Java is the Language of Choice for Creating Internet Services Applications for Wireless Devices

Abstract

Java 2 Platform, Micro Edition (J2ME) is turning into a major platform for wireless application development. Compared with desktop and server-side application development, Java-based wireless development faces some unique challenges. For example, small wireless devices have little processing power, limited memory, short battery life, and unreliable network connections. To develop wireless applications, we must design graphical user interfaces suitable for small devices, use lightweight components, and shift resource intensive tasks to the server side using sound network designs. As a result, J2ME supports only parts of the standard Java API libraries, with lightweight alternative APIs for handling complex tasks such as GUIs.

For more than five years, Java 2 Platform, Micro Edition (J2ME) and its predecessors, such as PersonalJava, have served as the only viable platforms for developing managed smart clients on mobile or embedded devices. But that will soon change. With the introduction of Visual Studio .Net (VS.Net) 2003 in the first half of 2003, Microsoft will make its managed environment, .Net, available on Windows-powered mobile devices with its new product, the .Net Compact Framework (.Net CF).

Given Windows' dominance in the desktop market and .Net's presence in the enterprise market, every mobile Java developer must be prepared for the coming challenge. I argue that even though some standardization is need, Java is the language of choice for developing Internet/Web services applications for Wireless devices. The reasons for this include: support for multiple platforms, security, the API specification process, support for both consumer and enterprise applications, life cycle management, and great variety in development tools.

Introduction

J2ME and .Net CF are the two platforms for managed smart mobile clients. They are critical new technologies for advanced mobile commerce. Compared with micro-browser technologies such as WAP (Wireless Application Protocol), smart clients support rich user interfaces (UIs), leverage device extensions (e.g., GPS (global positioning system) and bar-code scanners), and support more flexible integration and security schemes. Smart clients also reduce network traffic and improve transactional stability through on-device data storage. Compared with smart clients on native platforms (such as embedded Visual C++ or C++ SDKs for the Symbian OS), Java- or .Net-managed environments greatly improve developer productivity, application reliability, and mobile code security.

Designed for mobile computing, .Net CF is a lightweight version of Microsoft's .Net Framework. The .Net CF Common Language Runtime (CLR) runs standard .Net byte code applications. .Net CF contains a subset of standard .Net API libraries necessary for mobile application development. It runs only on Windows CE/Pocket PC-powered high-end PDAs.

In the Java camp, the situation is more complex. J2ME contains standardized configurations and profiles designed to provide the best compromise between portability and performance for a range of mobile devices. Configurations support the Java core APIs. Profiles are built on top of configurations to support device-specific features such as networking and UIs. Each valid combination of configuration and profile targets a specific type of device:

- Profiles on top of the Connected Device Configuration (CDC) target high-end networked devices. Those devices have similar hardware capabilities as .Net CF devices. The CDC includes a standard Java 2 Virtual Machine so it can run standard Java byte code and utilize Java 2 Platform, Standard Edition (J2SE) libraries if desired. The CDC Personal Profile has roughly the same functionality as the older PersonalJava environment.
- Profiles on top of the Connected Limited Device Configuration (CLDC) target low-end PDAs and small cell phones. The CLDC uses a small footprint VM that is not compatible with J2SE or the CDC.

Table 1. Overview of the three platforms

	.Net Compact Framework	J2ME Connected Device Configuration	J2ME Connected Limited Device Configuration
Device requirement	Powerful, expensive	Powerful, expensive	Cheap, pervasive
Cost	High	High	Medium
Market focus	Enterprise	Enterprise	Consumer and enterprise
Language support	C#, VB.Net	Java	Java
Platforms	Pocket PC, Windows CE	Major mobile platforms except Palm OS	All mobile platforms
Byte code compatibility	Standard .Net CLR	Standard Java 2	Not compatible with J2SE or CDC
API compatibility	Subset of .Net	Subset of J2SE plus standard optional packages	Partial compatibility with CDC with additional standard optional packages
Native APIs	P/Invoke; consistent across supported devices	JNI; device- and OS-specific	N/A
Development tools	VS.Net 2003	Command line, vendor SDKs, CodeWarrior, and WebSphere	Command line, vendor SDKs, all major Java IDEs

Specification process	Single company	Community	Community
Service gateway	N/A	Run gateways as OSGi servlets; run gateway clients via vendor-specific SDKs	Run gateway clients via vendor-specific SDKs
Security model	Simplified .Net model	Full Java security manager	Limited Java 2 model supplemented by OTA specification
Client installation	ActiveSync, Internet Explorer download	Sync, download	Formal OTA specification
Life cycle management	N/A	OSGi for gateway apps, J2EE Client Provisioning Specification for generic clients	Included in OTA spec, works with J2EE Client Provisioning Specification

