Development Platforms for Mobile Applications: Status and Trends

Damianos Gavalas and Daphne Economou, University of the Aegean

// A comparison of four popular runtime environments clarifies the options available for developing applications that run on resource-constrained mobile devices. //

MOBILE DEVICES HAVE STEADILY gained acceptance as a multimedia platform. Current tools offer application developers options to use various technologies—for example, Java, Open C, Python, Flash Lite, XHTML/CSS, JavaScript, and Mobile Ajax—to implement highly functional mobile applications. Content developers can work with audio, video, multimedia messaging, and Flash to create rich and compelling mobile content.

Although the choice of development platform is largely market-driven, it also depends on the characteristics of available platforms and the requirements of particular applications. To illuminate the status and trends in current development platforms, we reviewed and compared four popular mobile-application runtime environments with respect to various quantitative and qualitative criteria. We based the comparisons on data from earlier research,[1] technical specifications, white papers, and an informal survey of 32 mobile-application developers with hands-on experience using the platforms we reviewed. Most importantly, we developed a simple game application and implemented it on all four platforms as a case study for highlighting the main characteristics and relative merits and shortcomings of the four platforms.

We describe the general results of this comparison as well as details from the game application's development. We summarize the results from all sources in a table and conclude with our assessment of how appropriate the different platforms are with respect to critical application-development requirements.

Four Mobile-Application Platforms

Numerous development platforms are available for handheld devices, including native environments such as the Symbian, OpenC, iPhone, and Palm operating systems; Web runtimes such as widgets; and runtime environments such as Python, Lazarus, Brew, and the four we review here—Java Mobile Edition (ME), .NET Compact Framework (CF), Flash Lite, and Android—which currently enjoy the largest developer and deployment bases. Figure 1 summarizes the software stacks for these four platforms.

Java ME

This subset of the Java platform provides a certified collection of Java APIs for developing software for resource-constrained devices such as cell phones, PDAs, and set-top boxes (http://java.sun.com/javame).

Features. Java ME runs on top of a kernel-based virtual machine (KVM), which allows reasonable, but not complete, access to the underlying device's functionality. Java ME supports cross-platform development through configurations and profiles:

- A *configuration* defines the minimum Java VM features and library set for a horizontal family of devices—that is, devices with similar processing and memory limitations, user interface requirements, and connection capabilities.
- A *profile* comprises libraries specialized in the unique characteristics of a particular device class.

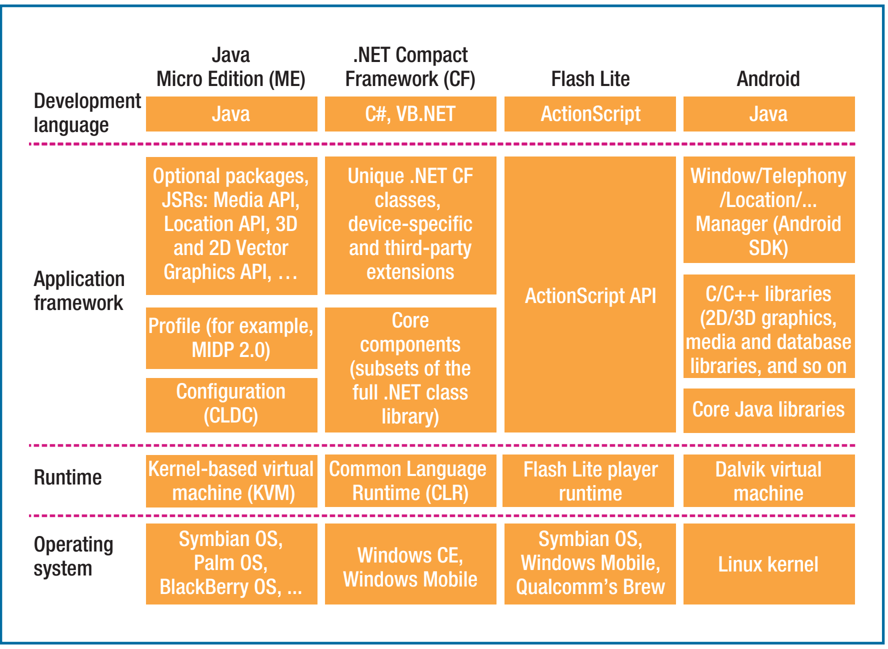
All Java ME-enabled mobile devices currently support the following configuration and profile specifications:

- *Connected Limited Device Configuration* (CLDC), a framework for Java ME applications targeting resource-constrained devices. CLDC contains a strict subset of the Java class libraries needed for mobile-application development.
- *Mobile Information Device Profile* (MIDP), a specification for using Java on embedded devices such as mobile phones and PDAs. MIDP is part of the Java ME platform and sits on top of CLDC.

