

# CLDC HotSpot™ Implementation Virtual Machine

Java™ 2 Platform, Micro Edition (J2ME™) Technology  
February 2005



# Table of Contents

<b>Executive Summary</b> .....	<b>.1</b>
<b>Java Technology in Small Devices</b> .....	<b>.3</b>
History of the Java Stack for Mobile Phones .....	.4
<b>Demand for Performance</b> .....	<b>.6</b>
Processor and Memory Requirements .....	.6
Key Requirements .....	.7
Other Small Consumer Devices .....	.7
<b>Design Considerations</b> .....	<b>.8</b>
<b>Value Proposition</b> .....	<b>.9</b>
Performance Advantage .....	.9
Robustness and Short Time to Market .....	.9
Scalability and Small Footprint .....	.10
<b>The CLDC HotSpot Implementation Architecture</b> .....	<b>.12</b>
Dynamic, Adaptive Compiler .....	.12
Compact Object Layout .....	.13
Unified Resource Management .....	.13
The CLDC HotSpot Implementation Garbage Collector .....	.14
Fast Thread Synchronization .....	.15
Lightweight Threads .....	.15
Thumb Mode Support (ARM Processors) .....	.15
<b>New Features in Version 1.1.2</b> .....	<b>.16</b>
AOT Option .....	.17
In-Place Execution .....	.17
Shorter Execution Pauses .....	.17
Multitasking Option .....	.17
<b>Multitasking in the CLDC HotSpot Implementation</b> .....	<b>.18</b>
Why Multitasking Capability? .....	.18
<b>Conclusion</b> .....	<b>.20</b>

## Chapter 1

# Executive Summary

The Connected Limited Device Configuration (CLDC) HotSpot™ Implementation is Sun's high-performance Java™ virtual machine for resource-constrained wireless phones and communicator-type devices.

The first generation of Java technology-enabled wireless devices was based on the K virtual machine (KVM), a reference design that demonstrated how the CLDC specification could be implemented and was the basis for CLDC's Technology Compatibility Kit (TCK). Sun introduced the CLDC HotSpot Implementation 1.0 in mid 2002 as an optimized implementation that focuses on performance and footprint. Not only does it comply with the CLDC specification, but it also includes a number of patented features that enable faster application execution as well as more efficient resource management. In addition, it is supported on a number of targeted platforms and optimized for the ARM processor architecture.

The CLDC HotSpot Implementation delivers nearly an order of magnitude better performance than the KVM while running in a similarly small memory footprint required by resource-constrained mobile phones and personal organizers. It delivers not only better performance, but also more robustness. The CLDC HotSpot Implementation is the recommended virtual machine technology for new product deployments in this class of devices, and can be integrated with the Sun Java™ Wireless Client 1.1.2 for a full stack solution using Java technology.

The CLDC HotSpot Implementation 1.1.2 includes a number of significant new features such as ahead-of-time (AOT) compilation of Java methods, in-place execution, significant enhancements in performance, reduction of pauses due to improvements in compilation and garbage collection, multitasking capability, and full integration with Java hardware acceleration on enabled ARM processors (using Jazelle technology under license from ARM Ltd.). Each of these new features will be explained in greater detail further on in this white paper.

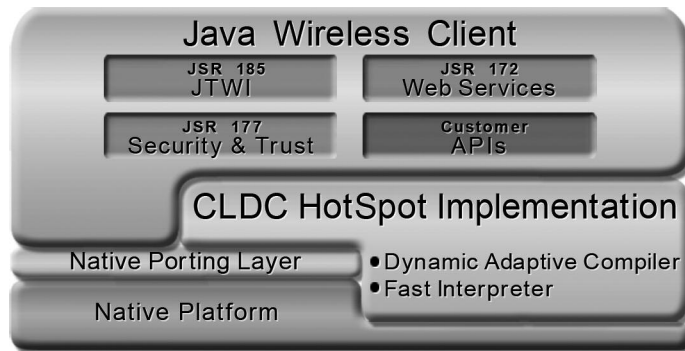


Figure 1. The CLDC HotSpot Implementation With the Sun Java Wireless Client 1.1.2

The following trends are driving the features of next-generation handsets:

- Users are more sophisticated, demanding applications such as gaming, information services, messaging, location services, and more.
- Screen sizes are larger and provide higher resolution and more colors, enabling more complex graphics such as 2D, 3D, animations, and so on.
- Networks are faster, allowing larger, more complex applications to be downloaded to handsets.
- Enterprises are deploying mobile workforce applications that require enhanced security and stability in the underlying platform.

