The following paper was originally published in the
Proceedings of the 4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS)
Santa Fe, New Mexico, April 27-30, 1998

The Design and Performance of MedJava

Prashant Jain, Seth Widoff, and Douglas C. Schmidt
Washington University

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org/

The Design and Performance of MedJava

A Distributed Electronic Medical Imaging System

Developed with Java Applets and Web Tools

Prashant Jain, Seth Widoff, and Douglas C. Schmidt
{pjain,sbw1,schmidt}@cs.wustl.edu
Department of Computer Science
Washington University
St. Louis, MO 63130, (314) 935-4215[*]

This paper appeared in the 4^{th} USENIX Conference on Object-Oriented Technologies and Systems (COOTS), Sante Fe, New Mexico, April 1998.

Abstract

The Java programming language has gained substantial popularity in the past two years. Java's networking features, along with the growing number of Web browsers that execute Java applets, facilitate Internet programming. Despite the popularity of Java, however, there are many concerns about its efficiency. In particular, networking and computation performance are key concerns when considering the use of Java to develop performance-sensitive distributed applications.

This paper makes three contributions to the study of Java for performance-sensitive distributed applications. First, we describe an architecture using Java and the Web to develop MedJava, which is a distributed electronic medical imaging system with stringent networking and computation requirements. Second, we present benchmarks of MedJava image processing and compare the results to the performance of xv, which is an equivalent image processing application written in C. Finally, we present performance benchmarks using Java as a transport interface to exchange large medical images over high-speed ATM networks.

For computationally intensive algorithms, such as image filters, hand-optimized Java code, coupled with use of a JIT compiler, can sometimes compensate for the lack of compile-time optimization and yield performance commensurate with identical compiled C code. With rigorous compile-time optimizations employed, C compilers still tend to generate more efficient code. However, with the advent of highly optimizing Java compilers, it should be feasible to use Java for the performance-sensitive distributed applications where C and C++ are currently used.

[*]This research is supported in part by a grant from Siemens Medical Engineering, Erlangen, Germany.

1 Introduction

Medical imaging plays a key role in the development of a regulatory review process for radiologists and physicians [1]. The demand for electronic medical imaging systems (EMISs) that allow visualization and processing of medical images has increased significantly [2]. The advent of modalities, such as angiography, CT, MRI, nuclear medicine, and ultrasound, that acquire data digitally and the ability to digitize medical images from film has heightened the demand for EMISs.

The growing demand for EMISs has been coupled with a need to access medical images and other diagnostic information remotely across networks [3]. Connecting radiologists electronically with patients increases the availability of health care. In addition, it can facilitate the delivery of remote diagnostics and remote surgery [4].

As a result of these forces, there is also increasing demand for *distributed* EMISs. These systems supply health care providers with the capability to access medical images and related clinical studies across a network in order to analyze and diagnose patient records and exams. The need for distributed EMISs is also driven by economic factors. As independent health hospitals consolidate into integrated health care delivery systems [2], they will require distributed computer systems to unify their multiple and distinct image repositories.

Figure 1 shows the network topology of a distributed EMIS. In this environment, medical images are captured by modalities and transferred to appropriate Image Stores. Radiologists and physicians can then download these images to diagnostic workstations for viewing, image processing, and diagnosis. High-speed networks, such as ATM or Fast Ethernet, allow the

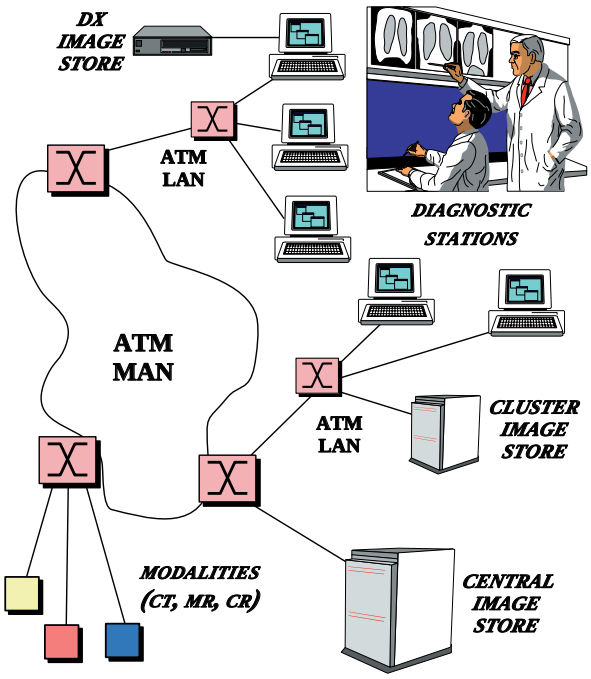

Figure 1: Topology of a Distributed EMIS

transfer of images efficiently, reliably, and economically.

