SavaJe OS: Solving the problem of the Java™ Virtual Machine on Wireless Devices

Summary

Since the arrival several years ago of Sun Microsystems' Java™ technology with its concept of 'write once, run anywhere' developers have struggled to adapt Java to information appliances that have limited resources in a small footprint. The core platform for Java functionality is Java 2 Standard Edition, J2SE™, which is targeted at the desktop, operating interactively within an intranet or on the Internet. Information Appliances do not have the resources to run standard Sun Microsystems' J2SE with anything approaching acceptable performance.

Java depends upon the structure of a Virtual Machine to deliver its concept of portability. Reference or optimized VMs, associated portability layers, classes and libraries normally run on top of a native operating system with its own portability layers, APIs and libraries. This creates the performance and size issues with Java on small footprint devices.

To overcome this problem Sun Microsystems has promoted a range of subsets of Java under the overall name of the Java 2 Micro Edition, J2ME™. These subsets overcome the performance issue by restricting functionality, but in doing so remove the core Java value proposition that any Java program written for one client machine will run on another. While this restriction has not to date been too onerous, the emerging generation of smart wireless devices, advanced mobile phones, wireless PDA's, interactive web tablets and internet-enabled gaming machines require the complete application support and client functionality that 'full' Java provides. Moreover the fragmentation of Java into a variety of subsets, and individual 'flavors' from third-party providers of VM technology, has introduced global incompatibilities such that an application provided by one provider may run correctly on a wireless device from one manufacturer but not on those provided by another.

SavaJe Technologies, a 1999 spinout from Lucent Technologies' Bell Labs, applies innovative technology to overcome the issues that prevent "full" Java from running on wireless handheld devices. By delivering a "Universal Java Client" platform that runs all APIs from all subsets of J2ME as well as all APIs from J2SE, the SavaJe OS overcomes both the performance and compatibility problems. With SavaJe OS a handheld wireless device or information appliance is enabled as a full Java client in the Internet or intranet environment.

SavaJe OS is not a Virtual Machine. Designed from the ground up to run Java applications optimally, SavaJe OS is built around a 32-bit, multi-threaded, multi-tasking, pre-emptive, fit-for-purpose kernel. The SavaJe OS includes an embedded Java Virtual Machine (JVM) that is tightly coupled to the operating system kernel, and a complete set of Java libraries. The API for the SavaJe OS is the full range of J2ME and J2SE APIs. SavaJe is a Sun commercial J2SE licensee. The SavaJe OS is fully Sun compliant and is Java branded.

This White Paper discusses the architecture and capabilities of the SavaJe OS, including new functionality that will be delivered in Version 2.0 in 2002.

The Goals of SavaJe OS

The primary goals of the SavaJe OS are:

- **A Small, Efficient Kernel**
  Information appliances have less CPU power, memory, and external storage than desktop or server-class systems. As a result efficiency in the use of memory and processing power are important commodities.

- **Portable Applications**
  By supporting the full range of standard Java 2 APIs, SavaJe OS extends the full functionality of Java to the information appliance world. While there are some concerns that must still be considered when writing applications for information appliances (e.g. the user interface, memory and storage limitations, and network connectivity), application programmers now have the complete and familiar J2SE libraries available.

- **A Rich Graphical Experience**
  Applications written using SavaJe OS can take full advantage of the abstractions in the Swing and Java 2D APIs. SavaJe OS supports a wide variety of screen dimensions, and color depths up to 32-bit.

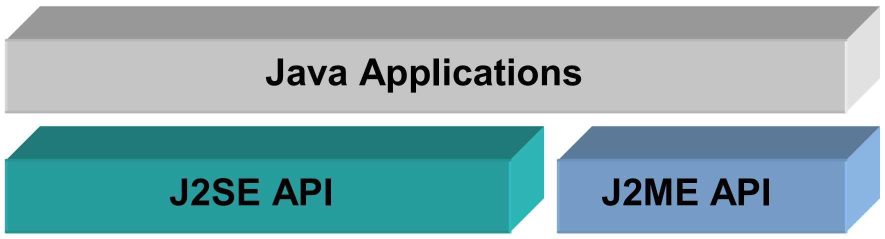
- **Strong and Flexible Security**
  SavaJe OS delivers application integrity by building directly on Java's inherent security. By default, each application runs in a secure environment that includes its own security manager controlled by a security policy.