Support for Multiple Platforms

Comparing J2ME and .Net Compact Framework is more complex than any simple features-to-features comparison can encompass. For one thing, the analysis must include nontechnology dimensions such as viability, market acceptance, development and testing tools, reach (J2ME today reaches down to far smaller devices than does Windows CE, and today, at least, .Net CF has not been ported beyond CE), and standardization and coherence of the platform. The future direction of each platform also needs to be taken into account, given the early stage not only of the two platforms but also of many of the markets in question. J2ME's Mobile Information Device Profile (MIDP) variant is in production, and the other two leading platforms, PDA Profile and Personal Profile, are in the final stages of standardization. Meanwhile, .Net CF is in the final stages of its beta tests.

However, Windows devices consist of only a small part of today's mobile device population. On cell phone devices, Motorola iDEN, Nokia Symbian OS, and Qualcomm Brew platforms are prevalent, and there are many vendor-specific OS platforms (e.g., Nokia Series 40). On low-end PDAs, Palm OS is the dominant player; on embedded or telematics devices, RTOS (real time OS) platforms such as QNX Software Systems and Wind River's VxWorks are the most widely used. Even on high-end PDAs, where Windows has the largest market share, devices based on the Symbian OS and different Linux flavors are growing more and more popular. More than 200 Linux PDAs are already available, and IBM has just recently announced a blueprint for high-end PDAs based on the PowerPC chip and Linux OS.

At the bottom of the J2ME hierarchy are two configurations that provide JVMs and core language libraries. Connected Limited Device Configuration (CLDC) is for the smallest wireless devices with 160 KB or more memory and slow 16/32-bit processors. CLDC has limited math, string, and I/O (input/output) functionalities and lacks features such as JNI (Java Native Interface), custom class loaders, and reflection. As a result, CLDC virtual

machines cannot support most J2SE standard libraries. Connected Device Configuration (CDC) is for more capable wireless devices with at least 2 MB of memory and 32-bit processors. CDC supports a fully featured Java 2 VM and therefore can take advantage of most J2SE libraries.

Configurations provide the most basic and generic language functionalities. On top of each configuration are multiple profiles targeted at specific devices. The profiles provide more advanced APIs such as a graphical user interface (GUI), persistent storage, security, and network connectivity. Mobile Information Device Profile (MIDP) and PDA Profile, two profiles based on CLDC, target cell phones and PDA devices, respectively. Based in CDC, the Foundation Profile provides more utility, network, and security functions for embedded devices, but no GUIs. The Personal Profile sits on top of CDC and the Foundation Profile. It provides J2SE Abstract Windowing Toolkit (AWT)-compatible GUI APIs for high-end PDAs and wireless Internet appliances. Figure 2 illustrates the J2ME platform architecture.

The heavy competition in the wireless devices market benefits customers by offering choices. But the fragmented platforms create a nightmare for developers. Repeatedly developing and maintaining different versions of the same application for different devices is prohibitively expensive. Yet, any general-purpose wireless application must run on all types of devices customers might own. Thus, cross-platform compatibility is a core issue to consider when developing wireless applications. This is where Java really shines. Many of the above-mentioned mobile platforms now have built-in Java support. Third-party J2ME runtimes from Insignia and IBM are available for all mobile platforms, including all Windows flavors. The Java approach allows developers to be productive across many mobile platforms.

Security

In addition to providing a compatibility layer, the JVM also provides application security through the following ways:

- The JVM verifies the byte-code application at class load time to ensure that the application does not do anything dangerous
- The JVM monitors the application execution and avoids memory leaks
- The JVM can incorporate a security manager or sandbox to verify application digital signatures and grant permissions to access specific APIs (domains) based on the signer's trust level

The three security features above prove important to wireless applications. Wireless devices usually store personal and sensitive data. Users often download mobile code from an insecure Internet. We must ensure that the downloaded application is not a virus and will not try to corrupt or even steal data from the device. To combat such attacks, the J2SE and J2ME/CDC platforms offer fully featured, secure JVMs. J2ME/MIDP uses a simplified byte-code verification scheme to minimize performance hits. Domain-based security managers are available for MIDP 2.0 VMs.