Java applications developed over CLDC/MIDP are referred to as MIDlets and are typically packaged in Java archive (JAR) files.

Java ME is designed to be cross-platform, so specification and implementation are two separate processes. The Java Community Process (JCP) refers to a formalized specification process that lets interested parties get involved in defining Java platform versions. JCP uses Java Specification Requests (JSRs) to document proposed additions to the Java platform. A committee of mobile-solution providers specifies a new Java ME standard API as a final JSR that includes source code for a reference implementation of the technology. Vendors are then free to develop their own implementations.

FIGURE 1. Software stacks for the reviewed mobile-application development platforms. Comparison of operating systems, runtimes, application frameworks, and development languages.


Java Micro Edition (ME)
.NET Compact Framework (CF)
Flash Lite
Android
Development language
Java
C#, VB.NET
ActionScript
Java
Application framework
Optional packages, JSRs: Media API, Location API, 3D and 2D Vector Graphics API, …
Unique .NET CF classes, device-specific and third-party extensions
ActionScript API
Window/Telephony /Location/... Manager (Android SDK)
Profile (for example, MIDP 2.0)
Core components (subsets of the full .NET class library)
C/C++ libraries (2D/3D graphics, media and database libraries, and so on
Configuration (CLDC)
Core Java libraries
Runtime
Kernel-based virtual machine (KVM)
Common Language Runtime (CLR)
Flash Lite player runtime
Dalvik virtual machine
Operating system
Symbian OS, Palm OS, BlackBerry OS, ...
Windows CE, Windows Mobile
Symbian OS, Windows Mobile, Qualcomm's Brew
Linux kernel

Review. Java ME is the dominant mobile-software platform with respect to its installation and developer base. However, the Java language's "write once, run anywhere" axiom doesn't apply to Java ME.[2] Developers must provide slightly different application versions to address variations in JSR sets and implementations across a wide range of device capabilities and choice of profiles, configurations, and APIs. This requirement often results in dozens of executables for a given title—a phenomenon referred to as *device fragmentation*, which considerably increases operational costs over a product's life cycle. Fragmentation restricts the devices that Java ME applications can reach and suggests that it's more suitable in vertical applications that target devices with similar capabilities and Java API support.

Nevertheless, by targeting individual operating systems, developers using Java ME have access to a large set of well-defined and mature JSRs. Java applications targeting the Symbian platform, for example, can reach about 70 percent of the world's smartphones. More than 80 JSRs provide MIDlet developers a rich set of additional technologies, although MIDlet programming isn't straightforward and requires serious Java development skills.

Commonly available JSRs that extend MIDP 2.0 on the Symbian platform include the Bluetooth API (JSR 82), the Wireless Messaging API (JSR 205), and the Mobile 3D Graphics API (JSR 184).

.NET CF

Designed for applications on Windows Mobile, .NET CF is a subset of Microsoft's full .NET platform.[3] .NET CF preloads the Common Language Runtime (CLR) engine in the device's memory to facilitate mobile-application deployment. CLR provides interoperability with the underlying device's operating system, allowing the integration of native components into mobile applications.

Features. In principle, the .NET CF runtime is analogous to the Java virtual machine (JVM). Instead of writing native code for the underlying operating system, .NET developers write managed code, which targets a managed execution environment. Microsoft originally designed and developed the .NET platform with support for multiple languages and operating systems, aiming to reach an extended developer base and to reuse existing libraries. However, the .NET CF development tool, Visual Studio (VS.NET), currently supports only two major .NET languages: C# and Visual Basic (VB.NET). Furthermore, it restricts operating system support to Windows platforms, which represent only a small part of today's mobile -device products.

The core components are a subset of the full .NET framework—roughly 30 percent of its classes and functionality. Some classes exist in both .NET and .NET CF, but the .NET CF version doesn't necessarily support all the full version's class members (properties, methods, or events). Many classes aren't implemented at all, and others are only partially implemented. Unique .NET CF classes address device-specific and third-party extensions.

The .NET CF user interface design is based on a rich subset of .NET Windows Forms.

Review. .NET CF is comparable to Java ME with respect to providing a managed runtime environment, rich libraries and components for reuse (advanced user interface components, network connectivity, data management, XML Web services, and so on), and familiar APIs from the full .NET framework, such as the Windows Forms controls. These features ease the transition for desktop developers to mobile applications.

Using a runtime system for intermediate (managed) code implies relatively low execution performance. However, unlike Java ME, .NET CF is language-agnostic and simply specifies Common Intermediate Language (CIL) instructions. .NET-supported languages compile to the same CIL, so the .NET CF runtime can execute them.

UNLIKE JAVA ME, .NET CF IS LANGUAGE-AGNOSTIC AND SIMPLY SPECIFIES CIL INSTRUCTIONS.