- **Full Network Support**
  SavaJe OS is designed to provide full support for networking and distributed computing. Connectivity to corporate LANs, local wireless, cellular services, and PAN connections such as Bluetooth is built into the SavaJe OS. Robust applications can be developed with standard Java APIs including RMI, CORBA, and JINI services.

- **Performance**
  SavaJe OS is designed to provide far better applications performance in a substantially smaller memory footprint than any other PersonalJava, Personal Profile or J2SE Java implementation on a given hardware platform.

The Core Platform

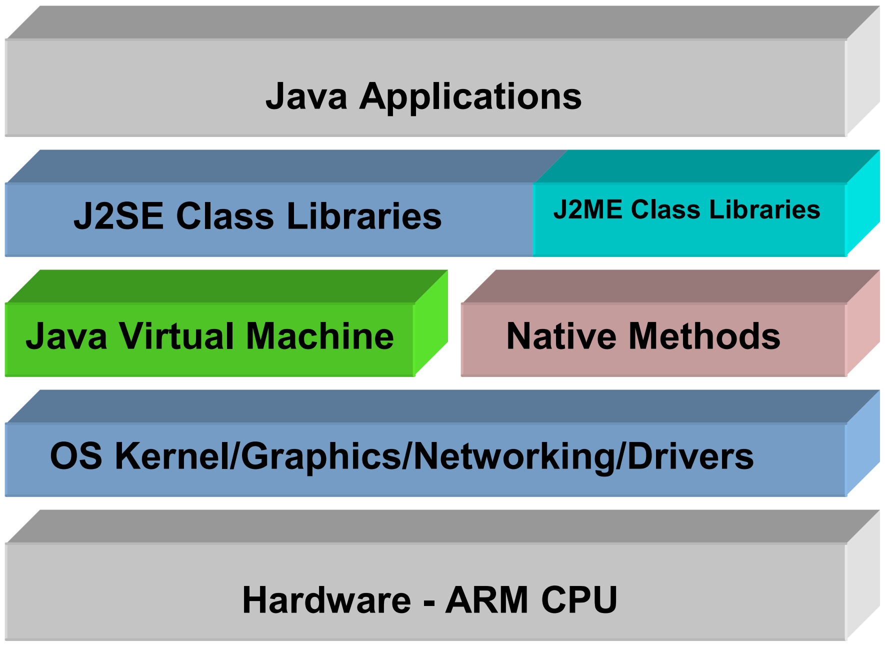
Application Programmer's View of SavaJe OS
Since SavaJe OS is designed to specifically to support Java applications, the developer's view of SavaJe OS is simple. Any SE or ME app will run on or can be developed for SavaJe.



Applications that run on SavaJe OS can be written using any one of the full range of Java APIs - from the Java 2 Micro Edition profiles, such as MIDP, MIDP 2.0 and Personal Profile, to the full rich environment of Java 2 Standard Edition. SavaJe has added optional Applications Services oriented extension Java APIs, making SavaJe OS the most complete Java client environment available.

While SavaJe OS does not support end user applications written in other languages, there are provisions to allow 3rd party native modules written in C, C++ or assembler (drivers, media codecs, or arbitrary libraries) to be integrated into the SavaJe OS. These modules can then be exposed to applications via a suitable Java API.

Architectural view of SavaJe OS
The diagram below provides a simplified view of SavaJe OS.



Note that the kernel, the graphics system, the JVM, and the Java native methods are tightly bound, allowing performance optimizations not possible in other "general-purpose" operating systems.

Kernel

The kernel of SavaJe OS provides memory management services, resource locking, thread control and scheduling, and a device driver interface. In addition, the Java Virtual Machine is closely integrated with the kernel. (Both the JVM and Java native methods can be considered as kernel-level resources).

The kernel, JVM, and device drivers are statically bound. Device drivers, written to a standard interface, allow the kernel to communicate with peripherals, including keyboards, touch screens, and communication ports.

Experience has shown that is a comparatively simple task to add existing native modules to the SavaJe OS. Developers may add native modules such as media codecs, or existing C libraries. To support this, SavaJe OS contains an extensive subset of ANSI C and POSIX APIs. Examples of 3rd party source modules that are present in the current default SavaJe OS are: the Java VM itself, the FreeType font engine, the Beatnik Audio Engine, and the ZIP compression library. Others are currently being integrated.

