WHITE PAPER

The CLDC HotSpot™ Implementation Virtual Machine

JAVA™ 2 PLATFORM, MICRO EDITION (J2ME™)

The CLDC HotSpot™ Implementation Virtual Machine

The deployment of Java™-enabled wireless devices reached nearly 15 million units in 2001 and will likely exceed 100 million in 2002. (Source: Future Mobile Handsets, Arc Group, May 2001.) This trend is expected to continue at a nearly exponential pace in the next few years.

Connected Limited Device Configuration HotSpot™ Implementation (CLDC HotSpot™ Implementation) is Sun's new high-performance Java virtual machine for embedded devices. The first generation of Java™-enabled wireless devices are based on the KVM (K virtual machine), and KVM deployments are continuing. CLDC HotSpot Implementation promises to deliver nearly an order of magnitude better performance than the KVM, while running in the small memory footprint required by devices such as mobile phones.

The deployment of Java technology is well under way into mass-market consumer devices such as mobile phones, wireless e-mail clients, and personal organizers. While the market penetration of current generation mobile phones has not yet reached its peak, the major manufacturers are already working hard on improved 2.5G and next generation designs. Such phones have greater demands in performance and data bandwidth due to features such as multimedia. The drive for better performance in embedded Java runtime environments has led Sun Microsystems to develop a new Java virtual machine technology that promises to deliver nearly an order of magnitude better performance than the KVM-based devices that are currently being shipped.

The name for this new virtual machine technology is CLDC HotSpot Implementation. It borrows techniques from Sun Microsystems' earlier revolution in virtual machine performance, the Hotspot™ performance engine. In addition, it incorporates several innovations in design that allow the virtual machine to run in resource-constrained devices. In general, CLDC HotSpot Implementation is intended to

- deliver cutting edge performance,
- deliver fast application startup time,
- requires minimal footprint,
- preserve battery life.

Version 1.0 of CLDC HotSpot Implementation now being offered by Sun Microsystems is integrated with CLDC. This initial offering conforms to the CLDC Specification version 1.0 and Technology Compatibility Kit (TCK) 1.0. To complete the Java technology stack, a compatible implementation of the MIDP 1.0 Specification is also offered.

Java Technology in Small Devices

A complete Java technology stack exists today to support embedded devices such as mobile phones. The stack is based on the Java 2 Platform, Micro Edition (J2ME™), and includes layers from the Java virtual machine to GUI support. These devices are characterized as small, battery-powered devices with limited, wireless connection to the Internet.

J2ME defines configurations and profiles, which, in combination with a Java virtual machine, make up the Java technology stack. A configuration of J2ME includes a Java virtual machine, as well as the Java programming language libraries that are required as the lowest common denominator of a range of embedded devices. A profile is a layer on top of the configuration that provides additional APIs for a specific class of devices. A particular combination of configuration and profile is appropriate only for specific Java virtual machines.

J2ME fits in with the other editions of Java, J2SE and J2EE, as illustrated in FIGURE 1. Up to now, small, battery-powered devices are the domain of the KVM and the Mobile Information Device Profile (MIDP), also shown.

