INSIDE the JavaOS™ OPERATING SYSTEM

om Saulpaugh • Charles Mirho

Inside the JavaOS™ Operating System

Tom Saulpaugh
Charles Mirho

ADDISON-WESLEY

An imprint of Addison Wesley Longman, Inc.

Reading, Massachusetts • Harlow, England • Menlo Park, California
Berkeley, California • Don Mills, Ontario • Sydney
Bonn • Amsterdam • Tokyo • Mexico City

Corporate, Government, and Special Sales Group
Addison Wesley Longman, Inc.
One Jacob Way
Reading, Massachusetts 01867

Library of Congress Cataloging-in-publication Data
Saulpaugh, Tom
Inside the JavaOS™ Operating System, Tom Saulpaugh, Charles Mirho.
p. cm.
Includes index.
ISBN 0-201-18393-5
1. JavaOS™ operating system. 2. Operating systems (Computers) 3. Java (Computer programming language) I. Mirho, Charles A. II. Title.
QA76.76.063S3563 1999
005.4'469--dc21 98-51423
CIP

ISBN 0-201-18393-5
Text printed on recycled and acid-free paper.

1 2 3 4 5 6 7 8 9 10 – MA – 03 02 01 00 99
First Printing, January 1999

Inside the JavaOS™ Operating System

Preface

I've been hooked on operating systems ever since I took my first OS course at Cal Poly, San Luis Obispo. I was lucky enough to land a job right after graduating, in 1982, working on operating systems for Digital Research Inc. (DRI) in Monterey, California.

In June of 1985, I joined Apple's MacOS group. Apple enjoyed tremendous growth from 1985 to 1990. Each new release of the OS added more functionality (QuickDraw in color, 32-bit addressing, SCSI bus support) for more and more flavors of Macintosh. The pace of addition was staggering, so much so that Apple never had time to recode the low-level OS and fix some of its shortcomings.

By 1990, these shortcomings, including no preemptive multitasking and no memory protection for applications, began to affect the quality of the product. The Mac was the easiest computer to use but also one of the most fragile. Mac users quickly learned the location of the reboot button on the back of the box.

In June of 1990, I had lunch with Bill Bruffey of the MacOS group. Bill is a great engineer who designed the Mac's innovative file system—the Hierarchical File System (HFS). Bill had grown tired of waiting for Taligent to produce a new MacOS and he had received permission to build a new microkernel, called NuKernel, tuned for the Macintosh operating system.

He envisioned a microkernel that ran Mac applications in a virtual machine and supported a new modern concurrent input/output (I/O) system. Bill hired me as employee number one on a project that was eventually known as Copland.

Fred Brooks could easily write a modern version of *The Mythical Man-Month* about the Copland project. Copland started lean and mean, with Bill hiring just four more engineers during that first year. After just a few months of work, we five demonstrated to management a microkernel-based MacOS running on a MacII-ci. The project gained steam over the next few years and eventually grew to more than 500 employees, and Bill and I became two contributing engineers with no management authority.

Somewhere during the middle of the Copland project, management asked me to back-port some Copland I/O technology to a new family of Macintosh computers sporting the PCI expansion bus. I took a year off from the Copland project and helped Apple ship a PowerPC-native device driver architecture for its new PCI-based Macs.

The PCI team was focused and lean. A small team of engineers built and deployed a large amount of software in a year's time with none of the bureaucratic overhead of the Copland project. My time working on PCI for System 7.5 proved to be the most enjoyable year of my Apple career.

When I returned to the Copland project in June of 1995, I found a mess. The Copland leadership had decided to recode the toolbox and break popular existing system extensions such as After Dark. Apple had gambled that users and developers wouldn't mind a new OS that wasn't a hundred percent backward compatible! I made up my mind that summer to leave Apple.

In May of 1995, Sun Microsystems introduced Java at SunWorld. As the Java phenomenon materialized over the next six months, Jim Mitchell and Peter Madany of Sun's JavaSoft began to build a new OS (code-named Kona) to run only Java software.

I was hired in March of 1996 to design an I/O architecture for Kona, soon to be renamed JavaOS™. The early Kona team consisted of seven people. The team was extremely focused and produced the first official release of the JavaOS operating system in just 15 months. After my experience with the Copland project, I felt lucky and honored to be working with bright, focused people on an innovative operating system.

In early 1997, JavaSoft handed over control of JavaOS to SunSoft. Late that year, the SunSoft JavaOS team, headed by Bob Rodriguez, began working closely with an IBM team to build the next release of JavaOS, eventually renamed JavaOS for Business™. The contributions from IBM were significant and included many key architectural features.