In some cases source code is not available when creating a SavaJe OS image. SavaJe OS is created with the industry standard GNU tool chain - compiler, linker, assembler, etc. This allows for binary code modules to be added to the SavaJe OS without access to source code. A binary created with these or similar tools can easily be integrated. In addition SavaJe OS can in some cases link in other arbitrary modules of ARM binary code.

Memory Management

To manage physical memory, SavaJe OS uses virtual addresses in a 4 GB address space. Physical memory is divided into 4 KB pages, managed through a two-level hierarchy. At the kernel level, memory is allocated through an efficient malloc() for most general uses. Some system objects such as monitors, locks and semaphores are allocated using memory blocks of fixed size.

There are no heap size settings since all memory that is not used by the system or by the DRAM file system is available to Java programs.

Real Time Response

The SavaJe OS is not a hard real time system. There are no provisions in the SavaJe OS to enforce real time deadlines that are needed for hard real time systems. Rather, the SavaJe OS was designed with the expectation of performing tasks that are time critical to the end user, but not catastrophic – for example playing media can be time critical but missing a frame is not catastrophic to the system. The focus has been on developing a system that is capable of performing these end-user oriented time critical tasks. This may be classified as a soft real time system.

One area of concern for key core modules is often that of interrupt dispatch. Measurements on commercially available hardware (206MHz SA-1110) have shown the SavaJe OS v1.0 system is capable of sustained processing of approximately 100,000 interrupts per second. This represents an approximate upper bound that will saturate the CPU. On the ARM architecture the use of FIQs would allow for more efficient interrupt dispatch. The SavaJe OS Kernel is very flexible in allowing for developers to create system threads that can isolate themselves from the Java Applications in order to perform sensitive operations. The results can then be passed up to a Java Application.

Virtual Machine

The JVM in SavaJe OS is developed from Sun Microsystems' VM, but has been optimized for size, speed, and low-memory environments. The JVM runs a highly optimized byte code interpreter producing efficient code while conserving memory.

The JVM garbage collector is a simple, efficient, mark and sweep garbage collector that uses handles for object references. The garbage collector is optimized for Java finalizers. There is a dynamically sized handle table and no object coalescing.

The SavaJe OS JVM is interpretative and does not use JIT techniques. SavaJe OS creates less memory overhead than JIT-based systems. Applications that interact with the user or network typically run up to an order of magnitude faster and with a considerably smaller memory footprint than on other Java platforms. The corollary is that intensive computational tasks written in Java can benchmark slower than on a VM with a built in JIT compiler.

The SavaJe OS thread model is implemented to directly support the Java threads API and semantics. A Java thread object is closely tied to a SavaJe OS thread. Non-Java threads are the exception in the SavaJe OS. The underlying thread scheduler implements a time-sliced pre-emptive, fixed-priority, round-robin scheduler. The highest priority ready thread always executes, and threads of equal priority execute in a round-robin fashion.

Hardware Java Accelerators and Project Monty

SavaJe OS can take advantage of any hardware acceleration available in the underlying platform. For example, SavaJe OS 2.0 can utilize the ARM Jazelle core when available to execute relevant byte codes in hardware. However, experience has shown that most "real-Java" applications (as opposed to small midlets) benefit far more from the other architectural features of the SavaJe OS, than simply from VM acceleration.

Sun Microsystems has recently announced the ongoing development of an optimized VM for J2ME CLDC, called the Project Monty VM. This is not a "full Java" VM but rather a next generation version of the original KVM for MIDP. As such this VM will help accelerate MIDP implementations where midlet performance is restricted by the KVM, but will not be usable "as is" for Personal Profile or J2SE platforms. As a Sun Licensee, SavaJe has access to the Monty VM technology and will incorporate features from Monty into SavaJe OS where benefit can be gained.

Floating Point Support

Since the SavaJe OS operates on platforms that do not necessarily include a hardware floating-point processor, a highly optimized software floating-point emulation library is included. In addition an optimized library supporting the intrinsic math functions is also provided. The SavaJe OS platform also uses integer math wherever possible to minimize performance issues commonly encountered with floating point code.