Image processing is a set of computational techniques for enhancing and analyzing images. Image processing techniques apply algorithms, called *image filters*, to manipulate images. For example, radiologists may need to sharpen an image to properly diagnose a tumor. Similarly, to identify a kidney stone, a radiologists may need to zoom into an image while maintaining high resolution. Thus, an EMIS must provide powerful image processing capabilities, as well as efficient distributed image retrieval and storage mechanisms.

This paper describes the design and performance of *MedJava*, a distributed EMIS developed using the Java environment and the Web. The paper examines the feasibility of using Java to develop large-scale distributed medical imaging applications with demanding performance requirements for networking speed and image processing speed.

To evaluate Java's image processing performance, we conducted extensive benchmarking of MedJava and compared the results to the performance of `xv`, an equivalent image processing application written in C. To evaluate the performance of Java as a transport interface for exchanging large images over high-speed networks, we performed a series of network benchmarking tests over at 155 Mbps ATM switch and compared the results to the performance of C/C++ as a transport interface.

Our empirical measurements reveal that an imaging system implemented in C/C++ always out-performs an imaging system implemented using interpreted Java by 30 to 100 times. However, the performance of Java code using a "just-in-time" (JIT) compiler is ∼1.5 to 5 times slower than the performance of compiled C/C++ code. Likewise, using Java as the transport interface performs 2% to 50% slower than using C/C++ as the transport interface. However, for sender buffer size close to the network MTU size, the performance of using Java as the transport interface was only 9% slower than the performance of using C/C++ as the transport interface. Therefore, we conclude that it is becoming feasible to use Java to develop large-scale distributed EMISs. Java is particularly relevant for wide-area environments, such as teleradiology, where conventional EMIS capabilities are too costly or unwieldy with existing development tools.

The remainder of this paper is organized as follows: Section 2 describes the object-oriented (OO) design and features of MedJava; Section 3 compares the performance of MedJava with an an equivalent image processing application written in C and compares the performance of a Java transport interface with the performance of a C/C++ transport interface; Section 4 describes related work; and Section 5 presents concluding remarks.

2 Design of the MedJava Framework

2.1 Problem: Resolving Distributed EMIS Development Forces

A distributed electronic medical imaging system (EMIS) must meet the following requirements:

• **Usable:** An EMIS must be usable to make it as convenient to practice radiology as conventional film-based technology.

• **Efficient:** An EMIS must be efficient to process and deliver medical images rapidly to radiologists.

• **Scalable:** An EMIS must be scalable to support the growing demands of large-scale integrated health care delivery systems [2].

• **Flexible:** An EMIS must be flexible to transfer different types of images and to dynamically reconfigure image processing features to cope with changing requirements.

• **Reliable:** An EMIS must be reliable to ensure that medical images are delivered correctly and are available when requested by users.

• **Secure:** An EMIS must be secure to ensure that confidential patient information is not compromised.

• **Cost-effective:** An EMIS must be cost-effective to minimize the overhead of accessing patient data across networks.

Developing a distributed EMIS that meets all of these requirements is challenging, particularly since certain features

conflict with other features. For example, it is hard to develop an EMIS that is efficient, scalable, and cost-effective. This is because efficiency often requires high-performance computers and high-speed networks, thereby raising costs as the number of system users increases.

2.2 Solution: Java and the Web

Over the past two years, the Java programming language has sparked considerable interest among software developers. Its popularity stems from its flexibility, portability, and relative simplicity compared with other object-oriented programming languages [5].

The strong interest in the Java language has coincided with the ubiquity of inexpensive Web browsers. This has brought the Web technology to the desktop of many computer users, including radiologists and physicians.