.NET CF demonstrates API-level consistency and compatibility with the full .NET platform. This design approach has had unforeseen memory footprint costs, but .NET CF nevertheless represents a fast-paced implementation driven by a powerful vendor. Developers know the hardware specifications to program against and can assume the availability of certain native software, such as Windows Media Player. It therefore offers satisfactory integration with device-specific functionality—telephony, short-message service, subscriber-identity-module card access, Bluetooth, and so on—and doesn't exhibit Java ME's fragmentation problem. On the other hand, .NET CF targets a limited set of Windows end devices, and the VS.NET development tools include license costs.

Adobe Flash Lite

Flash Lite (www.adobe.com/products/flashlite) is a proprietary technology, popular as a multimedia and game programming platform. Adobe created it specifically to help vendors rapidly deploy rich content and interactive interfaces to mobile devices.

Features. A Flash Lite application stores its contents and GUI description in the vector-based SWF graphics and animation format. It implements its application and presentation logic in ActionScript.

The number of original equipment manufacturer (OEM), operator, and developer adopters of Flash Lite is increasing rapidly worldwide. Flash Lite 1.1 supports Flash 4 and ActionScript 1.0. Flash Lite 2.0 and 3.1—based on Flash Player 7 and 9, respectively—support ActionScript 2.0. No support is yet available for Flash Player 9-compatible content based on ActionScript 3.0.

All versions support the World Wide Web Consortium (W3C) Tiny standard,[4] a mobile profile of W3C's scalable vector graphics (SVG) recommendation.

Review. The Flash Lite platform is a reasonable choice for graphics-intensive phone and PDA applications. Industry adoption has increased because developers skilled in Flash for desktop applications can easily switch to Flash Lite for mobile applications. Rapid development is a primary benefit of Flash Lite. It's easy to learn and easy to migrate Flash applications, and it includes a rich set of designer/developer tools. Additionally, it offers rich media support (images, video, sound, and animation), a relatively broad runtime installation base, and small deployment files based on vector graphics. As of Flash Lite 2.x, it supports compressed SWF, and Flash Lite 3.0 adds support for the popular native Flash video (FLV).

Currently, Flash Lite is mostly suitable for creating animations, casual

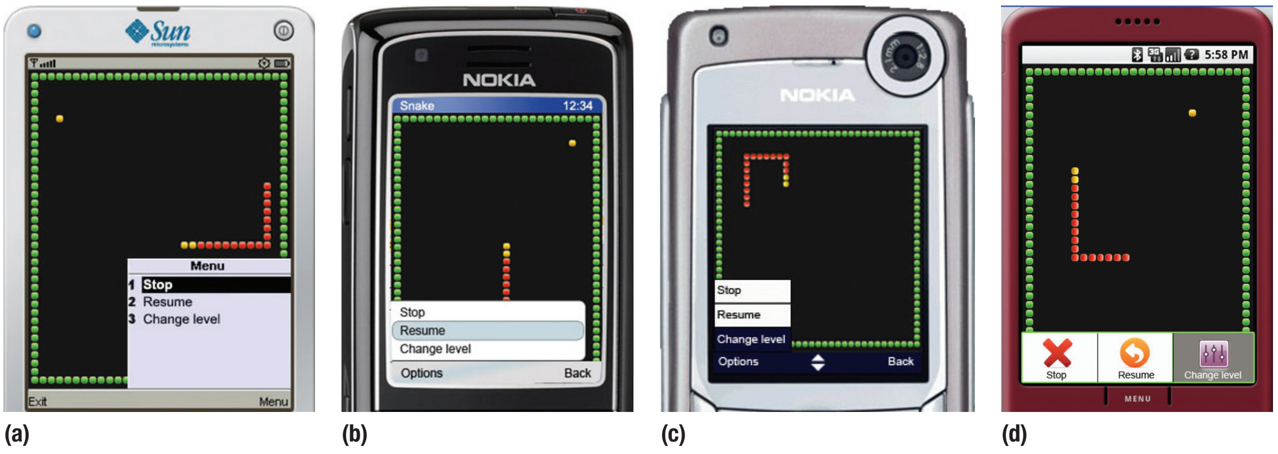


FIGURE 2. Screenshots related to the Snake game implementation developed in (a) Java ME, (b) .NET CF, (c) Flash Lite, and (d) Android. See Table 1 for comparison data on the game implementation.

games, mobile Web-based Flash applications, front-end interfaces, and device-specific content (wallpapers, screen savers, and so on). However, it isn't suitable for developing full-fledged stand-alone applications, mainly because it lacks the powerful mobile-oriented APIs of the Java ME platform.

Flash Lite exhibits relatively poor graphics performance, partly because of the complex processing required for vector graphics. It ships with an extensive toolset (Adobe CS5, Adobe Device Central), but the toolset requires a license fee.