Process Model

Similar to most modern operating systems, SavaJe OS launches each application in a distinct process, referred to as a JProc (Java Process). There are a number of unique attributes in the SavaJe OS process model related to Java. By default, a new JProc has the following characteristics:

- Its own class loader
- Its own security manager
- A private view of class fields (i.e., statics)
- Its own view of Java system properties
- Its own graphics context
- System threads (for various system-level functions)

Single JVM

These characteristics are important. They guarantee that one executing application will not interfere with another. SavaJe OS provides separation between processes while at the same time maintaining a single VM instance and sharing the standard system classes (both Java standard classes and SavaJe OS system classes). This is accomplished by treating the VM as an integrated kernel-provided resource to JProcs. In a traditional VM implementation, the VM executes within an OS process context. In contrast, on SavaJe OS, the VM is a service provided to the process by the kernel. This service model is beneficial to the system in general because the VM code and data is loaded and initialized only once, during system initialization. Because SavaJe OS is an always-on operating system, system initialization is a rare occurrence.

Shared System Classes

Unique to SavaJe OS is the ability to share system classes across multiple JProcs. All JProcs follow the Java 2 Class loader delegation model. In this model, classes are loaded via a hierarchy of class loaders. Each class loader, when first asked to load a class, will attempt to delegate the task to its parent class loader. The top of the class loader hierarchy is the system class loader (also known as the null, the bootstrap, or the internal class loader). The system class loader is responsible for loading the standard Java classes and all the vendor implementation classes (i.e., the SavaJe OS system classes). Since the individual application class loader does not have to load the system classes, the cost of loading them is shared by all executing SavaJe OS applications. The result is far less memory consumed across the system, and less processing resources required to execute applications.

Security

SavaJe OS security is built upon the standard Java security architecture. The following components play a part in the SavaJe security architecture:

- Java Language
- Java byte-code verification
- Java Security Managers
- Java Runtime
- Java Jar packaging
- Java Security Architecture
- SavaJe process model
- SavaJe user identity

The Java language contributes to security by providing a type safe language that prevents accidental or intentional access to arbitrary memory locations. In addition, the Java VM enforces language via a process known as byte-code verification. The Java Security Manager infrastructure and its consistent use by the Java Runtime are key components. A policy file that is supplied when the program is executed drives the security manager.

Under SavaJe OS the system will start each process with a security manager installed. This mandates that a program be given a policy file. This forced creation of a security manager is a unique feature of the SavaJe OS, providing a stricter security environment than that of a conventional desktop Java application environment.

In general, applications should be run with the security manager and given the desired set of permissions. However, under special circumstances (following an explicit action by the programs installer) SavaJe OS makes it possible to start applications without creating a security manager. This is to cater for applications that must load their own security manager in order to execute properly.

When a SavaJe OS device is in operation it has a single user identity. A device can support multiple user profiles but only one logged-in user at a time. The method by which this user identity is established is specific to the device. Examples include a user-entered password or a user supplied smart card.

The Java Security Architecture provides a number of tools to allow for the secure delivery of code to a device over an insecure network (e.g. mobile phone networks). The Java application execution model allows for Java Archives (JAR) files, which contain application code and data resources, to be digitally signed. The Java execution platform then will verify the originator of each JAR prior to executing the code. In addition the Java Cryptography Architecture (JCA), Java Cryptography Extension (JCE), and Java Secure Socket Extension (JSSE) provide APIs for implementing additional security. These APIs can allow for secure network connections and data encryption. This lays the foundation for implementing a very secure application delivery mechanism. In a wireless network this can be easily integrated with the carrier and/or service provider OTA delivery strategy.

Graphics Subsystem

The graphics subsystem is a key component of the SavaJe OS platform, playing a significant role in terms of performance and features. The target devices for the SavaJe OS platform are personal hand-held user-oriented devices with compelling graphical interfaces. The performance of a graphics subsystem is a function both of user perceptions and expectations. The interface must be reactive to the user's actions and provide good performance. Key goals of the SavaJe OS include:

- Immediate reaction to user input
- Optimized memory utilization
- Reduced CPU loading
- Close integration with Java 2 AWT/Swing
- Minimal software layers
- Support for a variety of hardware types

The SavaJe OS platform supports the high-level APIs of Java 2 AWT, Swing and Java2D.

The graphics subsystem is tightly integrated to the AWT semantics and object structure. This integration allows for many optimizations in both memory usage and processing. For example, many of the internal levels of indirection that are found in an AWT implementation on other platforms are not required in the SavaJe OS system. Information stored in many AWT objects is directly manipulated by the core graphics subsystem, rather than copied and/or translated between un-related software layers. This has the dual advantage of reducing the memory requirements of a high-level graphics component as well as cutting the processing time required to manipulate the component.