A feature supported by Java that is particularly relevant to distributed EMISs is the *applet*. An applet is a Java class that can be downloaded from a Web server and run in a context application such as a Web browser or an applet viewer. The ability to download Java classes across a network can simplify the development and configuration of efficient and reliable distributed applications [6].

Once downloaded from a Web server, applets run as applications within the local machine's Java run-time environment, which is typically a Web browser. In theory, therefore, applets can be very efficient since they harness the power of the local machine on which they run, rather than requiring high latency RPC calls to remote servers [7].

The MedJava distributed EMIS was developed as a Java applet. Therefore, it exploits the functionality of front-ends offered by Web browsers. An increasing number of browsers (such as Internet Explorer and Netscape Navigator and Communicator) are Java-enabled and provide a run-time environment for Java applets. A Java-enabled browser provides a Java Virtual Machine (JVM), which is used to execute Java applets. MedJava leverages the convenience of Java to manipulate images and provides image processing capabilities to radiologists and physicians connected via the Web.

In our experience, developing a distributed EMIS in Java is relatively cost effective since Java is fairly simple to learn and use. In addition, Java provides standard packages that support GUI development, networking, and image processing. For example, the package `java.awt.image` contains reusable classes for managing and manipulating image data, including color models, cropping, color filtering, setting pixel values, and grabbing bitmaps [8].

Since Java is written to a virtual machine, an EMIS developer need only compile the Java source code to Java bytecode. The EMIS applet will execute on any platform that has a Java Virtual Machine implementation. Many Java bytecode compilers and interpreters are available on a variety of platforms. In principle, therefore, switching to new platforms or upgraded hardware on the same platform should not require changes to the software or even recompilation of the Java source. Consequently, an EMIS can be constructed on a network of heterogeneous machines and platforms with a single set of Java class files.

2.3 Caveat: Meeting EMIS Performance Requirements

Despite the software engineering benefits of developing a distributed EMIS in Java, there are serious concerns with its performance relative to languages like C and C++. Performance is a key requirement in a distributed EMIS since timely diagnosis of patient exams by radiologists can be life-critical. For instance, in an emergency room (ER), patient exams and medical images must be delivered rapidly to radiologists and ER physicians. In addition, an EMIS must allow radiologists to process and analyze medical images efficiently to make appropriate diagnoses.

Meeting the performance demands of a large-scale distributed EMIS requires the following support from the JVM. First, its image processing must be precise and efficient. Second, its networking mechanisms must download and upload large medical images rapidly. Assuming that efficient image processing algorithms are used, the performance of a Java applet depends largely on the efficiency of the hardware and the JVM implementation on which the applet is run.

The need for efficiency motivates the development of high-speed JIT compilers that translate Java bytecode into native code for the local machine the browser runs on. JIT compilers are "just-in-time" since they compile Java bytecode into native code on a per-method basis immediately before calling the methods. Several browsers, such as Netscape and Internet Explorer, provide JIT compilers as part of their JVM.

Although Java JIT compilers avoid the penalty of interpretation, previous studies [9] show that the cost of compilation can significantly interrupt the flow of execution. This performance degradation can cause Java code to run significantly slower than compiled C/C++ code. Section 3 quantifies the overhead of Java and C/C++ empirically.

2.4 Key Features of MedJava

MedJava has been developed as a Java applet. Therefore, it can run on any Java-enabled browser that supports the standard AWT windowing toolkit. MedJava allows users to download medical images across the network. Once an image has been downloaded, it can be processed by applying one or more image filters, which are based on algorithms in the C source code

from xv. For example, a medical image can be sharpened by applying the Sharpen Filter. Sharpening a medical image enhances the details of the image, which is useful for radiologists who diagnose internal ailments.

Although MedJava is targeted for distributed EMIS requirements, it is a general-purpose imaging tool that can process both medical and non-medical images. Therefore, in addition to providing medical filters like sharpening or unsharp masking, MedJava provides other non-medical image processing filters such as an Emboss filter, Oil Paint filter, and Edge Detect filter. These filters are useful for processing non-medical images. For example, edge detection serves as an important initial step in many computer vision processes because edges contain the bulk of the information within an image [10]. Once the edges of an image are detected, additional operations such as pseudo-coloring can be applied to the image.

