral estimation application, discusses the application package developed using the inAspect Library, the general approach used, applet design, and screen shots of the applet. Section 5, Spectral estimation application for embedded devices, discusses the applications developed for the embedded devices such as cell-phones, PDAs, and Win CE devices including some screen shots. Section 6, Performance comparison, discusses the performance of the application package with different options with a readily available signal processing toolkit like Matlab.

2. Java

2.1. Java native interface

The Java Native Interface (JNI) provides a standard interface that allows a JAVA code to interoperate with the native code, written using C, C++, and/or assembly [10]. JNI acts as the “glue” between native and Java applications. JNI saves the time of rewriting high-performance applications that already exist in the native language. Fig. 2 shows the JNI layout.

The native implementations are compiled into a dynamic link library (known as shared objects (.so) in Solaris and dynamic link libraries (.dll) in Win32). The operating system loads and links this library into the process that is running the Java Virtual Machine.

2.2. Remote method invocation

Remote Method Invocation facilitates writing an application to distribute computing across the networking environment. RMI takes a step further in object-oriented design by selecting the appropriate machine for performing a specific task [17]. The RMI architecture consists of three layers: the stubs-skeleton layer, the remote reference layer, and the transport layer. Fig. 3 shows the relationships among these layers [13].

The stubs/skeletons layer maintains a connection between the objects of client–servers. The remote reference layer (RRL) provides the data stream to stubs and skeletons, and it deals with the transport layer. It thus manages the persistence and communication between the client–server. The transport layer is the layer that creates and maintains the actual connection between client and server, sending information over wire.
3. inAspect library

3.1. inAspect’s programming model

inAspect has been designed to provide a portable, high-performance library for signal processing in Java. Considering the fact that Java is computationally slow, inAspect uses native libraries on supported platforms. Currently, it provides the interface to the native library developed by VSIPL and VSI/PRO [14,15]. The programming model of inAspect using JNI is shown in Fig. 4, which also shows the utility and importance of the VSIPL-VSI/PRO native library in the Aspect package. Fig. 4 also represents the principle design to “connect” any developed application using the inAspect to the platform. In order to support the platform not supporting JNI, inAspect would be performing signal processing in Java. The pure Java version of inAspect is in the development phase.

3.2. Library data manipulation

Sample, Block, and View are the base classes for storage and operation of the user data. “Sample” is the term used in signal processing for a datum. Sample and its subclass provide the support for storing and retrieving the real and complex values of different primitive data types, including double, float, integer, and boolean. VSIPL introduced the concept of “Block” for allocating continuous memory for the storage of data. A similar concept to Block is the basic unit of the inAspect package. The data type specific classes ComplexBlock and RealBlock extend the Block class. These classes are extended according to the available primitive data types, double and float. Currently Java does not
An example block of size = 25

| 0 | 1 | 2 | 3 | 4 | 5 | ... | ... | ... | 20 | 21 | 22 | 23 | 24 |

Let’s form a Vector View with

- offset = 3 (4th element of the block)
- length = 6 (number of elements it can access from a given block)
- stride = 3 (space between consecutive element in vector view)

Resultant view is: 3 6 9 12 15 18

Fig. 5. Example for creating a specific view from a block.

3.3. Operation queuing

In light of the obstacles discovered from the benchmarks in the previous section, it became imperative that ways be found to streamline operations between Java and native C libraries. One solution was the idea of creating reusable queues into which numerical operations (i.e. signal processing and linear algebra) would be placed. Once created, these queued operations could be performed repeatedly on streams of data. Fig. 6 graphically displays this concept. This idea is consistent with the deferred evaluation concept of the original VSIPL [15] specification.