The SavaJe OS platform is targeted at information appliances with varying screen sizes, levels of color support, types of graphics hardware, and distinct input methods. The graphics subsystem was designed to support a wide range of these hardware variations. By way of an example, while it contains low-level graphics drawing primitives executed in software, it can substitute any available hardware-acceleration capabilities that are available. This has an additional advantage for developers - enabling a prototype graphics output device to be developed quickly, followed by incremental addition of hardware acceleration, a key benefit where time to market is critical.

Multimedia

Applications are able to access various media types via standard Java APIs (e.g. MIDP Multimedia Extensions (JSR-135) or the Java Media Framework (JMF) for audio/video). New media types may be added through protocol specific embedded Codecs. SavaJe OS supports

- Images: JPEG, GIF, PNG
- Audio: WAV, AU, MP3, MIDI
- Video: MPEG4 with audio stream (SavaJe OS 2.0)

The SavaJe OS allows media codecs implemented as native modules. Interaction with the controlling application is via the standard Java APIs, but the kernel can schedule codec threads in isolation from Java application threads. In addition the codec threads can interface directly to the graphics sub-system. This allows for efficient and timely processing of the media stream, even in the presence of Java application garbage collection cycles.

The media support framework within the SavaJe OS provides the following:

- Support for multiple media codecs
- Support for multiple media formats
- Buffering of media data (streaming or local access)
- Buffering of decoded video frames
- Intelligent frame dropping and re-sync
- Interface to the graphics subsystem
- Interface to the SavaJe OS audio mixer

File Systems

A wide variety of file systems are supported. The goals for the file system architecture were the following:

- All file systems accessed via the standard Java APIs
- Dynamically-mountable file systems
- File system implementation in Java or C
- Support for removable media
- Support for network file systems

The architecture consists of a core infrastructure implemented in Java as a high-level service. File systems can be dynamically mounted into a single namespace using a simple standard convention. File systems can be implemented as core kernel C modules within a simple Java wrapper, or entirely in Java.

SavaJe OS 1.1 contains five distinct types of file systems.

- A recoverable read-write RAM file system
- A read-only flash based file system (SavaJe OS 2.0 also includes Read-Write)
- A Microsoft-compliant FAT file system for removable compact-flash cards including disk drives
- Two network file systems – SMB, and a simple Web server based file system

The mounting convention is that all file systems are mounted at a root ("/") with a file system type specific name. Once mounted, a file can be universally accessed via a single namespace – for example: "/cf0/foo/bar.html". This namespace has the appearance of a UNIX-style file system, with properties similar to Microsoft UNC.

Connectivity
The SavaJe OS supports various connectivity options including short-range peripheral access, synchronization to PC or corporate server and Internet access. SavaJe OS supports a variety of protocols for the following connectivity solutions:

- Ethernet
- 802.11 Wireless LAN
- Packet Switched wireless networks: GPRS, CDMA 1xRTT, EDGE, UMTS
- Bluetooth
- IrDA
- USB
- Serial

Network Access
SavaJe OS provides support for standard protocols over IP networks. An IP network may consist of a home-network, corporate intranet, the Internet or a Peer to Peer connection. Connectivity can occur over a wired Ethernet, a wireless 802.11 Ethernet, or a PPP Link. A PPP link can be established over a standard raw-serial link, IrDA, Bluetooth, or USB connections. Alternatively a network connection can be established over the various cellular wireless technologies such as CDMA, GPRS or UMTS.

Java APIs are used to access the network. In general uses the standard Java Socket APIs. Additional industry defined specific Java APIs are supported, e.g. JSR-82 for Bluetooth.

TCP/IP Stack
In the SavaJe OS the TCP stack is an efficient implementation optimized for client use. The expectation is that a SavaJe-driven device is a personal, handheld unit. This does not preclude it from functioning as a server, but recognizes the fact that the target device is an end-user node not a large server-oriented appliance. This allows for various size/speed optimizations to be performed, resulting in a reliable, very efficient network access layer.

Application protocols
On the SavaJe OS a Java application can make use of raw TCP or UDP sockets to implement its own application level protocols. In addition application level protocols such as HTTP, POP, SMTP, etc can be utilized. For advanced, distributed applications SavaJe OS provides the Java APIs for RMI, CORBA, JNDI, JDBC and JINI. This full range of application support allows SavaJe OS to function as a powerful node on the network.