The desire for better performance in embedded Java runtime environments drove Sun to develop the CLDC HotSpot Implementation Java virtual machine technology, with a goal of achieving:

- Faster performance
- A more robust platform
- Faster time to market

The CLDC HotSpot Implementation applies advanced tuning and performance techniques utilized in both Java 2 Platform, Standard Edition (J2SE™) and Java 2 Platform, Enterprise Edition (J2EE™) technologies, and further reduces the footprint to fit into small devices. In addition, the CLDC HotSpot Implementation incorporates several innovations in design that allow the virtual machine to run in resource-constrained devices, and enables it to:

- Provide cutting-edge performance
- Deliver fast application start-up time
- Require minimal footprint
- Reduce porting efforts
- Preserve battery life

Version 1.1.2 of the CLDC HotSpot Implementation supports both the CLDC 1.0 or CLDC 1.1 specification (it can be compiled to support one of these). The CLDC HotSpot Implementation conforms to the corresponding version of the CLDC specification and the Technology Compatibility Kit (TCK).

## Chapter 2

# Java Technology in Small Devices

Today, a complete Java technology stack exists to support embedded devices such as mobile phones. These devices are characterized as small, battery-powered devices with limited wireless connections to the Internet. The stack is based on the Java 2 Platform, Micro Edition (J2ME™) specification, and consists of the virtual machine and CLDC libraries as the foundation and the Mobile Information Device Profile (MIDP) and optional packages on top.

The J2ME specification defines configurations, profiles, and optional packages that, in combination with a Java virtual machine, make up the Java technology stack. A *configuration* of J2ME technology 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.

The J2ME platform fits in with the other editions of Java technology — the J2SE and J2EE platforms — as illustrated in Figure 2. Small, resource-constrained, battery-powered devices such as wireless phones and communicator-type devices are the domain of the CLDC and MIDP specifications, also shown in this Figure 2.

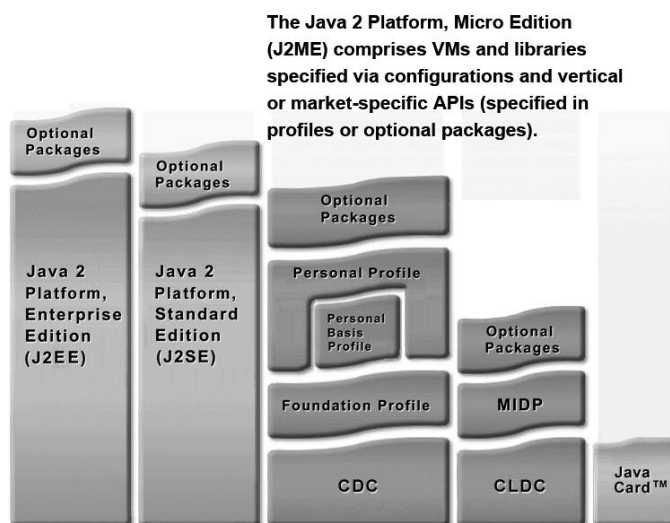


Figure 2. The J2ME, CLDC, and MIDP Specifications

## History of the Java Stack for Mobile Phones

The history of the Java stack for mobile phones really began in 2000 with the release of two reference implementations of Java technology by Sun. The first was the reference implementation of the J2ME CLDC specification. A key component of the CLDC Reference Implementation was the KVM, the first “complete” virtual machine for small, embedded devices such as mobile phones. Also in the same year, Sun released the first reference implementation of the MIDP specification. For the first time, it was possible to write useful applications in the Java programming language that could be run on small, embedded devices such as mobile phones. Thus, a revolution was born that today includes the deployment of more than 450 million Java technology-enabled mobile phones around the world.