Fig. 6. Basic queue concept in action.
Listing 1: Queue Benchmarking Tests

```java
public void test1 () {
    for (int i=0; i < numIterations; i++)
        block = javaOperation (java);
}

public void test2 () {
    for (int i=0; i < numIterations; i++)
        block = jniOperation (java);
}

public void test3 () {
    jniQueueSetup (block);
    for (int i=0; i < numIterations; i++)
        jniQueueProcess ();
    block = jniQueueRetrieve ();
}

public void test4 () {
    jniQueueSetup (block);
    jniQueueProcess (numIterations);
    block = jniQueueRetrieve ();
}
```

A simple benchmark was designed to test this queuing idea. Listing 1 contains four algorithms which are functionally identical. Each performs an element by element sine square operation on an array of double values. The `test1` function is a pure Java implementation of the algorithm while the native `test2` is called for each iteration. `test3` and `test4` both utilize a basic queuing design. Each requires two additional JNI calls: an initial call to pass the data block for binding, and a final call to retrieve the data. The difference is that `test3` calls the queue processing handler repeatedly for each iteration, whereas `test4` passes the iteration value for the native iteration. The results of these tests indicate that native queuing provides substantial performance over pure Java implementation. Fig. 7 shows the results of these tests.

4. Spectral estimation application

inAspect enables Java developers and Digital Signal Processor developers with a high-performance package capable of real world applications. One topic of interest in signal processing applications is spectral analysis, a process that involves the estimation of the spectral content, i.e. the distribution of the power over frequency, of a time series data from a finite set of measurements. A spectral estimation toolkit has been developed to demonstrate the usefulness of such signal processing packages. The developed toolkit can
be useful in developing more complex applications to further understand various concepts of spectral estimation.

Traditional methods, based solely on Fourier transforms, are mainly used for relating the time domain representation of a signal to its frequency domain. In addition, since most signals have noisy or random components, statistics play a major role in characterizing these signals. Accordingly, robust and reliable signal processing algorithms are needed to improve the overall performance, such as better frequency or spectral resolution and higher signal detection ability. The Spectral Estimation package layout is shown in Fig. 8. The estimation algorithms, identified for this task, are grouped as Classical, Parametric, and Statistical estimation [8,12].
4.1. General functional block diagram

The JNI introduces the overhead of transferring data from the native side to Java. This overhead may be large enough to reduce the high performance obtained using the native approach. With this issue in mind, inAspect uses the concept of queuing. All the operations to be performed on the data are queued in the queue object, eliminating the need for switching between C and Java for each operation. Thus, all operations are performed using the single native call [1]. Fig. 9 illustrates the concept of queuing, used in implementing the Spectral Estimation Package.

Classical estimation relies totally on estimating the power spectral density (PSD) of time series data via the discrete Fourier transform (DFT) \([8,12]\). A fast way to implement the DFT is the well known fast Fourier transform (FFT). This transform relationship is considered as a nonparametric description of the second order moment of the time series data. The most commonly used classical spectral estimators are the Periodogram, Correlogram, and Average Periodogram. Fig. 10 shows the functional block approach used for estimating the PSD using classical estimation methods. It also demonstrates the application of the queuing concept of the inAspect package mentioned previously in Fig. 9.

4.2. Spectral estimation applet design

The class inaspectSpectrumAnalysis extends the JApplet class and uses spectImp to call library functions that are bundled in the SpectralEstimation package. Class inaspectSpectrumAnalysis is the main applet class and is also the client application. Class inaspectInputPanel provides functions to obtain inputs from the user. Class inaspectSignalPanel displays
the input and output results. Class armaParameters provides a dialog box for entering various parameters required by the AR, MA, ARMA and Statistical Methods. Fig. 11 shows the class relationships for the main applet.
The RMI structure for the applet is shown in Fig. 12. Class spectEst provides the interface for calculating the power spectral density, depending upon the selected method. Class spectEstImp provides the implementation of the interface defined in spectEst. Classes spectEstImpStub and spectEstImpSkel help in maintaining the connection between the objects of client applet and server. Class spectServ is the server application and helps binding the server name to the object spectEstImp.