PAN Support
SavaJe OS 2.0 supports Personal Area Networking (PAN) via Bluetooth and IRDA. PAN is useful for quick short-range information transfer between two devices. Support for application registration and discovery is provided by the SavaJe OS. Applications can then use standard Java APIs to utilize OBEX, vCard and vCal as well as connections to devices such as headsets or PDA/laptops.

Data Synchronization
SavaJe OS 2.0 includes full support for industry standard SyncML client data synchronization. Various object types including vCard and iCal are used for synchronization to the built in PIM functions. This enables synchronization to a wide variety of industry standard server solutions. See www.syncml.org for further information.

Services
SavaJe OS 2.0 provides a services architecture that offers infrastructure for a number of core services. A service is a high-level system component that contains no user interface and may provide functionality to one or more user applications and/or other system components. The service framework is a simple, flexible message routing mechanism. Certain services are created at system startup and are critical to proper system behavior. Others are started on demand when needed. The following are some examples.

System Events
In a modern OS there are a number of operations that periodically occur which are of interest to higher-level components or to end-user applications. These operations are presented as events. System components or applications can register interest in various events and are notified when they occur. The event system is flexible and can be extended. The following are examples of event types.

- Dynamic device introduction/removal (i.e. CF Card insertion)
- Network connection
- Battery status (low/charging)

Document Manager
A document manager service presents applications with a simple access mechanism for all available files over a number of installed file systems. Within each file system the document manager will maintain a set of *managed files.* Files will be managed based on type. This frees applications from traversing multiple file systems. In addition it can hide the various file systems and their directory structure from the end user. When new file systems are mounted the document manager automatically discovers any previous managed files and adds them to its working set.

Messaging
Applications can make use of a messaging service that handles the low level details of delivering various forms of user level messages – email, SMS, MMS and IM. This facility enables applications to be developed with messaging features without concern for the lower level detailed functionality. For example a browser may have a send Viewed Page feature, yet an email client contains message creation, send and view features. Both applications can utilize the messaging service. This encourages developer creativity in the application GUI without any need to modify the core messaging functionality.

User Alert
There are times when the system, system components, or applications need a simple and consistent way to provide information to the end user. The Info Service answers this requirement. According to the circumstances this information may interrupt the user's

current activity, may require immediate response, or may be ignored by the user. The OS service provides this functionality without requiring the information generator to be aware of the exact presentation of the information for any given device.

User Interface

The user interface of a device is critical to the successful adoption of a wireless device. The SavaJe OS is designed to provide a flexible infrastructure upon which device makers can provide their own value-add interfaces. SavaJe OS can be shipped with:

- Device Specific/Custom UI
- SavaJe SmartPhone Reference UI
- SavaJe Pen Reference UI
- SavaJe Web Terminal Reference UI

The SavaJe OS can be simply configured to use a UI that replaces any of the reference SavaJe provided UI's. This allows a customer to take full control of the UI specification and implementation. All SavaJe-provided Reference UIs contain features that allow for OEM/Operator/end-user customization. Such customizations can be end-user themes or skins, operator fixed or OTA-delivered branding, or OEM branding.

The SmartPhone UI is appropriate for advanced wireless handsets that include numeric keypad, color screens (typically 208x176 to 320x240 pixel resolution), and no pen/mouse. The Pen UI is appropriate for pen-based tablet devices (typically 320x240 to 640x480) with hardware or software keyboards and handwriting recognition. Finally, the Web Terminal UI is intended for a large screen device (typically 640x480 to 1280x1024) with a keyboard and pointing device (pen, mouse, or a combination). It is also possible for a given device to support multiple UIs that are available in certain configurations.

Each UI is composed of a GUI presentation layer and interface to a common core engine. The core engine handles the tasks of managing installed applications: - what applications can be started, where the applications reside, what icons are associated with the applications, and how to launch the applications. This allows GUI presentation layer designer to be concerned with the details of how to present the information to the user and not with the mechanics of actually installing or launching the applications. The core engine is based on the Java JNLP specification and is compatible with both J2ME MIDP 2.0 OTA specifications, as well as the J2SE paradigm for client applications management. This meets the requirements for OTA application provisioning specified by individual wireless carriers. In addition, the core GUI engine also manages locally installed applications.

OS Upgrade

In addition to Applications Provisioning and Management SavaJe OS 2.0 includes capabilities to patch the OS from a connected server. This enables OTA upgrades and patches to be delivered, securely, to customers without the requirement for expensive handset recalls or upgrades. In addition, complete OS replacement can be carried out through a high speed network connection or using a memory card (if available).