### Connected Limited Device Configuration (CLDC)

Working through the Java Community Process<sup>SM</sup> (JCPSM) program, the CLDC configuration was created to deliver core Java library support to provide a basic application framework around the KVM. Java Specification Request (JSR) 30 — the J2ME Connected Limited Device Configuration specification — was approved in August 1999, and the final public release of the CLDC 1.0 specification occurred in May 2000. A number of major mobile phone and PDA manufacturers participated in the JCP expert group that developed CLDC 1.0.

CLDC 1.1 (JSR 139) was created to support the requirements of devices with more resources and capability, especially in the area of hardware supporting floating-point arithmetic. It was a natural evolution of CLDC 1.0. There was an even larger participation in the expert group for JSR 139, with the final public release of the CLDC 1.0 specification occurring in March 2003.

### 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 the *Java 2 Platform, Micro Edition* datasheet located at [java.sun.com/j2me/docs/j2me-ds.pdf](http://java.sun.com/j2me/docs/j2me-ds.pdf)) The MIDP specification was created through the JCP program to address the limited screen size and battery power of this class of device. JSR 37 — the Mobile Information Device Profile for the J2ME Platform — was approved in September 1999, and the final public release of the MIDP 1.0 specification occurred in September 2000.

MIDP 2.0 (JSR 118) was developed to extend and enhance the MIDP platform, especially in areas such as secure networking, support for network sockets and datagrams, support for push architecture, XML and GUI enhancements to support color, larger screens, and game technology. Expert group participation in JSR 118 was dramatically larger than for JSR 37. The final public release of the MIDP 2.0 specification occurred in November 2002.

### The HotSpot Virtual Machine

At about the same time that the first specification of the J2ME CLDC was released, product deployments began of J2SE and J2EE platform versions of a revolutionary Java virtual machine technology called the HotSpot performance engine. The HotSpot engine was developed to address the perception that Java virtual machine performance was insufficient for many mainstream applications, especially on large 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, v1.4.1 Technical White Paper* located at [java.sun.com/products/hotspot/docs/whitepaper/Java\\_Hotspot\\_v1.4.1/Java\\_HSpot\\_WP\\_v1.4.1\\_1002\\_1.html](http://java.sun.com/products/hotspot/docs/whitepaper/Java_Hotspot_v1.4.1/Java_HSpot_WP_v1.4.1_1002_1.html).)

In 2001, these two technology trends converged with the creation of the CLDC HotSpot Implementation virtual machine. In contrast to the KVM and CLDC Reference Implementation, the CLDC HotSpot Implementation is an *optimized implementation*. Creating Java technology-enabled consumer devices with the KVM and CLDC Reference Implementation was impressive, but the perception that formed in the marketplace was that here, as in conventional Java technology, there was a need for faster performance (while working within the restricted resources of the target devices). The CLDC HotSpot Implementation applies optimization techniques similar to those used in HotSpot technology (but using considerably less memory and consuming less power) to realize nearly an order of magnitude improvement in CLDC-based devices.

Version 1.1.2 of the CLDC HotSpot Implementation continues the path of technological evolution to further upgrade virtual machine performance and responsiveness in this class of devices.

### **Wireless Deployments of Java Technology**

In 2001, major manufacturers of mobile phones — such as Motorola, Nokia, Samsung, Siemens, and Sony Ericsson — and mobile operators — such as Vodafone, NTT DoCoMo, and Sprint — began high-volume shipments of Java technology-enabled phones. Other manufacturers have now entered this space, and combined shipments have accelerated nearly exponentially. Part of this acceleration is due to the adoption of advanced Java technologies such as the CLDC HotSpot Implementation.

## Chapter 3

# Demand for Performance

Many current-generation Java technology-enabled mobile phones have processor and memory requirements that are typical of the original design parameters of the KVM and CLDC. But increasingly, there are models being introduced with more computing power available. In the category of mass-produced handsets, the typical processor is a 16- or 32-bit processor with a clock speed sometimes under 50 MHz, but more often from 50 to 200 MHz. The minimum memory requirements for a target device are 300 KB of RAM and about 1 MB of flash and ROM. More typical devices increase these sizes to 600 KB of RAM and about 1.5 MB of flash and ROM.