Although Flash Lite's low-level device integration might seem to be a limitation, third parties offer low-level device APIs that support the development of innovative applications. For example, the KuneriLite toolkit extends Flash Lite capabilities in the Symbian platform. The cost, of course, is greater because of fragmentation issues and memory footprint.

Android

Google launched Android (http://code.google.com/android) in 2007, to advance open standards for mobile devices. Android is an Apache free-software platform with an open source license for mobile devices based on Linux. Its software stack for mobile devices includes an operating system, middleware, and key applications.

Features. Android applications are primarily written in Java and compiled into Dalvik executable (DEX) format, a custom byte code. Each application executes on its own process, with its own instance of the Dalvik virtual machine. Dalvik runs DEX files, which are converted at compile time from standard class and JAR files. DEX files are more compact and efficient than class files.

Developers have full access to all the frameworks and APIs that the core applications use and to Google-developed software libraries. Android's software architecture is designed to simplify component reuse. Any application can publish its capabilities, and any other application can then use those capabilities, subject to security constraints enforced by the framework.

The Android software development kit (SDK) supports authoring applications with rich functionality. Like the iPhone, it can handle touch screens, accelerometers, 3D graphics, and GPS as well as collaboration among applications like email, messaging, calendars, social networking, and location-based services.

Review. Android supports a relatively large subset of the Java Standard Edition (SE) 5.0 library, implying reduced migration cost from Java desktop applications. It also supports several third-party libraries. Similarly to Java ME, application development is powered by popular Java integrated development environments (IDEs) such as NetBeans and Eclipse. Android provides inherent support for modular service-oriented applications and interapplication communication. Java ME's MIDP 3.0 similarly supports inter-MIDlet communication.

New platform releases introduce many new user and developer features—for example, account synchronization, improved media-playing performance, and database and geolocation API support—but also raise fragmentation concerns. Phones running Android 1.0, 1.5, 1.6, and 2.0 as applications might have trouble working smoothly across all the operating system versions. The platform's openness in the targeted device stacks aggravates the fragmentation problem.

A Mobile-Game Case Study

We implemented four identical versions of a toy application, a game called Snake (see Figure 2). The implementations let us compare the platforms with respect to development effort and time as well as several technical issues, such as sound reproduction, still image dis-

TABLE 1

Technical issues related to the Snake game implementation.

Technical issue	Java ME	.NET CF	Flash Lite	Android
Development effort (desktop to mobile porting)	Port GUI from Swing to liquid-crystal display user interface (LCDUI) classes; convert the main() method of the base class to the MIDlet's startApp(); convert Java Database Connectivity (JDBC)-based persistent storage to Record management system (RMS)	Adapt GUI to use Windows Forms controls, and switch the database to SQL Server Mobile Edition	Adapt GUI to screen size	Make changes to interfaces implemented and parent classes extended; switch the database to SQL Lite; specify the GUI in an XML file, using the DroidDraw GUI designer
Lines of code	~1,100	~800	~600	~900
Lines of code modified to port the desktop application	~400	~150	~50	~200
Deployment application size	29 Kbytes	63 Kbytes	12.6 Kbytes	23 Kbytes

play, menu and application interface design, key events handling, memory use, deployment file size, and reusability of code authored for equivalent desktop applications (see Table 1).

To ensure fair comparisons, we focused on evaluating the particularities of mobile-application development rather than the different programming language characteristics. Along this line, we first implemented the desktop game application and then migrated the applications to the mobile platforms, reusing source code wherever possible—that is, we ported Java SE code to Java ME and Android, C#.NET code to .NET CF, and Flash (ActionScript) code to Flash Lite. The game used relatively simple graphics to convey snake body movements in response to key-pressing events. The implementation included a short sound file that played when the snake ate food. It also offered pause, resume, and change functions to adapt the snake's speed, and it kept high scores and game state in the device's persistent storage.

The smoothness and expressiveness of the snake's movement largely depends on the device's characteristics (screen resolution, screen frame rate, and processor frequency). We weren't able to quantitatively assess these characteristics on the available platform emulators.

Mobile gaming has been a major driving force for the mobile-application market. Java ME is currently the de facto standard for downloadable cell phone games,[4] particularly because it has a game API. Furthermore, Java ME is the only framework providing a low-level 3D graphics API. Flash Lite is a popular gaming platform, mainly because of its development speed and suitability to graphics-intensive applications. Flash Lite 3.0 focuses more on video and multimedia support than game development. However, Flash Lite is ideal for integrating games in webpages—similar to developing Flash movies for desktops. .NET CF and Android haven't yet gained significant market share in game development.