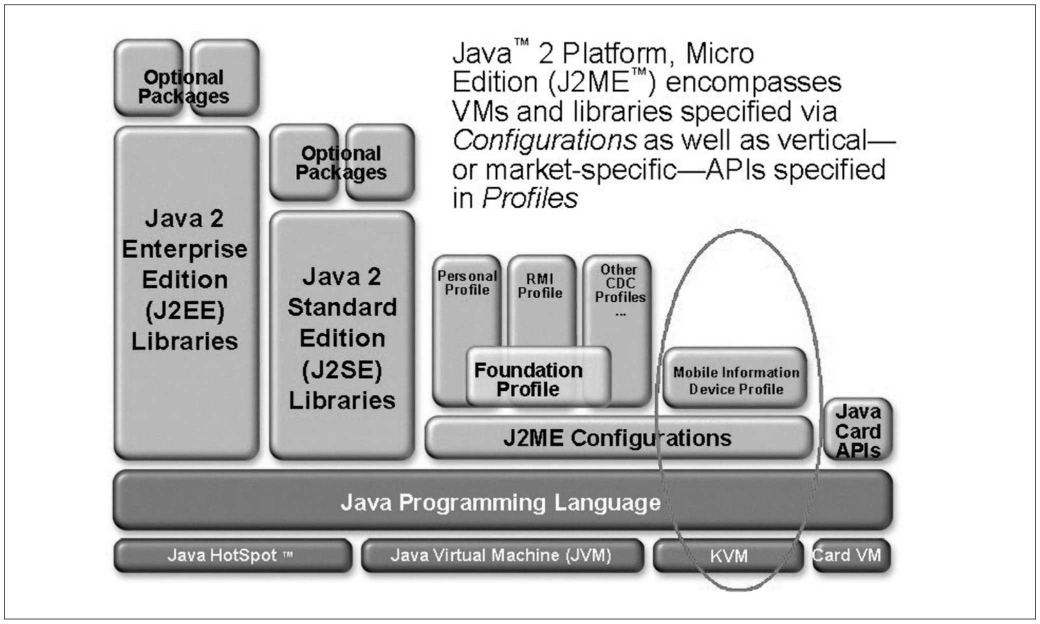
Java™ 2 Platform, Micro Edition (J2ME™) encompasses VMs and libraries specified via Configurations as well as vertical—or market-specific—APIs specified in Profiles
Optional Packages
Java 2 Enterprise Edition (J2EE) Libraries
Optional Packages
Java 2 Standard Edition (J2SE) Libraries
Personal Profile
RMI Profile
Other CDC Profiles ...
Foundation Profile
Mobile Information Device Profile
J2ME Configurations
Java Card APIs
Java Programming Language
Java HotSpot ™
Java Virtual Machine (JVM)
KVM
Card VM

Figure 1. J2ME, KVM and MIDP

CLDC HotSpot Implementation is now poised to take the place of the KVM as the high-performance Java virtual machine for the next generation of embedded devices.

History of the Java Stack for Mobile Phones

In 1998, the *Spotless* research initiative at Sun Microsystems Laboratories created a compact version of the Java virtual machine to run on small, handheld, battery-powered devices. For the first time, it was possible to write applications in the Java programming language that could be run on such devices. Thus, a revolution was born that, today, sees the deployment of tens of millions of Java technology-enabled small devices such as mobile phones. (Sun Labs publication 1999-0169.)

Spotless evolved to become the *K Virtual Machine* (KVM), a key component of Java 2 Micro Edition (J2ME). It also spawned the J2ME CLDC (Connected, Limited Device Configuration), targeted at small mass-market consumer devices such as mobile phones, wireless e-mail devices, and personal organizers.

CLDC 1.0/KVM

Working through the Java Community Process (JCP), the CLDC configuration was created to provide core Java library support to provide a basic application framework around the KVM. JSR-30 was approved in August 1999, and the final public release of the CLDC Specification 1.0 occurred in May 2000. Practically every major manufacturer of mobile phones, as well as PDA manufacturers and software vendors, participated in the JCP expert group that developed CLDC 1.0.

Mobile Information Device Profile (MIDP)

In addition to a configuration, J2ME technology requires that a profile be defined to provide a complete Java application framework for a particular market segment. See Chapter 2 of *J2ME Building Blocks for Mobile Devices, White Paper on KVM and the Connected, Limited Device Configuration (CLDC),* (Sun Microsystems, Inc., 2000). The MID profile (Mobile Information Device Profile, MIDP) was created through the Java Community Process to address the limited screen size and battery power of this class of device. JSR-37 was approved in September 1999, and the final public release of the MIDP Specification 1.0 occurred in September 2000.

Wireless Deployments

In 2001, major manufacturers of mobile phones, such as Motorola, Nokia, and Siemens, and mobile operators such as NTT DoCoMo, J-Phone and Nextel, began shipping Java technology enabled phones in high volume. It is estimated that the number of units in the field based on J2ME will approach 15 million by the end of 2001, and should reach 108 million by the end of 2002. (Source: *Future Mobile Handsets,* Arc Group, May 2001.)

Further projections of market penetration of Java technology-enabled phones are 420 million by 2003 and 680 million in 2004. These numbers reflect the adoption of new Java technologies such as CLDC HotSpot Implementation.

The Hotspot™ Virtual Machine

At about the same time that the Spotless project began, a revolutionary Java virtual machine technology called Hotspot was nearing product deployment. The Hotspot™ performance engine was developed to address the perception that Java virtual machine performance was insufficient for many mainstream applications especially on big servers. By implementing a host of performance enhancing techniques that went beyond innovations like just-in-time (JIT) compilers, the performance of the Java virtual machine increased by an order of magnitude. Hotspot technology was rolled out in April 1999. (See the Java HotSpot Virtual Machine Technical White Paper, Sun Microsystems, 2001.)