In addition to application security, Java provides excellent support for industry-strength authentication, authorization, and cryptography-based communication security protocols. The Java Cryptography Extension (JCE) and Java Authentication and Authorization Service (JAAS) frameworks are both pluggable and flexible. They provide a set of abstract interfaces of common security algorithms and concepts to application developers. Different providers supply the implementations. You can easily add new algorithms into the frameworks. Java's security framework design separates applications from security solution providers. Therefore, it allows you to choose or switch to any provider and algorithm at any stage of software development without learning new APIs or worrying about compatibility issues. We will discuss more about Java application security technologies in the following section.

API Specification Process

.Net CF provides a homogeneous set of tools and APIs for development on all Microsoft-supported devices. When a new technology emerges, Microsoft can make it available on .Net CF quickly, with no lengthy debates and duplicated efforts. However, it's Microsoft, and not the customers, decides those "important" features are. This strategy's success depends on Microsoft's ability and willingness to listen to customer feedback.

Since J2ME is designed to be cross-platform, the J2ME specification and implementation are two separate processes. Through the Java Community Process (JCP), a committee of mobile solution providers decides the new J2ME standard APIs. Sun has veto power on only Java language specifications. After the API specification is developed, each company can develop its own implementation. That ensures portability and minimizes developers' learning costs while preserving vendors' incentives to differentiate and innovate. Since all API specifications are reached by industry consensus, most likely they will be supported in the future. The JCP develops all current J2ME configurations, profiles, and optional packages. This democratic process has worked great so far; however, it has been criticized for being slow and inefficient.

Consumer Applications

During the past several years, mobile developers have focused on consumer wireless applications. Mobile games available on the NTT DoCoMo networks and new camera phones with multimedia messaging service capabilities are hot topics.

Although .Net CF is not specifically marketed toward the consumer market, it supports direct draw on canvas, double buffering, and device button remapping through its rich Windows Forms UI library. Using the native APIs from Windows Media Player on Pocket PC, .Net CF applications can support multimedia playback.

The J2ME platforms have strong support for consumer applications. MIDP 2.0 includes animation and game controls in the javax.microedition.lcdui.game package. Multimedia

playback is supported via the Java Media Framework (JMF) on the CDC or the multimedia optional package for the CLDC/MIDP. The Java Game Profile (Java Specification Request (JSR) 134) could enable 3D games on CDC devices, but it has been inactive for a long time.

Due to the lack of direct hardware access, neither .Net CF nor J2ME proves suitable for high-performance video game applications. Both platforms' futures lie in enterprise mobile applications. Microsoft's "depth-but-not-breath" approach allows it to pack many enterprise-oriented features in .Net CF. In the J2ME camp, support for enterprise applications is also going strong with the backings of enterprise IT giants such as IBM and Oracle. In the next several sections, I will focus on features essential to enterprise mobile applications. Table 2 summarizes features supported on each platform.

Table 2. The feature matrix

	.Net Compact Framework	J2ME Connected Device Configuration	J2ME Connected Limited Device Configuration
User interface	Rich subset of Windows Forms	Rich subset of AWT (Abstract Windowing Toolkit), vendor-specific UI libraries	MIDP liquid crystal display UI, PDA Profile subset of AWT, vendor-specific UI libraries
Database API	Subset of ADO.Net, DataGrid	Rich subset of JDBC	Vendor-specific JDBC-like APIs
Mobile database	SQL Server CE, Sybase iAnywhere Solutions(coming soon)	IBM DB2 Everyplace, iAnywhere Solutions, PointBase, Oracle9i Lite	Vendor-specific relational implementation over RMS, Oracle SODA
Remote database	Any ADO.Net compatible	Any JDBC compatible	Vendor-specific JDBC-like API bridge
Database synchronization	Vendor specific	Vendor specific	Vendor specific
XML API	Build into ADO.Net and other standard APIs	Third-party tools (standards coming soon)	Third-party tools (standards coming soon)
Web services	Built-in	Third party (standards coming soon)	Third party (standards coming soon)
Web services tools	Integrated with VS.Net	kSOAP plug-ins for leading IDEs	kSOAP plug-ins for leading IDEs
Direct RPC	Not recommended	Rich subset of RMI (remote method invocation)	N/A
Email and PIM (personal information manager)	P/Invoke Outlook APIs	JavaPhone and third-party APIs	Upcoming PDA Profile and third party
SMS	P/Invoke device SMS stack	Wireless Messaging API	Wireless Messaging API
	P/Invoke MSN (Microsoft Network)	Third-party APIs for most IM clients	Third-party APIs for most