Desktop-to-mobile application porting was more labored in Java ME. Tools like JDiet (a Java SE 1.4-to-Java ME CLDC converter) can be useful but don't support GUI conversion. We designed the MIDlet's menu templates using the Java ME Polish tool collection (www.j2mepolish.org), which includes build tools for creating application bundles for multiple devices and locales; a device database that helps adjust applications to different handsets; tools for designing GUIs using simple CSS text files; and utility classes. We had to port JDBC-based storage—for example, to store game state or scores—to RMS, which isn't a full-fledged database system but is similar to the shared-objects approach taken in Flash Lite. However, the TiledLayer and GameCanvas classes of the Java ME Game API were extremely useful for painting the game's landscape and conveying the snake's movement.

.NET CF and Android applications were easier to develop because of their improved compatibility with the full .NET and Java SE frameworks, respectively. The use of SQL databases in both these platforms was also straightforward. We adapted a few C# method invocations for .NET CF because it didn't support the corresponding libraries. Android didn't require such modifications. Furthermore, sound support was poor in .NET CF, handling only uncompressed sound playing, which increases the application size. The Flash-to-Flash Lite migration was relatively effortless because we used the same ActionScript code in both cases.

We had to translate desktop application key events to the respective mobile phone's keyboard events in all platforms. The GUI design was relatively easy using available Visual GUI builders; for Android, we had to get this tool through the third-party DroidDraw builder. Notably, the separation of the program logic from the GUI design proved useful for all platforms, letting us use the same game logic classes for both the desktop and mobile applications.

Platform Comparison

Table 2 evaluates the reviewed platforms in five qualitative and quantitative areas: software architecture and technical issues, application development, capabilities and constraints, developer communities and market success, and development tools.

The data reflects our product reviews and development experience, both prior to and after the Snake game implementation. It also factors in results generalized from an informal online survey we conducted of 32 mobile-application developers. These results are indicated by an asterisk in the issue description. For each platform, at least seven developers participated in the survey, which included 16 questions regarding their experience with the platform. The survey is available at http://mobileapps.limeask.com. Some quantitative information derived from a compilation of the surveys and not discussed here is available at http://www2.aegean.gr/dgavalas/en/mob_survey.pdf.

Platform Status and Trends

In current practice, devices vary along so many axes that it's almost impossible to write a single version of a mobile application to run on a broad range of devices. Fragmentation increases the production effort in almost the entire software life cycle—driving up the cost, lengthening the time to market, and narrowing the target market. Better standardization (for example, fewer optional APIs and more detailed specifications) and stricter enforcement of standards (for example, using API verification initiatives and technology compatibility kits) could help in this regard. Major players in the mobile-application industry (such as platform vendors, device manufacturers, and operators) can play a critical role in the war against fragmentation.

Java ME is undoubtedly the platform with the broadest deployment base and still maintains the largest market share, yet it's the platform most affected by fragmentation and so might be displaced by alternative platforms. Sun Microsystems has published a set of guidelines to reduce the practice of generating distinct executables for each phone.[6] Some tools for resolving Java ME device fragmentation are already available (for example, NetBeans Mobility Pack 5.5 for CLDC), but there is still a long way to go.

Along the same line, the Mobile Services Architecture (MSA) has emerged as an industry standard to reduce fragmentation and give developers a consistent Java ME platform. In addition to specifying what component JSRs a compliant device must include, the MSA also clarifies behavioral requirements to improve JSR predictability and interoperability. The MSA defines two stacks: a full stack that comprises 16 JSRs (JSR 249), and a subset of eight JSRs (JSR 248). JSR 248 is being pushed ahead of JSR 249 to help developers get an earlier start on MSA-compliant applications. JCP has recently approved JSR 248, but its adoption by OEMs remains to be seen.

Java ME's competitiveness against platforms that target graphics-heavy applications, such as Flash Lite, will also depend on technologies that enable expressive, feature-rich content on mobile devices. Along this line, Sun Microsystems has recently released JavaFX Mobile (http://javafx.com), a new platform and language with Rich Internet Applications (RIA)-friendly features, including a declarative syntax of the JavaFX Script language for GUI development. JavaFX Mobile lets developers build expressive interfaces while reusing existing back-end Java code. It also lets development team members with no programming experience, such as designers and graphic artists, create graphics-intensive front ends for mobile applications. However, OEMs will decide JavaFX Mobile's success by the support they offer, for example, by integration of its binaries and runtime on mobile handsets.

