

A large, abstract, light gray graphic on the left side of the page, consisting of several overlapping, curved shapes that create a sense of depth and movement.

The CDC **APPLICATION MANAGEMENT SYSTEM**

Java™ Platform, Micro Edition
White Paper
June 2005

Table of Contents

Introduction	3
Operating Systems Background	4
Enter Java Technology	5
Process-Based Application Management	6
Native Integration Does Not Mean Native Replacement	8
Comparison with CLDC MVM	9
Conclusion	10

Chapter 1

Introduction

The more we use our devices, the more tasks we give them. We then expect these tasks to be performed seamlessly, and we prefer to avoid learning about which device feature is available at a given moment because we really want them all. Concurrency is the ability to perform multiple tasks using a shared resource. For devices, doing this well has benefits that include operational efficiency, reliability, and an improved user experience.

This white paper describes a new application management system (AMS) for the Connected Device Configuration (CDC) Java™ runtime environment that can concurrently launch and manage multiple applications. The goals for the CDC AMS are:

- *Robustness*. Real-world applications must often share CPU time and resources with other applications. Yet it is difficult for the designer of one application to anticipate the behavior of another application. So operating system software must prevent the behavior of one application from affecting another. One example of this principle is fault containment, where a buggy or rogue application is prevented from accessing the resources of another application. Another example is application termination, where an application's resources are released, so they can be easily reallocated to other applications.
- *Scalability*. An AMS system should provide quick application launching, as well as resource sharing and control. System overhead for application management should be minimized.
- *Full native integration*. An AMS should recognize that existing native applications and services represent significant investments by everyone in the device ecosystem: users, OEMs and service providers.
- *Full compatibility with existing APIs*. Backward compatibility is important because it protects the value of software investments in existing applications. Therefore, exposing new APIs or coding guidelines to application developers should be avoided, because this may require modification of existing software.
- *Transparency*. Users choose when to launch, terminate, download, and provision applications. Within reason, users should not be affected by the details of how an AMS is implemented.

To achieve these goals, the CDC AMS is based on the principal of *application isolation*, which is the ability to encapsulate an application to simplify its management by system software. Isolation lays the groundwork for robustness and scalability. The CDC AMS is a distributed system of native OS processes that each contain a logical VM instance or native application. The key principle is that native OS processes can provide complete and robust application isolation that does not need to be duplicated at the Java platform level.

Chapter 2

Operating Systems Background

The building blocks of a modern operating system, such as Linux, are based on five principles for organizing the execution of different application programs:

- A *process* is an abstraction of a running program that executes on a single CPU. It can be represented with the following:
 - process priority
 - CPU state
 - virtual address space

Native processes provide an excellent isolation boundary mechanism, because they have good fault containment and often use hardware-assisted memory protection.

- Process management is the organization and control of multiple tasks on a computer based on the following activities:
 - creation and termination
 - scheduling
 - resource management
 - interprocess communication
- A *thread* is a simplified execution environment operating within a process. It can be represented with the following:
 - thread state
 - thread priority
 - CPU state

Threads within a single process share the same address space. This greatly reduces their resource requirements, and streamlines the task of communicating between threads.

- *Thread management* is the organization of multiple threads of control within a process. Since threads share the same address space, thread-safe programming guidelines are necessary to avoid resource contention issues. Within applications, thread libraries can organize asynchronous activities like event queues.
- *Concurrency* is the operation of multiple processes or threads. Because they can execute in parallel, processes and threads may compete with their peers for processor time or other resources. And because they may have related purposes, both processes and threads may need to communicate and share data with their peers.

Conventional operating system architectures emphasize process management to promote concurrency for multiple heterogeneous applications. Since an application does not know who its neighbors will be, reliable process management protects one application from the transgressions of others.

Chapter 3

Enter Java Technology

The Java programming language is one of the first to include thread support as an integral part of its design. Developers can organize their applications as collections of objects that more naturally represent real-world data sets and workflows. These objects can then use multiple threads of execution to operate and communicate asynchronously.

The Java programming language and the associated Java virtual machine also include a namespace mechanism that protects objects from unauthorized access. Objects outside of a namespace of an application's classloader are not allowed to access the object's otherwise public methods and fields.

Many early uses of Java technology were monolithic `main()`-based applications. But from the very beginning, Java technology included examples of managed application frameworks that allowed a single Java runtime environment to launch, run, and manage several applications at once. For the Java Platform, Standard Edition (Java SE) technology, the most successful example has been applets. An applet is a dynamically loaded Java application that operates within an applet manager, usually a Java runtime environment embedded in a Web browser. The applet manager can launch and manage multiple applets. Since these applets can literally come from anywhere on the Web, the applet application model includes a default security model that prevents applets from manipulating local data or using local interfaces.

CDC includes the newer `xlet`-managed application model. `Xlets` are similar to applets in their purpose, but different in their interfaces and implementation. Their greater flexibility is based on a streamlined life cycle model and support for both GUI and non-GUI application scenarios.

Both of these managed application frameworks use thread-based concurrency and namespace-based isolation. This has many advantages, but applications can still interfere with each other in subtle ways and so are not fully isolated. Namespace-based isolation provides security without robustness.

New Requirements

ClassLoader-based isolation works for well-behaved applications, but cannot provide the robustness needed in an arena filled with applications from different vendors and built for different purposes. Adding native applications and service to this mix only makes the problem more complex.

Chapter 4

Process-Based Application Management

If process management works well for operating systems and native application management, then can it help provide reliable application management for the Java runtime environment? The CDC AMS is an attempt to fuse these two approaches.