4.3. Screen shot of the JAVA applet

The main JAVA applet that is developed using Java 2 Standard Edition v 1.4.1 (J2SE) is divided into three modules; user input, computational, and display modules. A brief description of each module follows:

User-Input Module: The user-input module is divided into menus and it lets the user select different methods, signal type, and window type from the Method menu, Signal menu, and Window menu, respectively. The user can draw or clear the spectrum plot from the Spectrum menu. Furthermore, the user is able to input the various parameters of a signal, such as the sampling frequency, number
of samples, FFT length, window size, and overlap length for the Average Periodogram. A separate dialog box is provided to enter the model orders for the AR, MA, and ARMA methods. These parameters are then passed to the server (spectServ) to obtain the corresponding PSD. The results are then returned to the client.

**Computational Module:** This module is used to compute an estimate of the power spectral density (PSD) of a time series data according to a certain set of parameters as defined in the UserInput Module. The computational module resides on the server side, hence all the computations are performed on the server side.

**Display Module:** This module displays the input signal and outputs the corresponding PSD in dB. Fig. 13 shows the actual screen-shots of the display module. The output results are obtained for a set of 1500 data samples that consists of a 50 Hz sinusoid with a unit amplitude in additive white noise of unit variance (SNR = −3 dB), sampled at 800 Hz, using a rectangular window, and taking a 1024 point FFT.

### 5. Spectral estimation application for embedded devices

It is likely that embedded devices (potentially with modest computational abilities) will have a need to more fully exploit speech recognition technology as a means of providing an additional input modality—as such, spectral estimation and other forms of signal processing are likely to be of use. Hence an inAspect package with different framework and limited functionality has been developed for those devices. Using this toolkit a midlet has been developed to perform spectral estimation.
Portability is a primary functional goal of the inAspect package. With portability being one of the primary functional aspects of the Java language, designing a portable package might seem trivial. However, when developing a package that depends on natively implemented algorithms, portability becomes intricate. Furthermore, complexity increases as Java has been targeted towards more platforms with limited memory and processing constraints.

In order to provide Java on limited memory devices, Sun Microsystems redesigned Java and produced the scaled down version named Java 2 Micro Edition (J2ME) [6]. The J2ME standard specifies a system of different hardware “configurations” that describe the minimum qualifications a Java compatible embedded device must meet. The two current “configurations” are the CLDC (Connected Limited Device Configuration) and the CDC (Connected Device Configuration). The CLDC is designed for devices with memory limited to 512 KB and consists of a restricted version of the standard Java interface. One restriction is the absence of a JNI, a restriction that prompted the development of the pure-Java inAspect implementation.

5.1. MathFP

The J2ME specification also does not permit the usage of any type of floating point number support. Fixed-point arithmetic is the solution to this problem. Fixed-point arithmetic utilizes integer values at the bit level to represent real values. jScience Technologies has developed a fixed-point library package for J2ME called MathFP [7]. In addition to the fixed-point representation, this package provides functionality that the core J2ME library lacks, such as basic arithmetic and trigonometric functionality.

The MathFP package encodes decimal values by placing the integer part in the upper 20 bits (40 bits for 64-bit precision devices) and the fractional part in the lower 12 bits (24 bits for 64-bit precision). This differs from the IEEE-754 standard where one bit is used for the sign bit, eight bits are used for the exponent (11 for 64-bit precision), and 23 bits are used for the significand (52 for 64-bit precision).

The IEEE-754 standard allows precision up to $\pm 10^{-38.53}$ for single precision values and $\pm 10^{-200.3}$ to $10^{308.3}$ for double precision values [5]. The MathFP library, on the other hand, only allows precision up to three decimal places for 32-bit integers and seven decimal places for 64-bit integers.

While the library provides basic fixed-point functionality, the lack of precision caused overflow errors in several of the algorithms in the inAspect package. As a result, these algorithms have been rewritten using methods such as the CORDIC (COordinate Rotation Digital Computer) algorithm.