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. With the CLDC HotSpot Implementation, Sun accelerated performance in the current generation of devices while looking ahead to emerging mobile phone designs.

In addition to performance demands, device manufacturers also demand robustness and rapid time to market. CLDC HotSpot Implementation is easy to port, which greatly enhances rapid product development.

Before finalizing the features of the 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 the minimum and typical processor and memory requirements that Java virtual machine technology must work within for next-generation mobile phones.<sup>1</sup>

Item	Minimum	Typical
CPU Type	Mostly ARM	Mostly ARM
CPU Speed	50 MHz	50 to 200 MHz
RAM	300 KB (including MIDP)	> 600 KB (including MIDP)
ROM/Flash	1 MB	> 1.5 MB

*Table 1. Next-Generation Mobile Phone Minimum Requirements*

<sup>1</sup> Sun Microsystems Customer Survey, 2001.

The ARM processor represents the majority of handset market share, and CLDC HotSpot Implementation includes numerous optimizations for that platform.

The design challenge for the development team was to make the CLDC HotSpot Implementation run within the same limits and restrictions as the KVM. It was anticipated that tuning and optimization would be required for the CLDC HotSpot Implementation to run at the minimum configurations. The typical configurations in the table would make hosting the CLDC HotSpot Implementation easier and also allow abundant room for Java technology-based applications.

## Key Requirements

The development team's 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 portable to a wide range of devices with varying capabilities.

## 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 communicator-type devices. Communicator-type devices typically have much more memory available than inexpensive mass-market handsets, however, 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.

## Chapter 4

# Design Considerations

Certain fundamental challenges need to be addressed in the design of any virtual machine technology in small, embedded devices. These challenges are also addressed in the CLDC HotSpot Implementation design:

- **The Trade-off Between Fast Execution and Small Footprint:** There is a 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 Memory Efficiency:** How can an implementation avoid memory fragmentation as stacks and heaps shrink and expand? An efficient garbage collector is a must.
- **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 the CLDC HotSpot Implementation to obtain good cache behavior so that the working set for the 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 the CLDC HotSpot Implementation directly enhances battery life. Quite simply, faster execution consumes less power.
  - **Leveraging the Advantage of Java Programming:** The implementation must execute Java language programs so efficiently that it minimizes the traditional advantages of native or low-level programming. More software, including system software, may now be written in the Java programming language.
  - **Portability:** The implementation must be relatively easy to port to a different operating system.

## Chapter 5

# 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, the CLDC HotSpot Implementation has revolutionized the deployment of Java technology in battery-powered, handheld devices.

### Performance Advantage

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, adaptive compilation
- A lightweight threading system
- Generational garbage collection
- Fast synchronization
- Unified resource management

To apply these techniques in the context of handheld devices, some very clever innovations were necessary. (See Chapter 6, *The CLDC HotSpot Implementation Architecture*.)

### Robustness and Short Time to Market

The CLDC HotSpot Implementation virtual machine addresses the customer requests listed previously, specifically:

- **Increased Robustness:** The CLDC HotSpot Implementation virtual machine can be built to comply with either the *CLDC Specification version 1.0* or *CLDC Specification version 1.1*. Approximately 10,000 TCK tests ensure compliance with these specifications. Several thousand additional tests, including virtual machine stress tests, are run on the CLDC HotSpot Implementation by Sun's Quality Engineering team.
- **Faster Time to Market:** The architecture of the CLDC HotSpot Implementation is designed for easy portability to different target operating systems. In most cases, a successful port can be accomplished within a few weeks. It is also designed to be easily integrated with a J2ME profile, such as MIDP 2.0, to implement a complete Java runtime environment.

## Scalability and Small Footprint

The CLDC HotSpot Implementation places no restrictions on the number of loaded classes or on the size of the object heap.

Despite its high performance, the CLDC HotSpot Implementation is compact enough to meet the footprint constraints of next-generation and many current-generation mobile phones. The minimum total flash and ROM memory requirement for the virtual machine and software is on the order of 1 MB. This includes the CLDC HotSpot Implementation virtual machine, CLDC class libraries, MIDP class libraries, and Java applications.