In 2001, these two technology trends converged to inspire the creation of the CLDC HotSpot Implementation virtual machine. The feat of creating Java technology-enabled consumer devices with KVM and CLDC is impressive, but the perception is forming in the marketplace that here, as in conventional Java technology, there will ultimately be a need for faster performance. By applying optimization techniques similar to those used in Hotspot, but using considerably less memory and consuming less power, nearly an order of magnitude improvement can be realized in CLDC-based devices.

Demand for Performance

The current generation of Java technology-enabled mobile phones have processor and memory requirements that are typical of the original design parameters of the KVM and CLDC. The typical processor is a 16 or 32-bit processor with a clock speed starting from approximately 12-32 MHz, with a memory budget for the Java virtual machine and libraries of about 512 kilobytes. Although the KVM easily met the footprint requirements of this generation of target devices, the relatively slow processor and the conventional implementation of a bytecode interpreter resulted in performance that was adequate but not impressive. Sun Microsystems began to examine the possibility of accelerating performance in the current generation of devices, while looking ahead to the next generation mobile phone designs.

Before finalizing the features of CLDC HotSpot Implementation, the development team surveyed key manufacturers to get an accurate picture of the capabilities of current generation and next generation mobile phone designs.

Processor and Memory Requirements

The following table summarizes processor and memory configurations for next generation mobile phones. (Source: Sun Microsystems customer survey, 2001.)

CPU type	mostly ARM
CPU speed	30-400 MHz
On board RAM	128-384 kB
RAM	1-4 MB
ROM / Flash	8-24 MB
RAM for Java stack	mostly under 1 MB

TABLE 1. Next generation mobile phone capabilities

Key Points

Our survey revealed that the following key points are important to manufacturers of current generation and next generation mobile phones:

• Most of the available memory in a current generation or next generation handset is needed for system software and media capabilities. Thus, the memory footprint of the virtual machine and CLDC libraries must be minimized.

• Moore's Law does not apply to battery life: so far, no exponential expansion of battery capacity with the passage of years has been observed. Every effort must be made to minimize battery consumption for the foreseeable future.

• The key to executing Java programs at high speeds without draining the battery is keeping the working set of the Java virtual machine inside the on-processor cache.

• Tunability is key: Implementers must be able to use different size parameters and policies per device.

Other Small Consumer Devices

Besides mobile phones, the CLDC HotSpot Implementation development team also considered the processor and memory requirements of other devices that potentially belong in the CLDC and MIDP category, such as wireless personal organizers (PDAs) and communicators.

Communicator type devices typically have much more memory available than inexpensive mass market handsets, but they are also manufactured in much smaller volume. Although footprint constraints are much less stringent in this class of device, the next generation of Java virtual machine technology for embedded devices must be appropriate for smaller, high-volume handsets as well.

Value Proposition

There was a perception early in the history of the Java programming language that the performance of the applications written in the Java programming language was inadequate. With the advent of the Hotspot performance engine, the competitive landscape was revolutionized for Java virtual machines on servers and on the desktop. In much the same way, CLDC HotSpot Implementation will revolutionize the deployment of Java technology in battery-powered, handheld devices. The performance of the CLDC HotSpot Implementation virtual machine approaches that of Java virtual machines running on desktop systems. It does so using techniques such as:

- Dynamic compilation
- Generational garbage collection
- Fast synchronization
- Unified resource management

To apply these techniques in the context of handheld devices, some very clever innovations were necessary. (Refer to "CLDC HotSpot Implementation Architecture" on page 10.)

CLDC HotSpot Implementation is a clean 32 bit virtual machine that complies with the *CLDC Specification*, version 1.0. Except for the areas documented in Chapter 4 of the *CLDC Specification*, CLDC HotSpot Implementation is fully compliant with the *Java™ Virtual Machine Specification* and the *Java™ Language Specification*. CLDC HotSpot Implementation places no restrictions on the number of loaded classes or the size of the object heap.

Despite its high performance, CLDC HotSpot Implementation is compact enough to meet the footprint constraints of next generation and many current generation mobilephones. The total memory requirement for the virtual machine and software is less than 1 Mb. This includes the CLDC HotSpot Implementation virtual machine, the CLDC class libraries, the MIDP class libraries, and Java applications.