.NET CF will probably maintain its developer base as long as Windows handhelds remain in the picture. It's a powerful platform for programming and accessing native components of Windows-compatible PDAs and smartphones. However, its market share isn't likely to increase because porting it to popular phone operating systems is cumbersome. Specifically, it requires implementing a platform-adaptation engine to interface between the CLR and the operating system.[7]

The release history of Flash Lite indicates that Adobe has concentrated more on supporting multimedia than defining a powerful API for developing applications with rich functionality. Despite the effort to establish Flash Lite as a gaming platform, it lacks APIs or classes specifically targeting game development. For example, Flash Lite 3.0 doesn't support the BitmapData object that's part of Flash 8 and useful for game development. It also needs to improve its sound handling. Furthermore, comparative studies indicate

DEVICES VARY ALONG SO MANY AXES THAT IT'S ALMOST IMPOSSIBLE TO WRITE A SINGLE VERSION OF A MOBILE APPLICATION.

TABLE 2 Comparison of programming platforms in five areas.

Issue description	Java ME	.NET CF	Flash Lite	Android
Software architecture and technical issues				
Footprint	~128 Kbytes for storage of kernel-based virtual machine and associated libraries	1.55 Mbytes on Windows Mobile-based Pocket PC 2000/2002; 1.35 Mbytes on Windows Mobile for Pocket PC 2003 or Windows CE .NET devices	450 Kbytes for the core library of Flash Lite 2.1; 374 Kbytes for Flash Lite 3.1	3 Mbytes
Runtime memory requirement	< 0.5 Mbytes	~ 0.5 Mbytes	2–4 Mbytes	Minimum 32 Mbytes of RAM
Memory management	Automatic memory management provided by the traditional garbage collector, which deallocates memory occupied by objects that the program no longer uses	Automatic memory management provided by Common Language Runtime (CLR); the CLR garbage collector manages the allocation and release of memory for an application	Garbage collection executed automatically every minute or whenever an application's memory use increases by 20 percent or more	Automatic memory management handled by Dalvik's garbage collector; garbage collections might noticeably decrease performance
Device support	All devices support Connected Limited Device Configuration (CLDC), Mobile Information Dance Profile (MIDP) (practically, lacks support only for Windows Mobile-based Pocket PCs)	Pocket PC 2000, Pocket PC 2002, Windows Mobile 2003-based Pocket PCs and smartphones, embedded systems running Windows CE .NET 4.1 and later	Mobile phones and PDAs from major manufacturers such as Fujitsu, Hitachi, LG, Mitsubishi, Motorola, Nokia, Panasonic, Samsung, Sanyo, Sharp, and Sony Ericsson	Mostly HTC devices (Magic, Hero, Tattoo); also T-Mobile (G1, Pulse), Motorola Dext, Samsung Galaxy i7500, Acer Liquid, Sony Ericsson Xperia X10; Android 2.0-compatible handsets announced (Motorola, Samsung)
User interface (UI) components	High-level LCDUI components, such as Form or List; low-level LCDUI for controlling every UI pixel; support for SVG (defined in JSR 287); J2ME Polish allows design along with animations and effects specified in external CSS-like files	Windows Forms controls (vary for Pocket PCs and smartphones)	Nokia Flash Lite Feather Framework (FL 2.x), Sony Ericsson Adobe XD UI components (FL 1.1/2.x)	View and ViewGroup objects; DroidDraw tool serves for rapid UI design; J2ME Polish enables conversion of Java ME MIDlets' UI to Android-compatible UI
Development languages	Java (CLDC/MIDP)	C#, Visual Basic .NET	ActionScript 1.0, ActionScript 2.0	Java (Android SDK)
Packaging	Java Application Description (JAD) and Java archive (JAR) files	Cabinet (CAB) file installers	SWF files	Android package (APK) files
Deployment methods	Over the air (OTA), Bluetooth/IR, Wireless Application Protocol (WAP) push	OTA, Bluetooth	Bluetooth, physical cable, OTA	OTA, Bluetooth
Server-side technologies*	Java servlets, JavaServer Pages (JSP)	ASP.NET Mobile Controls	Flash Media Server (uses ActionScript 1 for server-side logic)	Java servlets, JSP
Persistent storage and database support	RMS and Perst Lite from mObject	Local database support for SQL Server Mobile Edition; on the server side, support for SQL Server	Persistent storage through shared objects; on the server side, support for interaction with PHP scripts and use of MySQL database	Android APIs contain support for SQLite database

(CONTINUED ON P. 84)

TABLE 2 (CONTINUED)