5.2. CORDIC algorithms

The CORDIC algorithm, first described by Jack E. Volder, can be useful in calculating trigonometric functions, polar to rectangular conversion, rectangular to polar conversion, and vector magnitude as well [2,3]. It is an iterative method of performing vector rotation by a required angles using shift and add operations. It is a useful technology for computing real values in integer-only systems, and is widely exploited in field programmable gate arrays (FPGAs) and elsewhere, but not widely in general purpose processing.
Given an \([x, y]\) vector, the algorithm produces a new vector, \([x', y']\), where 
\[
x' = x \cos \theta - y \sin \theta
\]
and 
\[
y' = y \cos \theta + x \sin \theta
\]
, which is equivalent to rotating the vector \([x, y]\) by some angle \(\theta\). The above equations can be rearranged to the form 
\[
x' = \cos \theta (x - y \tan \theta)
\]
and 
\[
y' = \cos \theta (y + x \tan \theta)
\]
. By limiting \(\tan \theta\) to \(\pm 2^{-i}\), tangent multiplication is replaced by shifts \([2]\). This iterative process requires a certain number of precision bits to perform smaller rotations and, at each iteration, a decision to rotate a vector in a clockwise or counterclockwise direction is made. This produces equations of the form 
\[
x_{i+1} = K_i[y_i - x_i d_i 2^{-i}]
\]
and 
\[
y_{i+1} = K_i[y_i + x_i d_i 2^{-i}]
\]
, where 
\[
K_i = \frac{1}{\sqrt{1 + 2^{-2i}}}
\]
and 
\[
d_i = \pm 1.
\]
For each iteration, it is also required to accumulate the angle and hence a third equation is added to the CORDIC algorithm. That is 
\[
z_{i+1} = z_i - d_i \tan^{-1}(2^{-i})
\]
. By removing the scale constant from the iterative equation and storing the value of \(\tan^{-1}(2^{-i})\) in a small lookup table, this set of equations involves only shift and add operations. Note that the CORDIC rotator operates only in “vectoring” and “rotation” modes. When operating in the vector mode, the input vector is rotated by some angle such that the output vector is aligned to the \(x\)-axis. This mode is used to directly find the arctangent, vector magnitude, and polar transformation from the Cartesian coordinates.

In the rotation mode, the CORDIC iterations rotate a given vector by an arbitrary angle. Using this mode, both polar to rectangular transforms can be performed and the sine and cosine of the required angle can be found. For example, the rotation mode is used by the CORDIC FFT algorithm to shift a sample by a specified angle.

While the MathFP library allows the creation of and operation on fixed-point values, the packaged operations often cause overflow or underflow with certain signal processing algorithms. However, modifying these signal processing algorithms to exploit the CORDIC functionality produces correct results without these errors. The combination of CORDIC algorithms and MathFP is consequently a significant step forward in delivering an effective, though limited-precision, inAspect for resource-constrained systems.

### 5.3. Screen shots

#### 5.3.1. MIDlets on handled devices

The JAVA 2MicroEdition Wireless toolkit was used to develop MIDlet for the classical spectral estimation. MIDlet was tested on the Motorola\(i85\) emulator and also on the emulator of other 32-bit devices like RIM Java Handheld, palm OS, and Default Color Phone of J2MEWTK.

Configuration and Input Parameters are the two sub-modules of the user-input module. The Configuration sub-module consists of a “Method choice group”. It is used to select one from the three classical methods. The “Signal choice group” allows the selection of a pure sinusoid and a noisy sinusoid with uniform or normally distributed noise, and the “Window choice group” allows the user to select a window type from a menu that includes Rectangular, Hanning, Hamming, Bartlett, and Blackman windows. The Input Parameters sub-module allows the user to input the frequency and amplitude of a sinusoid, sampling frequency, number of samples, FFT length, and window size. Figs. 14 and 15 show the screen shots of the input for Motorola\(i85\), RMI handheld, and Palm OS Emulator devices respectively using the same code.
The output power spectral density estimate on these devices for the noisy sinusoid signal of 50 Hz frequency, amplitude of 5 units, and additive white noise of unit variance (SNR = 11 dB), and sampled at 1000 Hz is illustrated in Fig. 16. Note that a rectangular window of length 128 is applied to the 500 samples of the input signal. Note also that for a 32-bit device and using a precision of 12 bits, a power spectral density in the range of 114 dB to −72 dB can be estimated.