The manufacturers who have successfully developed and deployed Java technology-enabled handsets might feel little competitive pressure to change their offerings. However, there is a substantial value to upgrading their offerings to incorporate CLDC HotSpot Implementation technology.

CLDC HotSpot Implementation versus the KVM

In the KVM design, a heavy emphasis was placed on portability and platform-independence of the virtual machine. Consequently, the KVM is a conventional virtual machine that executes Java applications exclusively by means of a bytecode interpreter written in ANSI C. However, measurements reveal that, on average, interpreted virtual machine performance is approximately one order of magnitude slower than compiled virtual machine performance.

To improve the performance of a virtual machine beyond pure interpreter performance, some kind of a static or dynamic compilation strategy is needed. Static compilation of Java applications is an undesirable solution in the wireless space, where it is important for third parties to quickly develop applications and deploy them widely, and to take advantage of Over-The-Air (OTA) provisioning to distribute bytecode streams to handsets in the field.

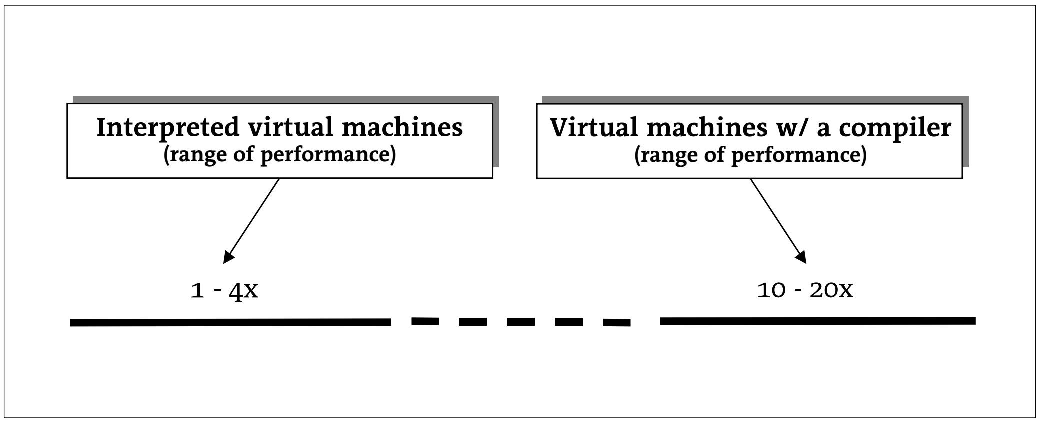

FIGURE 2. Performance of interpreted and JIT virtual machines

Additional performance enhancement compared to straightforward virtual machines is achieved with a HotSpot-style garbage collector and a fast synchronization mechanism.

Faster execution consumes less power

The dramatic improvement in performance of CLDC HotSpot Implementation “turbocharges” application startup time and execution time, resulting in a positive subjective experience. Just as importantly, it consumes battery power at a proportionally lower rate.

The increasing demands of 2.5G and next generation networks

With the emergence of 2.5G and next generation networks, the performance demands are dramatically increasing for on-phone applications and data communications.

Next generation mobile networks will support data bandwidth rates from 384Kbits per second to 2 Mbits per second, opening up new possibilities for applications in the areas of:

- Games and gambling applications
- Multimedia applications
- Location based services
- E-commerce applications
- System software
- Banking applications

The virtual machine must provide sufficient performance for these new types of applications while minimizing battery drain. Paradoxically, battery power can be optimized even though a faster processor consumes battery power at a proportionally faster rate. A very fast virtual machine such as CLDC HotSpot Implementation makes possible an overall savings in power even while servicing this new generation of software, because it finishes all tasks much sooner than a slower virtual machine.

CLDC HotSpot Implementation Design Challenges

A set of fundamental challenges had to be addressed in the CLDC HotSpot Implementation design:

- The trade-off of fast execution versus small footprint
- Good cache behavior
- Enhancing battery efficiency
- The need for tunable parameters

Speed Versus Footprint. There is a seeming trade-off between speed of execution and memory (footprint) requirements. How can one build a fast dynamic compiler without blowing the memory budget? To simply port the Hotspot technology would result in a memory footprint far too large for mass market, battery-powered devices.