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

## The CLDC HotSpot Implementation Versus the KVM

In the KVM design, 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 the ANSI C language. However, measurements reveal that on average, interpreted virtual machine performance is approximately one order of magnitude slower than compiled virtual machine performance.

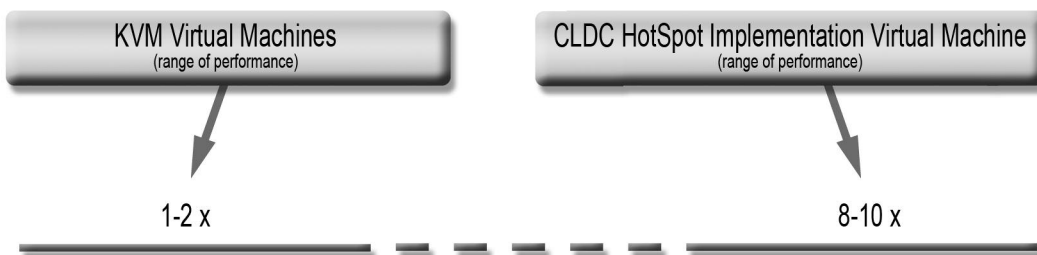


Figure 3. Performance of the KVM Compared With the CLDC HotSpot Implementation

To improve the performance of a virtual machine beyond pure interpreter performance, some type of a static or dynamic compilation strategy is needed. To approach an order-of-magnitude improvement in performance, while maintaining device-independent bytecode as the standard for applications, the CLDC HotSpot Implementation virtual machine has an innovative adaptive compiler. The adaptive compiler dynamically compiles the most frequently used, time-critical pieces of the applications into native code for significantly faster execution. The execution speed of optimized native code can be up to 50 times faster than the speed of a conventional interpreter. When run in mixed mode (using the adaptive compiler to optimize the frequently used operations and using the optimized bytecode interpreter for infrequently used code), the CLDC HotSpot Implementation system can achieve a performance advantage of approximately 8 to 10 times when compared to traditional bytecode interpreters.

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

## **Faster Execution Consumes Less Power**

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

## **Increasing Demands of Next-Generation Networks**

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

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

- Enterprise mobility applications
- 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 the CLDC HotSpot Implementation, makes an overall savings in power possible — even while servicing this new generation of software — because it finishes all tasks much sooner than a slower virtual machine.

## **Multitasking Capability**

The CLDC HotSpot Implementation 1.1.2 introduces optional multitasking — the virtual machine’s ability to run multiple MIDlets concurrently. This capability allows system software to be written as MIDlets and allows user MIDlets to interact with them — in a dynamic mix of foreground and background tasks. The implementation of multitasking in the CLDC HotSpot Implementation optimizes the use of compute resources. Multitasking capability dramatically enhances your return on investment in Java technology.

## Chapter 6

# The CLDC HotSpot Implementation Architecture

The CLDC HotSpot Implementation design team met the demands for performance and design challenges detailed in the earlier sections of this white paper. The resulting design strikes a strong balance between performance and footprint constraints.

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

- Dynamic, adaptive compiler, which compiles the most-used Java methods at runtime
- Optimized interpreter (written in assembly language)
- Support for lightweight threads, which greatly simplifies porting
- Compact object layout
- Unified resource management
- Accurate generational garbage collection
- Fast synchronization
- ROMizer, which stores system classes in a compact format that allows faster execution
- Support for CLDC 1.0 or CLDC 1.1 with full TCK compliance
- Integration of Java hardware acceleration technology, with battery savings and fast start-up
- 32-bit addressing capability to support a wide range of devices
- 16-bit Thumb mode (ARM) support
- Quick application start-up time
- Virtual machine can be built as a main program or as a subroutine in an event loop
- New features, as detailed in Chapter 7

### Dynamic, Adaptive Compiler

In general, Java virtual machines with a compiler are an order of magnitude faster than those with only an interpreter. For that reason, the 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 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, and loop peeling. It is an adaptive compiler because it reacts to data gathered at runtime to decide which methods to compile. Only the methods that execute most frequently are compiled. Once compiled, a method is subject to deoptimization if it no longer is invoked for a period of time.