This book provides an inside look at the results of Sun's and IBM's efforts to build a new thin-client operating system. The book uses the name JavaOS throughout, but the version of the JavaOS operating system presented here is JavaOS for Business.

Tom Saulpaugh
Senior Staff Engineer
Sun Microsystems, Inc.

When I first learned about the JavaOS operating system, I was a second-year, part-time law student at Santa Clara University, with a full-time job writing patents during the day. The last thing I needed was another distraction in my life. But I have always been fascinated by operating systems, which I consider the most intricate and complex software programs on the planet. A new operating system based around, and written in, the Java programming language was intriguing. Think of the possibilities: system services loaded on demand and distributed execution between client and server, or even on multiple clients and multiple servers! A single OS code base, regardless of client or server hardware architecture, residing

in a central location. An end to complicated software upgrades—simply *subscribe* to your operating system and applications, and the latest upgrades and bug fixes magically appear each time you boot up. These are some of the possibilities opened up by JavaOS technology.

When I first met Tom Saulpaugh, he was an Apple Computer refugee who had just recently joined JavaSoft. The JavaOS team was only about ten people, and there was a sense that the rest of JavaSoft didn't see the potential of this new technology. Someone needed to get the word out. Tom, myself, and a hard-driving Sun technical writer, Tom Clements, set out to do just that. First came an article in *BYTE* magazine, a bit of undisguised evangelism. Next a meeting with James Gosling, at which we pitched the merits of JavaOS with regards to the Java language itself. Things started to happen.

A Sun product group took responsibility for the JavaOS operating system from JavaSoft, Chorus was purchased for their microkernel technology, IBM signed on to co-develop and market JavaOS, and the team grew. I'm certainly not going to take credit for making JavaOS a success; I was mostly an outsider looking in, but I like to think my early enthusiasm had some impact on getting folks to stand up and take notice. It is safe to say that the time for a book on JavaOS has arrived.

Inside the JavaOS™ Operating System is about using Java technology to make an operating system simpler, more reliable, more powerful, and easier to maintain. In this spirit of simplicity and power, we have tried to create a book that explains the workings of JavaOS in simple, concise terms. This was not always easy, because operating systems are by their nature obscure and complex beasts. I hope you enjoy reading about JavaOS as much as we enjoyed writing about it.

Charles Mirho

Acknowledgments

This book was two years in the making. It was written as the software was developed and has changed many times. Along the way, the following people have contributed to its content: Jeff Schmidt, Anne Bluntschli, Tom Clements, Bill Kain, and Bruce Montague.

Among the tremendous engineering talent from Sun who worked on this operating system are: Rajeev Bharadhwaj, Bob Rodriguez, Ron Karim, Dennis Aaron, Mohamed Abdelaziz, Angela Byrum, Jagane Sundar, Rich Berlin, Nedim Fresko, Mike Shoemaker, Dean Long, Graham Hamilton, Lisa Stark-Berryman, Greg Slaughter, Bill Keenan, Jason Li, Tom Mason, Tim Sia, Don Hudson, Ed Goei, Bernard Traversat, Sam Yan, Eric Yeh, and Mercia Zheng.

A special thanks goes to two Sun individuals, Bob Delaney and Ron Kleinman, who provided valuable insight into the operating system's feature set.

Steve Woodward, Bill Tracey, Mike Sullivan, Jonathan Wagner, Sheila Harnett, Joe Tano, and Les Wilson were among the great IBM engineering contributors.

Above all, thanks to Maureen, Matthew, Evan, Rachel, and Erika and Max for their encouragement and support.

Introduction

Why a New OS?

JavaOS™ is a new commercial operating system (OS) developed by Sun Microsystems, Inc., and IBM. A commercial operating system is perhaps the most complicated piece of software anyone can endeavor to build and maintain.

Once deployed, a successful operating system takes on a life of its own. Device drivers, tools, and applications are built to take advantage of the new OS. In turn, the OS is bug-fixed and expanded to reward early software developers with more functionality and, if all goes well, more performance and reliability. Early pioneering users are asked to be patient as the system matures. Typically any operating system does not mature until its third major release.

Today, companies such as IBM, Microsoft, Apple, and Sun Microsystems put so much time, effort, and money into developing, enhancing, and maintaining an OS that very few new operating systems are built any more. Simply put, in the current market there must be a compelling reason to build a new commercial OS.

Yesterday's Reason: A New Hardware Architecture

In the past, new commercial operating systems typically were created to take advantage of the power of a new computer hardware architecture, or platform. As you know, a computer's architecture, consisting of the set of attributes that determine what software will run on the computer, is used as a blueprint to *build* the computer.