	(Microsoft Network) and other IM client APIs	most IM clients including Jabber and Jxta	IM clients including Jabber and Jxta
Enterprise messaging	P/Invoke MSMQ	Proprietary JMS (Java Message Service) APIs	JMS via third-party toolkits (e.g., WebSphere MQ Everyplace, iBus Mobile)
Cryptography	Third-party APIs	JCE (Java Cryptography Extension) and third-party libraries	Third-party libraries
Multimedia	P/Invoke Windows Media Player APIs	Subset of JMF	Built into MIDP plus J2ME multimedia APIs
Game	Included Windows Forms UI	Direct draw on Canvas	GameCanvas support in MIDP
Location API	APIs provided by carriers	Third party (standards coming soon)	Third party (standards coming soon)

Enterprise Databases

To fully leverage smart clients' offline capabilities, an on-device mobile database is crucial. .Net CF supports a substantial subset of ADO.Net (Active Data Objects). The standard relational database access API on the Java platform is Java Database Connectivity (JDBC). The J2ME JDBC optional package supports most JDBC 3.0 APIs on the CDC platform. PersonalJava supports the JDBC 2.x API. On the CLDC platform, several vendors have devised proprietary database implementations over the record management system (RMS). Those implementations support limited JDBC-like methods.

Isolated mobile databases by themselves are hardly useful. They must be synchronized and consolidated with enterprise backend databases. Currently, no standard API for database synchronization exists in either .Net CF or J2ME. Each vendor's mobile database can synchronize only through its own synchronization engine. Mobile database vendors differentiate themselves by offering special optimization and additional features. There are several solutions provided by several leading mobile database vendors:

- **Microsoft SQL Server CE:** The SQL Server CE provides full ADO.Net support for the .Net CF platform. Since the SQL Server CE is bundled with VS.Net, it is pervasive on .Net CF. However, SQL Server CE has a rather large (1.5 MB) memory footprint. Also, SQL Server CE only synchronizes with SQL enterprise databases, requiring the backend environment to be Microsoft only.
- **Sybase iAnywhere Solutions:** iAnywhere's SQL Anywhere Studio is currently the most popular mobile database. According to a 2002 Gartner survey, its market share is 78 percent. A core innovation of SQL Anywhere Studio is to automatically generate custom-built UltraLite databases that only contain the exact functionality your application requires. That drastically slashes the memory footprint without compromising features. SQL Anywhere Studio can currently

generate UltraLite databases for CDC and PersonalJava. UltraLite support for .Net CF will surface in early 2003. Auto-generated UltraLite databases contain APIs to synchronize with Sybase and third-party enterprise databases through iAnywhere Solutions' MobiLink synchronization engine.

- **PointBase Micro:** PointBase provides pure Java embedded databases that run on both the CDC and CLDC/MIDP platforms. SQL support on MIDP is impressive. PointBase UniSync synchronization engine synchronizes with any JDBC backend database with special optimization for Oracle and PointBase Embedded databases.
- **IBM DB2 Everyplace:** DB2 Everyplace is a stripped-down version of DB2 Enterprise database. It supports JDBC and ODBC (Open Database Connectivity) APIs. DB2 Everyplace also contains a product called FastRecordStore, which provides a relational layer over the MIDP RMS. Through the IBM Sync, DB2 Everyplace databases synchronize with most popular backend databases.
- **Oracle9i Lite:** Oracle9i Lite databases have footprints of 50 KB to 1 MB depending on their edition. Oracle9i Lite supports the JDBC and ODBC APIs. On MIDP, Oracle provides a mobile database implementation over RMS. Oracle's MIDP database is completely object oriented using the SODA (Simple Object Database Access) technology. For remote data access from MIDP, the Oracle J2ME SDK has a package that enables simple SQL access to backend databases through the Oracle9i wireless application server. Oracle9i Lite databases only synchronize with Oracle enterprise back ends.

Web Services

XML Web services are the key to future enterprise integration. SOAP (Simple Object Access Protocol) is becoming the ubiquitous protocol for accessing enterprise back ends components. Being an early adopter and promoter of SOAP Web services, Microsoft has a head start on Web services integration with mobile devices. In many cases, developers do not need to write any code and can just treat the remote service as a local object.