Image filters can be dynamically configured and reconfigured into MedJava via the *Service Configurator* pattern [6]. This makes it convenient to enhance filter implementation or install new filters without restarting the MedJava applet. For example, a radiologist may find a sharpen filter that uses the unsharp mask algorithm to be more efficient than a sharpen filter that simply applies a convolution matrix to all the pixels. Doing this substitution in MedJava is straightforward and can be done without reloading the entire applet.

Once an image has been processed by applying the filter(s), it can be uploaded to the server where the applet was downloaded. HTTP server implementations, such as JAWS [11, 12] and Jigsaw, support file uploading and can be used by MedJava to upload images. In addition, the MedJava applet provides a hierarchical browser that allows users to traverse directories of images on remote servers. This makes it straightforward to find and select images across the network, making MedJava quite usable, as well as easy to learn.

To facilitate performance measurements, the MedJava applet can be configured to run in benchmark mode. When the applet runs in benchmark mode, it computes the time (in milliseconds) required to apply filters on downloaded images. The timer starts at the beginning of each image processing algorithm and stops immediately after the algorithm terminates.

2.5 The OO Design of MedJava

Figure 2 shows the architecture of the MedJava framework developed at Washington University to meet distributed EMIS requirements. The two primary components in the architecture include the MedJava client applet and JAWS, which is a high-performance HTTP server also developed at Washington University [12, 11]. The MedJava applet was implemented with components from Java ACE [13], the Blob Streaming framework [14], and standard Java packages such as `java.awt` and `java.awt.image`. Each of these components is outlined below.

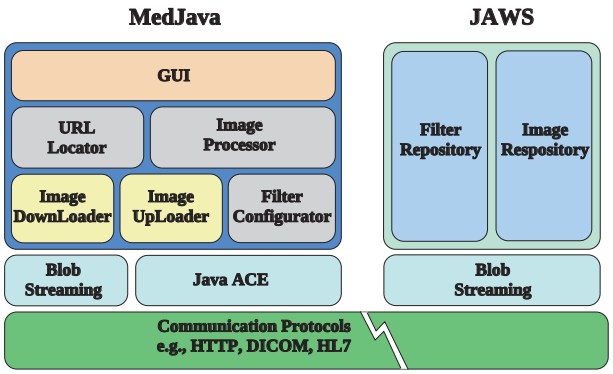

Figure 2: MedJava Framework

2.5.1 MedJava Applet

The MedJava client applet contains the following components shown in Figure 2:

Graphical User Interface: which provides a front-end to the image processing tool. Figure 3 illustrates the graphical user interface (GUI) used to display a podiatry image. The

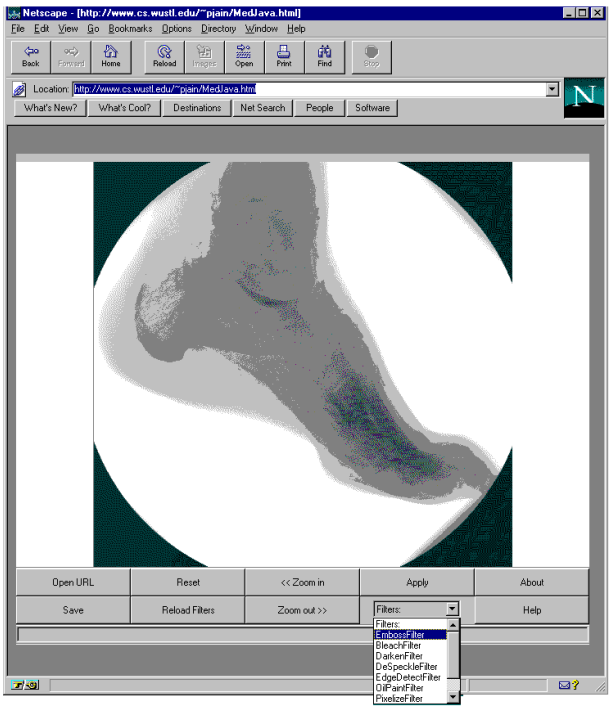

Figure 3: Processing a Medical Image in MedJava

MedJava GUI allows users to download images, apply image processing filters on them, and upload the images to a server.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.