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

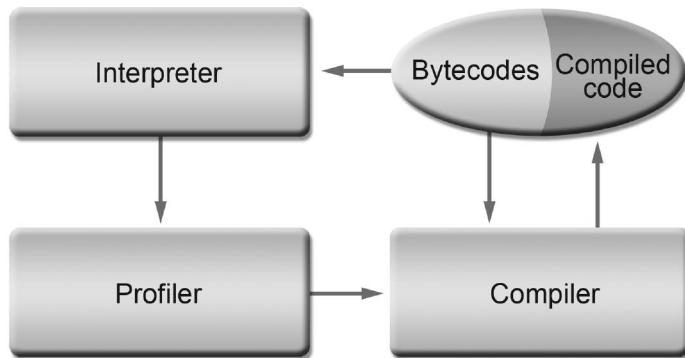


Figure 4. CLDC HotSpot Implementation Architecture

## Compact Object Layout

The 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, and 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. The CLDC HotSpot Implementation optimizes the usage of stack space. Only one word is required for the object header. In addition to reducing memory usage, object allocation becomes faster.

## Unified Resource Management

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

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

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. The 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 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. The CLDC HotSpot Implementation uses an accurate generational mark-sweep-compact garbage collector that 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 because 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 5.

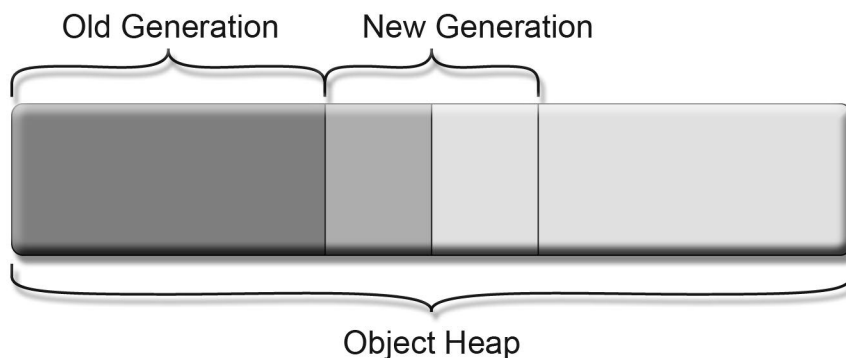


Figure 5. Two-Generational Garbage Collection

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.

## Fast Allocation

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

## Fast Thread Synchronization

The Java programming language provides language-level thread synchronization that makes it easy to express multithreaded programs with fine-grained locking. The 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.

## Lightweight Threads

The CLDC HotSpot Implementation provides a lightweight threading system, which greatly reduces complexity compared to native threading mechanisms. Porting of the CLDC HotSpot Implementation is simplified. There is also a dramatic improvement in memory usage: There is no longer a need to allocate native stack space, which must typically be of fixed size. All threads can be maintained with stacks that can be allocated in the same heap as all other objects (another example of unified resource management).

## Thumb Mode Support (ARM Processors)

The CLDC HotSpot Implementation supports ARM Thumb mode as follows:

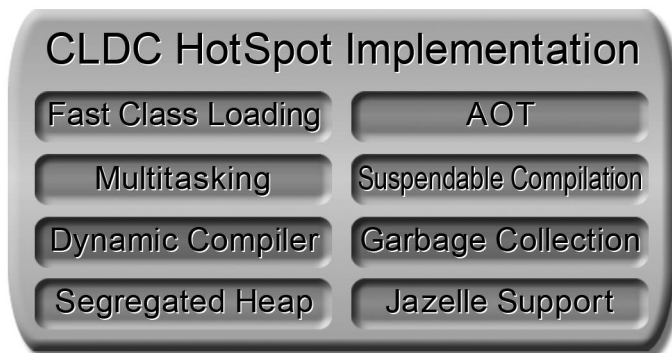
- The CLDC HotSpot Implementation virtual machine can be compiled into 16-bit Thumb machine code.
- The dynamic adaptive compiler of CLDC HotSpot Implementation can be configured to generate 16-bit Thumb code.
- The optimized interpreter of CLDC HotSpot Implementation was implemented using the 32-bit ARM instruction set. (There is no performance advantage in implementing it as 16-bit Thumb code.)