5.3.2. Real-time application

In order to demonstrate the performance of inAspect handling real-time signals, a small application is developed to visually display incoming sound data from a microphone on a Windows CE device. This device uses the Java Personal Edition that is superset of J2ME. Fig. 17 shows an HP Jornada device utilizing the inAspect package to perform real-time analysis on an incoming signal through the built-in microphone.


6. Performance comparison

The inAspect package provides two types of queuing: persistent and non-persistent. If the queue is declared persistent, all the required operations are queued and they remain in the queue until the queue is destroyed or a specific command is given to remove it. Queuing is processed using a specific function. The same set of operations can be performed on new data by processing the same queue. In the non-persistent queue, each operation is processed immediately and a specific operation is removed from the queue. In order to perform the same set of operations on new data, all the operations are required to be enqueued. The performance of the inAspect package is also compared with popularly available signal processing toolkits in Matlab (version 6.1).

Figs. 18 and 19 show the timing of two Classical Estimation methods, Periodogram and Correlogram, under Windows 2000 Professional (Hardware: Intel Pentium III 500 MHz,
256 MB RAM), SunOS 5.8 (Hardware: Sun Ultra 5/10 UPA/PCI [UltraSPAR, C-IIi 333 MHz], System clock frequency: 83 MHz, Memory size: 256 MB) operating systems respectively and Sun Microsystems’s J2SE 1.4.1 SDK. A comparison is also made between the algorithms implemented in pure Java, using the non-persistent queue, Matlab, and persistent queue of the inAspect package. In order to thoroughly test the performance of each implementation, random data is generated, a 1024 point FFT is taken, and an average time for 1250 cycles is calculated.

These figures clearly show the advantage of using the inAspect package with the persistent queue and native libraries (if available for that platform). It can be seen that the implementation using non-persistent queuing is slower due to the extra overhead involved in transferring data from the C layer to the Java layer for each operation. For the
Windows platform, the Matlab and Java implementation using non-persistent queuing take nearly equal times to perform signal processing operations. For the Unix platform, the Matlab version outperformed the Java implementation using non-persistent queuing. Furthermore, the pure Java version is even slower due to the overhead and extra checking which Java performs during execution. For this reason, the design of such algorithms is essential in order to maintain an acceptable level of performance.

7. Conclusion

Various independent and customized signal processing packages in Java are available, but developing a new application from these packages is limited. Hence, the development of a signal processing toolkit provides more usability and better performance. inAspect™ provides a high-performance signal and image processing toolkit in Java. This package outperforms Matlab operations when functions are enqueued. This performance comparison shows the advantage of using the JNI compared to pure Java. Furthermore, the creation of a spectrum analysis applet using the inAspect package shows the ease of using this toolkit for the development of various signal processing applications.

8. Future work

The extended Markup Language (XML) is a standard that is used to process data-related matters such as data collection, data transmission and exchange across different hardware, operating systems and applications, and back-end operation of computing with the characteristics of “extensibility” and “self-description” and little human intervention. Developing XML standards for signal processing applications would simplify the storage, retrieval, management, and analysis of information in any form, audio, images, video, and text data across the internet.

With the release of .NET by Microsoft, that is useful for developing applications using Common Runtime Language (CRL), it would be interesting to compare the performance of DSP web applications developed using Java and .Net.

It would be of interest to develop standard fixed-point signal processing libraries in higher-level languages for embedded platforms, using the maximum available bits for a specific device, benchmarking various algorithms on embedded devices, and developing a better GUI for the MIDlet.

Other research goals will be the incorporation of multi-threading in the spectrum analysis applet to make it real time and improve its performance, the development of additional application applets to understand the concept of signal and image processing, and the creation of an online tool analogous to Matlab for performing various signal and image processing functions.

Acknowledgements

The authors would like to acknowledge the National Science Foundation for their funding under NSF Phase II SBIR Grant 0078563.
References