Good Cache Behavior. The importance of cache behavior might not be obvious at first. Abundant memory adds to manufacturing cost, although Moore's law tempts designers to waste memory. But additional memory—especially RAM—also puts a great load on battery capacity. It was a prime design objective of CLDC HotSpot Implementation to obtain good cache behavior so that the working set for Java stack could fit within the on-processor or in the secondary (on-board) cache. In this way, substantial battery conservation is achieved by avoiding reads and writes to the main memory array.

The design objective of good cache behavior implied a number of software strategies:

- Designing the virtual machine with mostly small objects
- Use of a generational garbage collector, which often touches memory only locally
- Keeping compiled code in the object heap, where it is fully relocatable or flushable

Enhancing Battery Efficiency. It bears repeating that the leap in execution speed provided by CLDC HotSpot Implementation directly enhances battery life. Quite simply, faster execution consumes less power.

The need for tunable parameters. A high level design must also be tempered by a consideration of real devices.

- Cache behavior varies greatly between devices
- CLDC HotSpot Implementation's flexible design allows device-specific tuning

CLDC HotSpot Implementation Architecture

The architecture of the CLDC HotSpot Implementation virtual machine includes the following features:

- Pure 32 bit virtual machine
- Compact object layout
- Unified resource management
- Accurate generational garbage collection
- Optimized interpreter
- Adaptive compilation, which only compiles the most used methods
- Fast synchronization
- No restriction on number of loaded classes

Pure 32 bit virtual machine

CLDC HotSpot Implementation is a pure 32 bit virtual machine. This provides a large address space and scalable architecture well-suited for mid- to high-end mobile phones. It is especially suited for the emerging 2.5G and next generation mobile phones, which typically have larger memory capacity.

Compact Object Layout

CLDC HotSpot Implementation supports a compact object layout to reduce general memory consumption. A Java object has two parts. The first part is the object header, which provides reflective information and contains hash code and locking status. The second part is the object body, containing the object fields.

Most other virtual machines use at least two words for the object header. However, since the average object size is small, object headers take up a big fraction of the total object space.

CLDC HotSpot Implementation introduces a new design, in which only one word is needed for the object header. In addition to reducing memory usage, object allocation becomes faster.

Unified Resource Management

A major benefit of CLDC HotSpot Implementation is unified resource management. This means that all allocated data resides inside the object heap. Allocated data includes:

- Java level objects,
- Reflective objects, such as methods and classes,
- Compiler generated code, and
- Virtual machine internal data structures.

An important advantage of this unification is that the same garbage collector takes care of cleaning up all allocated resources, even compiled code. Almost all other virtual machines have designated areas for user objects, reflective data, temporary data and generated code. Such a scheme results in memory fragmentation, multiple cleanup strategies and other complexities. CLDC HotSpot Implementation solves these issues by using the mark-sweep-compact garbage collector for everything.

Another benefit of unified resource management is that compiled code can be removed dynamically to free up space for user-level objects.

The CLDC HotSpot Implementation Garbage Collector

A garbage collector automatically reclaims unused object memory and makes the freed memory available for new allocations. CLDC HotSpot Implementation uses an accurate generational mark-sweep-compact garbage collector, which results in:

- Fast object allocation
- Small garbage collection pauses
- No memory fragmentation

Accuracy

An accurate garbage collector knows where all pointers are when garbage collection takes place. This has two major benefits. First, all inaccessible object memory can be reclaimed reliably. Second, all objects can be relocated, allowing object memory compaction and eliminating fragmentation. Using a conservative garbage collection approach would be highly undesirable on a memory-constrained system, since it causes object fragmentation and unpredictable memory leaks.

Generational Mark-Sweep-Compact Collector

The CLDC HotSpot Implementation virtual machine employs a two generational garbage collector, as illustrated in FIGURE 3.