Issue description	Java ME	.NET CF	Flash Lite	Android
Sound handling and supported formats	MP3 and whatever format the device supports	Uncompressed pulse-code modulation (PCM) files only; support for well-known formats (WAV, MP3, and so on) offered by third parties (such as Resco Audio for .NET CF)	Sound files embedded within the SWF file; supports MP3, AIFF, AU, WAV, and so on; no support for simultaneous playback of multiple sounds	3GP, MP3, MP4
2D/3D graphics handling and supported formats	All MIDP versions support the display of rasterized images (in PNG format only); MIDP added support for SVG (JSR 226); MIDP 3.0 adds support for GIF images; support for mobile 3D graphics on Mobile 3D Graphics (M3G) format: M3G 1.0 (JSR 184) or M3G 2.0 (JSR 297)	Support for BMP, JPG, GIF, and PNG formats; doesn't support SVG; Direct3D mobile applications available for Windows Mobile 5.0	Vector-based graphics includes support for bitmap; doesn't provide low-level 3D graphics API, but it's possible to use a sequence of images exported from a 3D tool	Supports PNG, JPG, and GIF; doesn't support SVG; supports 3D graphics via the OpenGL API
Application development				
Learning curve*	Moderate (developers need to be familiar with several APIs that aren't part of the Java SE platform)	Average (significant overlap with the .NET platform APIs)	Steep (reuse of the same ActionScript code)	Average (significant overlap with the Java SE platform APIs)
Developer community base*	Large community	Relatively large community	Relatively large community	Fair-sized and fast-growing community
Debugger availability	Excellent	Excellent	Good	Integrated in Eclipse; stand-alone debugging monitor also available
Cross-platform deployment	Execution on any device supporting CLDC/MIDP, but inconsistent implementations across vendors necessitates separate builds	Windows Mobile, Symbian-based devices (via third-party tools)	Excellent (supported by top five mobile manufacturers; best Web compatibility)	Android only, because of Dalvik VM
Deployment speed (packaging, installing, testing)*	Slow (fragmentation problem)	Relatively fast	Relatively fast	Relatively fast
Capabilities and constraints				
Functionality	Varies by handset, depending on available JSRs; no high-resolution pictures, no cell ID, limited file access	Limited audio support	No support for accessing native components	Touch screen, accelerometer, GPS, cell ID, interapplication communication
Event model	Event-handling mechanism based on command objects	GUI events bound to methods through multicast delegates	Uses the powerful ActionScript's event model (movie clip and object events)	Inherits the Java event model; uses a special class (intent) that enables application responses to external events, such as a phone call
Phone data access	Varies by handset, depending on available JSR 75, the PDA Optional Packages	Full	None	Full
Runtime speed	Average (because of Java bytecode)	Average (because of CLR managed code)	Below average (interpreted language)	Average because of Java bytecode

TABLE 2 (CONTINUED)

Issue description	Java ME	.NET CF	Flash Lite	Android
Developer communities and market success				
Developer community and support*	Extensive	MSDN	Extensive	Recent, growing
Market penetration*	Extensive (also the basis of the Danger Sidekick Platform)	Average	Average	Potential to gain wide acceptance, based on the support of 34 major software, hardware, and telecom companies
Distribution and licensing	None (Signed Java)	None (third parties can provide licensing)	None	None
Development tools				
Integrated development environment (IDE) availability	Eclipse, NetBeans	Visual Studio .NET 2008, 2010	Adobe Flash CS4, Adobe Device Central	Eclipse, NetBeans (with Android plug-in)
Emulator availability	Free emulator, Sun Java Wireless Toolkit	Bundled with IDE	Bundled with IDE	Free emulator
Development tool cost	Free	Free (but only basic tools)	Varies: free but limited with Motion-Twin ActionScript 2 Compiler IDE	Free

** Information derived mainly from compilation of online survey reports.*

that Flash Lite exhibits lower performance and frame rate while consuming more memory than Java ME.[10] On the other hand, Flash Lite appears a natural choice for designing user interfaces and graphically rich applications. In that sense, it lets designers into the mobile-development space. A promising evolution path for Flash Lite seems to lie in its synergy with different application platforms, bringing together the best of diverse worlds. Recently, the Capuchin Project (http://developer.sonyericsson.com/site/global/docstools/projectcapuchin/p_projectcapuchin.jsp) defined a Java ME API as a bridge between Java ME and Flash Lite. It enables use of the latter as the front end and the former as the back end of applications, allowing developers to use Flash tools for GUI design while still having access to all the phone services available to Java ME.

Android initially received an enthusiastic welcome from manufacturers and developers, but some handset manufacturers are taking longer than expected to integrate it. Hence, its market share isn't growing as rapidly as anticipated. Still, the Android developer community seems to be growing—mainly in comparison to Java ME. Its future will largely depend on providing technologies for simplifying the design of multimedia-rich applications. Sun Microsystems announced that JavaFX Mobile will be available on the Android OS. Most important will be how well Android handles fragmentation problems. It's too early to answer this question now, given Android's relatively narrow installation base.

Because Android is a relatively young software platform, it's struggling with a small number of available applications. Google has invested in attracting developers and preparing a critical mass of applications prior to the first Android phone release. Running a large number of existing Java ME applications could also add value to Android. Along this line, some vendors are providing porting services to convert existing Java ME titles to the Android platform. Examples include Tira Wireless and J2ME Polish.