New computer architectures were created when significantly greater functionality and performance become possible with a new family of "iron." By far, the most common reason to build a new computer-hardware architecture was to deliver more memory-addressing capability, such as Digital Equipment Corporation's introduction of the VAX architecture.

Computer-addressing capability is measured by the size, in bits, of an address. A 32-bit address, for example, yields 4GB of addressable memory space, or *address space*. A 32-bit address space enables a computer to run larger and more complex applications than is possible with, say, a 16-bit address space.

Today's Reason: Java™ Technology

In the 1970s, IBM and Digital Equipment Corporation helped to standardize the business and scientific computing worlds. In the 1980s, Apple, IBM, and Microsoft introduced standardized personal computing to the individual, and Sun introduced the workstation.

The decade of the 1980s saw a tremendous consolidation of computing based on this evolving set of common standards. These standards, because of their value to the consumer, permeated to the deepest levels of the platform—the CPU and devices.

The IBM PC architecture was defined in 1981; the turbo-charged PC of today is a superset of that original architecture. The burden of supporting years of legacy hardware and software products grows with each year that passes. Surprisingly, while hardware has advanced by leaps and bounds, operating system technology has progressed more slowly. Most operating systems today are still written largely in C, C++, and assembly code. These software technologies place a practical limit on what the OS can do.

The Java programming language opens new possibilities in OS design. A large portion of the system software is entirely insulated from the underlying platform. This enables a degree of standardization, centralization, and footprint customization that was simply impractical with native code OSs. It also enables a more secure and robust OS environment because of the Java programming language's inherent fail-safe features.

JavaOS for Business™ represents the most advanced JavaOS implementation, and many of the technologies described in this book were implemented first there. Already, some JavaOS technologies have found there way into other environments. The JavaOS System Database and the portions of the device driver architecture are good examples of this, having found there way into point-of-sale applications.

How This Book Is Organized

This book takes a high-level look at the JavaOS operating system but does not cover specific programming interfaces in great detail. The JavaOS design is presented top-down, beginning at the highest level and progressing layer by layer more deeply into the operating system. The reader is assumed to be familiar with the Java programming language and with the Java Development Kit (JDK).

Chapter 1, Introduction, explores the evolution of JavaOS from a simple stand-alone Java Virtual Machine (JVM) to a modern, microkernel-based operating system.

Chapter 2,The Database, covers the JavaOS System Database (JSD) which may be used to configure the operating system.

Chapter 3, Events, presents the JavaOS Event System which may be used to support automatic plug-and-play devices.

Chapter 4, Service Loader, presents the JavaOS Service Loader (JSL) which may be used to load operating system services such as device drivers.

Chapter 5, Standard Device Support, gives information on support for standard JDK devices, such as network and graphic devices.

Chapter 6, Device Drivers, gives an overview of the JavaOS Device Interface (JDI) and JavaOS device driver architecture.

Chapter 7, Memory, presents the JavaOS memory model.

Chapter 8, Interrupts, covers the JavaOS interrupt model.

Chapter 9, The Microkernel, covers the JavaOS microkernel.

Chapter 10, Booting, presents the JavaOS boot architecture and the JavaOS Boot Interface (JBI).

Contents

Chapter 1
Overview

The JavaOS™ software is a new operating system (OS) optimized to run software written in the Java™ programming language on a variety of devices, from embedded platforms to network computers. (We use the term "JavaOS" in place of "JavaOS operating system" throughout this text.) As an alternative to hosting the Java Development Kit (JDK) on a native OS, JavaOS provides a standalone JDK hosting environment. More than two-thirds of JavaOS is written in the Java programming language, with the remainder written in C and a small assembly language component.

Because JavaOS is new, it provides a minimal implementation for hosting the JDK. There is no extra code in the system for supporting legacy applications. Because a large portion of the OS is software written in the Java programming language (which we shall refer to as simply "Java software"), JavaOS is object-oriented and portable. As you will learn later in this chapter, JavaOS may serve as an "incubator," gradually reducing the amount of native code necessary for hosting the JDK.

1.1 Evolution of JavaOS

JavaOS began its existence as a platform for embedded devices. James Gosling, working for Sun Microsystems, created a small runtime environment and language definition for enabling cross-platform programming on small, consumer electronic devices. Gosling designed Java as an interpreted language running on a virtual machine, a software central processing unit (CPU) with its own instruction set. In essence, this virtual machine created an abstraction of the physical CPU that programs could target in a device-independent way.