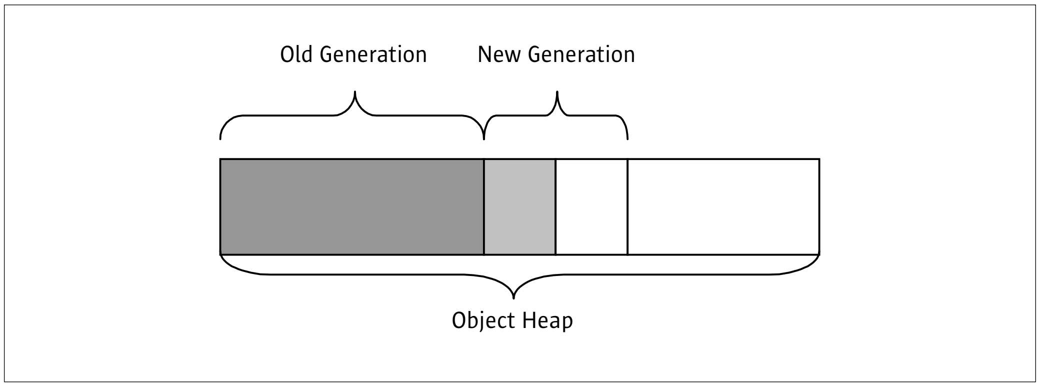

FIGURE 3. Two-generational Garbage Collection

The object heap is segmented into old generation, new generation and as-yet-unused portions of memory. The old generation segment contains objects that were previously garbage collected and compacted. New objects are allocated in the new generation segment, which is generally much smaller. When the new generation segment is full, the garbage collector runs briefly and reclaims the unused memory for that generation. When all memory in the object heap is consumed, the garbage collector runs across the entire heap and compacts objects into a "new" old generation. Only during this large garbage collection is there a noticeable pause, but it occurs infrequently.

This scheme takes advantage of the fact that the vast majority of objects are short-lived. Since most objects are short-lived, only a small portion of allocated objects are promoted to the old generation. Most garbage collection operations focus only on the new generation, resulting in only small pauses.

Tracking Pointers Across Generations

One requirement of a generational system is the ability to track pointers from old generation to new generation. For this, CLDC HotSpot Implementation uses a write barrier.

Whenever a pointer store takes place, the field is marked as a possible future pointer from old to new generation.

Fast Allocation

A side benefit of a compacting garbage collecting is that new objects are allocated contiguously in stack-like fashion in the first generation. Object allocation is then simply a matter of increasing a pointer.

Execution Engine

In general, Java virtual machines with a compiler are an order of magnitude faster than those with only an interpreter. For that reason, CLDC HotSpot Implementation includes a dynamic compiler to provide fast bytecode execution. A well-known problem with compiling bytecodes into native instructions is that the generated code takes up four to eight times as much space as the original bytecodes. Adaptive compilation alleviates this problem by only compiling methods that are recognized as "hotspots", i.e., the most frequently used parts of the application. The CLDC HotSpot Implementation dynamic compiler finds the hotspot by running a statistical profiler.

To minimize the amount of compiled code, the CLDC HotSpot Implementation virtual machine includes an optimized interpreter used for infrequently executed methods.

The CLDC HotSpot Implementation compiler is a simple one-pass compiler that utilizes the following basic optimizations: constant folding, constant propagation, loop peeling.

The components of the CLDC HotSpot Implementation virtual machine are shown in FIGURE 4.

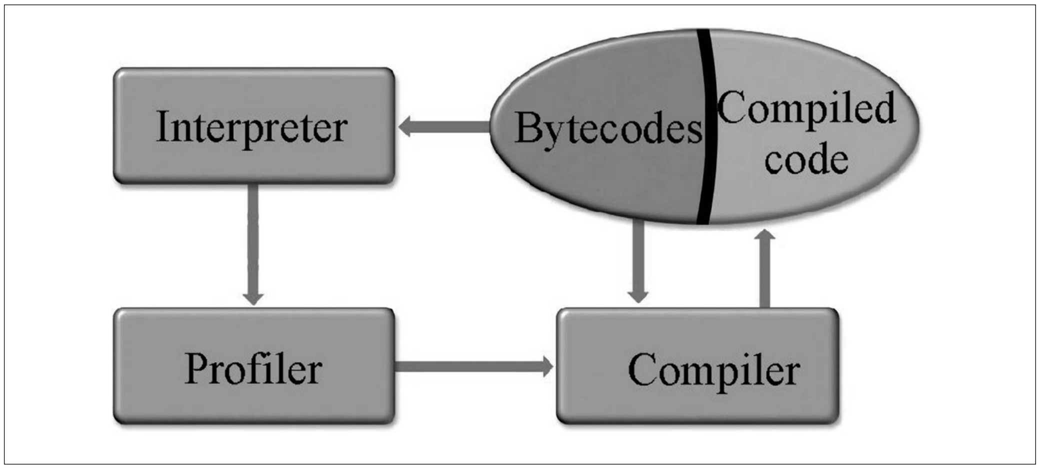

FIGURE 4. CLDC HotSpot Implementation Architecture

Fast Thread Synchronization

The Java programming language provides language-level thread synchronization, which makes it easy to express multi threaded programs with fine-grained locking. CLDC HotSpot Implementation uses a variant of the block structured locking mechanism developed for the HotSpot virtual machine. As a result, synchronization performance becomes so fast that it is no longer a performance bottleneck for Java programs.