Assessments

On the basis of our review, we've assessed the appropriateness of each platform with respect to four critical application development requirements:

Portability

The diverse hardware and software represented in today's handheld devices inevitably make portability a puzzle for mobile-application developers. Portability primarily depends on runtime support and the feasibility of achieving identical look-and-feel and functionality across platforms. In terms of runtime support, Java ME is undoubtedly the winner, followed by Flash Lite. Android is likely to extend its deployment base, and .NET CF will probably remain a Windows-only platform.

On the other hand, Java ME exhibits fragmentation in cross-platform application development. Flash Lite is a better choice in this regard because of Adobe's strict control over its runtime environment. Android's handling of fragmentation remains unclear given its slow adoption pace and its alternative business model, which is open source yet tightly controlled by Google. Fragmentation isn't an issue for .NET CF, given its narrow range of supporting devices.

Functionality

Java ME best serves the aim of implementing multimedia-rich full-fledged applications, such as games, through the numerous APIs (JSRs) implemented by OEMs to exploit handset capabilities. .NET CF and Android applications also use rather powerful APIs. Flash Lite is most suitable to graphics-heavy applications.

Development Speed

Rapid time-to-market is a critical requirement in mobile applications. Taking advantage of developers' programming experience on desktop applications is the safest way to ease the learning curve and shorten the development time. For instance, Java developers will find a natural fit with Java ME and Android, Flash developers with Flash Lite, and so on.

Developers not familiar with any of the platform foundation languages will feel more comfortable and productive with Flash Lite's ActionScript. Nevertheless, the development process in traditional platforms such as Java ME and .NET CF is accelerated because the documentation and developer community bases are extensive.

Performance

As mobile applications become more computationally intense and require faster runtime speeds and storage I/O, performance also becomes a critical issue. Metrics such as processing overhead, memory consumption, frame rate, and deployment file size all depend on the particular development platform toolset. For example, does it support SVG Tiny, graphics buffering, compressed sound files, and so on? Java ME, .NET CF, and Android achieve comparable performance, whereas Flash Lite has lagged in various benchmarks.[8]

Even though market and application requirements largely determine the platform for mobile development, our review offers some specific and general guidance into the assets and deficiencies of available tools as developers face the increasing demand for applications on resource-constrained devices.

ABOUT THE AUTHORS

DAMIANOS GAVALAS is an assistant professor in the University of the Aegean's Department of Cultural Informatics. His research interests include mobile computing, mobile ad hoc and wireless sensor networks, and optimization algorithms. Gavalas has a PhD in electronic engineering from the University of Essex, UK. Contact him at dgavalas@aegean.gr.

DAPHNE ECONOMOU is a lecturer in interactive multimedia and hypermedia at the University of the Aegean's Department of Cultural Technology and Communication. Her research interests include collaborative virtual reality environments for learning and archaeology, human-computer interaction, and multimedia application design for mobile devices. Economou has a PhD in computer science from Manchester Metropolitan University. Contact her at daphne@ct.aegean.gr.

References

1. M. Kenteris, D. Gavalas, and D. Economou, "An Innovative Mobile Electronic Tourist Guide Application," *Personal and Ubiquitous Computing*, vol. 13, no. 2, 2009, pp. 103–118.
2. S. Blom et al., "Write Once, Run Anywhere A Survey of Mobile Runtime Environments," *Proc. 3rd Int'l Conf. Grid and Pervasive Computing* (GPC 08), IEEE CS Press, 2008, pp. 132–137.
3. C. Neable, "The .NET Compact Framework," *IEEE Pervasive Computing*, vol. 1, no. 4, 2002, pp. 84–87.
4. O. Anderson et al., *Scalable Vector Graphics (SVG) Tiny 1.2 Specification*, World Wide Web Consortium (W3C) recommendation, 22 Dec. 2008; www.w3.org/TR/SVGMobile12.
5. J. Soh and B. Tan, "Mobile Gaming," *Comm. ACM*, vol. 51, no. 3, 2008, pp. 35–39.
6. Sun Developer Network, "Java ME: De-fragmentation," 2006; http://developers.sun.com/mobility/reference/techart/design_guidelines.
7. A. Gefflaut et al., "Porting the .NET Compact Framework to Symbian Phones-A Feasibility Assessment," *J. Object Technology*, vol. 5, no. 3, 2006, pp. 83–106.
8. A. Koller, G. Foster, and M. Wright, "Java Micro Edition and Adobe Flash Lite for Arcade-Style Mobile Phone Game Development: A Comparative Study," *Proc. ACM Ann. Conf. South African Inst. Computer Scientists and Information Technologists* (SAICSIT 08), ACM Press, 2008, pp. 131–138.