## Chapter 7

# New Features in Version 1.1.2

Version 1.1.2 of the CLDC HotSpot Implementation introduced numerous enhancements and optimizations, including:

- Ahead-of-time (AOT) compilation of Java methods
- In-place execution (formerly known as Project Monet)
- Basic virtual machine performance enhancements
- Minimal pauses due to improvements in compilation and garbage collection
- Java hardware acceleration on Jazelle-enabled ARM processors (use of this feature requires a separate license from ARM Ltd.)
- Multitasking capability — the ability to run multiple MIDlets concurrently



*Figure 6. Key New Features of the CLDC HotSpot Implementation*

## AOT Option

Ahead-of-time (AOT) compilation can be used to significantly reduce start-up times. In this release, only Java methods in system classes may be AOT-compiled. Such classes are compiled and ROMized during the build process on the development host.

## In-Place Execution

The new in-place execution feature allows handset manufacturers to transform selected Java application class files into a directly executable format known as an *application image*. An application image is loaded directly into the Java heap for execution, greatly reducing application start-up time and dramatically increasing execution speed.

## Shorter Execution Pauses

A number of new techniques have virtually eliminated noticeable pauses in execution. This is accomplished through:

- *Suspendable Compilation* — Execution continues with bytecode interpretation
- *Segregated Heap Architecture* — Separate heap areas for compiled methods
- *Improved Garbage Collection* — Each heap area is managed separately for garbage collection

## Multitasking Option

To further increase opportunities for product differentiation, this release of the CLDC HotSpot Implementation includes optional support for multitasking — running multiple MIDlets concurrently. This permits, for example, a running MIDlet to be suspended temporarily so that another MIDlet can alert the user about an incoming e-mail or instant message.

This virtual machine feature allows a whole new universe of implementation possibilities, as discussed in Chapter 8, *Multitasking in the CLDC HotSpot Implementation*.

## Chapter 8

# Multitasking in the CLDC HotSpot Implementation

With version 1.1.2 of the CLDC HotSpot Implementation, multitasking (the ability to concurrently run multiple MIDlets) was introduced. Multitasking in the virtual machine enables a much more dynamic user experience. In addition, multitasking is well-suited to mobile applications such as e-mail and instant messaging, which demand immediate user attention even when running another application. Multitasking capability dramatically enhances product differentiation opportunities for manufacturers of mass-produced, resource-constrained handsets.

### Why Multitasking Capability?

There are significant advantages of including the capability for multiple running MIDlets in next-generation handsets. The key areas of advantage include:

- **Native System-level Applications** — System-level services written as native applications are inherently non-portable and expensive to maintain. They must be rewritten for each new model of handset. It is very advantageous to develop system level services as MIDlets.  
System-level applications that typically run on today's handsets include instant messaging (IM) clients, calendar clients, call management, and the application management system (AMS), which allows users to launch other applications and MIDlets. Today, most system-level services on handsets are provided as native applications. Again, it is very advantageous to implement these system-level applications as MIDlets.
- **The Security Offered by Java Technology** — The security features of Java technology prevent MIDlets from being used as hacking tools.
- **The Portability of Java Technology** — There is a tremendous advantage, namely the high degree of portability, if the native applications can be replaced by Java applications or MIDlets.

### Providing Multitasking in the CLDC HotSpot Implementation

There are a variety of problems that were solved to permit multiple running MIDlets in limited-resource environments. These problems included:

- **Backwards Compatibility** — Due to the success of the Java 2 Platform, Micro Edition (J2ME platform) and the large number of MIDlets that are already deployed, these MIDlets must be usable in a multitasking environment without being rewritten to require new APIs. The MIDlets must be able to run without knowledge of other running MIDlets.