Conclusion

To keep pace with the demands for performance of the next generation of mobile phones and other wireless devices, Sun Microsystems saw the need for a new Java virtual machine technology. The result is CLDC HotSpot Implementation, which achieves a performance gain of nearly an order of magnitude compared to the first generation of J2ME deployments. The CLDC HotSpot Implementation virtual machine was demonstrated on real devices at Java One 2002. This technology is available to device manufacturers under license from Sun Microsystems.

NOTES

HEADQUARTERS **SUN MICROSYSTEMS, INC.,** 901 SAN ANTONIO ROAD, PALO ALTO, CA 94303-4900 USA
PHONE: 650 960-1300 FAX: 650 969-9131 INTERNET: www.sun.com

Sun microsystems
We make the net work.

SALES OFFICES

AFRICA (NORTH, WEST AND CENTRAL): +9714-3366333 • **ARGENTINA:** +5411-4317-5600 • **AUSTRALIA:** +61-2-9844-5000 • **AUSTRIA:** +43-1-60563-0 • **BELGIUM:** +32-2-704-8000 • **BRAZIL:** +55-11-5187-2100 • **CANADA:** +905-477-6745 • **CHILE:** +56-2-3724500 **COLOMBIA:** +571-629-2323 • **COMMONWEALTH OF INDEPENDENT STATES:** +7-502-935-8411 • **CZECH REPUBLIC:** +420-2-3300-9311 • **DENMARK:** +45 4556 5000 • **EGYPT** +202-570-9442 • **ESTONIA:** +372-6-308-900 • **FINLAND:** +358-9-525-561 • **FRANCE:** +33-01-30-67-50-00 **GERMANY:** +49-89-46008-0 • **GREECE:** +30-1-618-8111 • **HUNGARY:** +36-1-202-4415 • **ICELAND:** +354-563-3010 • **INDIA:** +91-80-5599595 • **IRELAND:** +353-1-8055-666 • **ISRAEL:** +972-9-9710500 • **ITALY:** +39-039-60551 • **JAPAN:** +81-3-5717-5000 **KAZAKHSTAN:** +7-3272-466774 • **KOREA:** +822-2193-5114 • **LATVIA:** +371-750-3700 • **LITHUANIA:** +370-729-8468 • **LUXEMBOURG:** +352-49 11 33 1 • **MALAYSIA:** +603-264-9988 • **MEXICO:** +52-5-258-6100 • **THE NETHERLANDS:** +00-31-33-45-15-000 **NEW ZEALAND:** +64-4-499-2344 • **NORWAY:** +47 23 36 96 00 • **PEOPLE'S REPUBLIC OF CHINA:** • **BEIJING:** +86-10-6803-5588 • **CHENGDU:** +86-28-619-9333 • **GUANGZHOU:** +86-20-8755-5900 • **SHANGHAI:** +86-21-6466-1228 • **HONG KONG:** +852-2202-6688 **POLAND:** +48-22-8747800 • **PORTUGAL:** +351-21-4134000 • **RUSSIA:** +7-502-935-8411 • **SINGAPORE:** +65-438-1888 • **SLOVAK REPUBLIC:** +421-7-4342 94 85 • **SOUTH AFRICA:** +2711-805-4305 • **SPAIN:** +34-91-596-9900 • **SWEDEN:** +46-8-631-10-00 **SWITZERLAND:** • **GERMAN:** 41-1-908-90-00 • **FRENCH:** 41-22-999-0444 • **TAIWAN:** +886-2-2514-0567 • **THAILAND:** +662-636-1555 • **TURKEY:** +90-212-335-22-00 • **UNITED ARAB EMIRATES:** +9714-3366333 • **UNITED KINGDOM:** +44-1-276-20444 **UNITED STATES:** +1-800-555-9SUN OR +1-650-960-1300 • **VENEZUELA:** +58-2-905-3800

SUN™