Figure 1 shows how the CDC AMS uses a single native process to provide an isolated application context that contains an instance of a Java runtime environment for running a single Java application. It could be a standalone Java application or a managed Java application, such as an xlet or a MIDlet suite. The life cycle of the Java application is based on the native process that provides its application context. When the native process is killed, the Java runtime environment and its application are reliably and cleanly terminated, and its resources are returned to the OS for reuse.

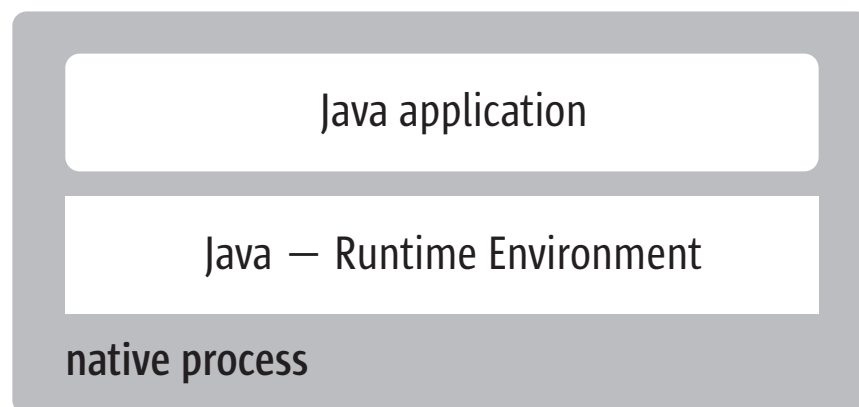


Figure 1. Isolated Application Context

By itself, this is not different from the way in which a Java runtime environment is launched by a native application manager. The real value is apparent when this scheme is used to run multiple Java applications that cooperate with each other and with other native applications. Figure 2 shows how an application manager can control multiple Java applications and native applications.

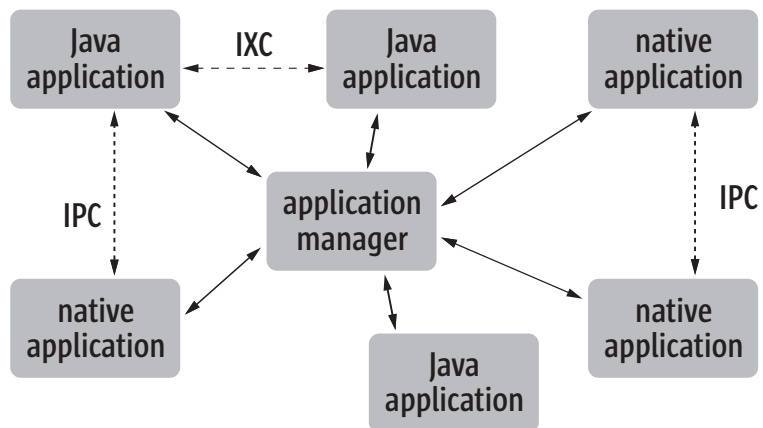


Figure 2. Application Management

From a developer's perspective, using the CDC AMS does not require learning any extra APIs or programming guidelines. The isolation and robustness of native processes already provide these benefits.

Applications can also communicate with their peers with different interapplication communication protocols, for example Inter-xlet Communication (IXC), Remote Method Invocation (RMI) or some other native interapplication communication protocol.

The application manager itself can be structured as either a native application manager or with a pure Java implementation. What matters is that it can control the life cycles of native and Java applications, and can be implemented with either native or Java technology.

Chapter 5

Native Integration Does Not Mean Native Replacement

The CDC AMS is, above all, a native integration technology that allows multiple native and Java applications to run concurrently using mature, native OS technology. It does not require replacement of native applications, services, or AMSs, and thus protects investments made in native technology.

Java technology has always shown the ability to integrate native services within the functionality of the Java class library. For example, the Java security framework has service provider interfaces (SPIs) for integrating different platform-based security provider implementations. The Content Handler API (JSR-211) provides a mechanism for registering different platform-based content handlers based on their MIME types.

At the GUI level, the Abstract Windowing Toolkit (AWT) and Swing were designed around the need to coexist with native GUI frameworks. This includes matching the appearance of native GUI components, matching their interaction characteristics, and coordinating the sharing of data with cut-and-paste and drag-and-drop techniques.

The CDC AMS extends native integration into the realm of application management. The key architectural feature of the CDC AMS is a life cycle management capability that can be implemented with native or Java technology. The CDC AMS can then manage a heterogeneous mix of Java applications and native applications.

Chapter 6

Comparison with CLDC MVM

The Connected Limited Device Configuration (CLDC) Multitasking Virtual Machine (MVM) (see java.sun.com/j2me/docs/pdf/CLDC_mvm.pdf) provides an alternate application management solution for platforms with more limited operating systems. It basically answers the question, “What if the underlying operating system does not have the process management facilities needed to support the application isolation model described in this white paper?” In this case, CLDC MVM takes on application management responsibilities that the operating system cannot provide. It does so by creating multiple application isolation contexts within a single Java virtual machine instance.

In contrast, the CDC AMS is appropriate for more capable operating systems, such as Linux, that provide fast and robust process management. By leveraging mature native process management, the CDC AMS avoids modification of the CDC class library or applications, which would be difficult given their size.

Chapter 7

Conclusion

Robustness, compatibility, and native integration are the touchstones of the Java platform. The CDC AMS is robust and scalable without creating compatibility hurdles for existing applications. By seamlessly integrating Java applications with native applications and services, the CDC AMS provides a convenient user experience that promotes productivity.

© 2005 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054 U.S.A. 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. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, and Java are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. 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.

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 ARE HELD TO BE LEGALLY INVALID.