- **Robustness** — In prior versions of the CLDC HotSpot Implementation, a virtual machine needed to run only as long as a single MIDlet was running. The virtual machine would be restarted every time a MIDlet was launched. With the multitasking capability, the virtual machine must now run continuously, and robustness requirements must be correspondingly high.
- **OS Limitations** — Most designs for emerging handsets do not have enough processor or memory resources to run a full-scale operating system such as Linux. This would be inappropriate for products produced on a mass scale. A multitasking solution must work within the resource constraints of this class of device.

### Virtual Machine Support for Multitasking

It is the responsibility of the virtual machine to provide basic facilities and safeguards for multiple running MIDlets. It must guarantee:

- **Fair, Preemptive Scheduling** — Similar to the operating systems of large platforms, the multitasking option of the CLDC HotSpot Implementation virtual machine provides the mechanisms for scheduling of applications.
- **Working Within Resource-Constrained Environments** — The targeted platforms of the CLDC HotSpot Implementation are limited in memory and processor power.
- **Firewalled Tasks** — Tasks (MIDlets) are isolated from one another except for orderly and appropriate exchange of data.
- **No Runaway Tasks** — The virtual machine insures that no tasks can run past memory boundaries.
- **Cleanup** — The virtual machine must gracefully end each task when it completes and reclaim the system resources of the task.

### The CLDC HotSpot Implementation Provides the Multitasking Solution

Multitasking support within the virtual machine was achieved by providing task management and overall resource management similar to that provided by a large operating system.

- The CLDC HotSpot Implementation does not rely on multitasking services from an operating system.
- It implements multitasking within a resource-constrained, virtual machine implementation.
- It offers a multitasking implementation that is very robust and reliable



Figure 7. Task Management and Resource Management

The higher MIDP levels of the full technology stack can be reimplemented to take full advantage of multitasking support in the virtual machine.

## Chapter 9

# Conclusion

Sun continues to keep pace with the demands for performance of the emerging generation of mobile phones and other small wireless devices. The CLDC HotSpot Implementation continues to evolve, adding improved performance with each version. This optimized implementation of the CLDC is available to device manufacturers under license from Sun Microsystems.

© 2005 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, CA 95054 USA

All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California.

Sun, Sun Microsystems, the Sun logo, HotSpot, Java, Java Card, Java Community Process, JCP, J2EE, J2ME, J2SE, and The Network is the Computer are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a). DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS HELD TO BE LEGALLY INVALID.

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA Phone 1-650-960-1300 or 1-800-555-9SUN Web sun.com



**Sun Worldwide Sales Offices:** 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 +202-570-0442, Estonia +372-6-308-900, Finland +358-9-525-561, France +33-134-03-00-00, Germany +49-89-46008-0, Greece +30-1-618-8111, Hungary +36-1-489-8900, Iceland +354-563-3010, India-Bangalore +91-80-2298989/2295454; New Delhi +91-11-6106000; Mumbai +91-22-697-8111, Ireland +353-1-805-666, Israel +972-9-9710500, Italy +39-02-641511, Japan +81-3-5717-5000, Kazakhstan +7-3272-466774, Korea +82-2-193-5114, Latvia +371-750-3700, Lithuania +370-729-8468, Luxembourg +352-49 11 33 1, Malaysia +603-21161888, Mexico +52-5-258-6100, The Netherlands +00-31-33-45-15-000, New Zealand-Auckland +64-9-976-6800; Wellington +64-4-462-0780, 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, Saudi Arabia +9661-273-4567, Singapore +65-6438-1888, Slovak Republic +421-2-4342-9485, South Africa +27-11-256-6300, Spain +34-91-767-6000, Sweden +46-8-631-10-00, Switzerland-German +41-1-908-90-00; French +1-22-999-0444, Taiwan +886-2-8732-9933, Thailand +662-344-6888, Turkey +90-212-335-22-00, United Arab Emirates +9714-3366333, United Kingdom +44 (0)1252 420000, United States +1-800-555-9SUN or +1-650-960-1300, Venezuela +58-2-905-3800, or online at sun.com/store

**SUN**™ © 2005 Sun Microsystems, Inc. All rights reserved. Sun, Sun Microsystems, the Sun logo, HotSpot, Java, Java Card, Java Community Process, JCP, JVM, J2EE, J2ME, J2SE, and The Network is the Computer are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. Information subject to change without notice. 02/05 R1.0