On J2ME, SOAP client support is currently not standardized. We must rely on third-party J2ME SOAP libraries, such as the open source kSOAP, to build mobile SOAP clients. Popular J2ME IDEs such as Sun ONE Studio, CodeWarrior Wireless Studio, and WebSphere Studio Device Developer have recently added SOAP client stub generators for kSOAP. Oracle supports J2ME Web services clients through its upcoming 9i wireless application server. The server communicates with J2ME clients using a proprietary RPC (remote procedure call) protocol and relays SOAP messages. In the future, the J2ME Web Services Specification (JSR 172, available in third quarter 2003) will likely standardize J2ME Web services client APIs.

Support for the Service Gateway Paradigm

Service gateway is not a part of either .Net CF or J2ME. But it is an important component in today's smart home or enterprise networks. A service gateway integrates with backend

servers, stores application data, and facilitates messaging on behalf of its thin or lightweight smart mobile clients. The service gateway architecture allows us to use small and pervasive devices while still leveraging rich functionalities. Asynchronous and cache-enabled messaging middleware in the gateway prove essential to ensuring mobile applications' quality of service. The gateway architecture also allows more efficient application designs. For example, the Model-View-Controller (MVC) design pattern can be easily applied.

Mobile Gateways

Mobile gateways implemented on mobile platforms are very attractive. For example, a mobile gateway could be a set-top box for a home network, an auto-mounted device for a car network, or a Bluetooth PDA for a personal network.

Since .Net CF is brand-new, few gateway products are specifically designed for it. In addition, .Net CF presents some technical difficulties. .Net CF was not designed to run lightweight application servers required in mobile gateways. It does not directly support MSMQ (Microsoft Message Queuing) either.

On the more mature J2ME, the primary mobile service gateway product is from IBM: the WebSphere Everyplace Embedded Software Suite that runs on IBM's J2ME runtime known as the WebSphere Micro Environment (formally known as J9). WebSphere Everyplace Embedded Software Suite contains the IBM Service Management Framework (SMF), which is a lightweight server framework that runs on top of the CDC and the Foundation Profile. SMF is an implementation of the OSGi (Open Service Gateway Initiative) specification. OSGi defines a Java framework that runs managed service modules (bundles). One type of service OSGi provides is an HTTP service based on J2EE servlets. The SMF bundles interact with middleware components like IBM DB2 Everyplace and WebSphere MQ Everyplace.

Fixed Gateways

Of course, service gateways can reside on fixed server computers too. In this scenario, the gateway computer runs a full-blown server framework such as ASP.Net (Active Server Pages) or J2EE. This architecture is suitable for heavy-duty gateways serving devices in a limited range (e.g., a factory floor or an office). .Net CF or J2ME devices are gateway clients in this scenario.

In the Microsoft camp, the Microsoft Mobile Information Server (MIS) is a powerful gateway, messaging, and synchronization server for Pocket PC as well as thin client devices. However, .Net CF lacks built-in APIs to interact with Microsoft MIS. I expect third-party vendors will provide such support soon after .Net CF 1.0 is released

In the Java world, Oracle has a complete line of gateway application server products running on top of J2EE. Together with Oracle J2ME SDKs, the upcoming Oracle9i wireless application server provides gateway integration points for mobile devices to many other Oracle or third-party application servers.

Client Provisioning and Life Cycle Management

Device management is one of the most costly parts of today's mobile enterprise solutions. Ensuring that the right users get the right software and that the software is updated promptly is important. For general public mobile applications, wireless network carriers must build walled gardens to protect customers as well as revenue sources.

On .Net CF, applications are installed over ActiveSync or over the air (OTA) through the Pocket PC Internet Explorer. There is no standard way for the back end to control the client once the client deploys.

On J2ME, an application can be managed from the back end throughout its life cycle. For MIDP applications, a well-defined OTA provisioning specification mandates how a MIDlet suite is installed and updated. For mobile service gateway applications, the OSGi framework handles bundle life cycle management. On the server side, the J2EE Client Provisioning Specification defines a complete server framework that matches and deploys smart clients to a variety of devices. The J2EE client provisioning server integrates with customer relationship management modules to enable custom tracking, billing, and upgrading logics.

Development Tools

.Net CF Development Tools

Microsoft's flagship IDE Visual Studio .Net is an excellent product that provides similar design interfaces for desktop and mobile applications. For example, to migrate a desktop UI design to .Net CF, you merely copy and paste visual components to a new designer window. VS.Net also features strong support for Web services integration and relational database access. VS.Net is tightly integrated with Visio Enterprise Network Tools edition, which can generate C# or VB.Net code from UML (Unified Modeling Language) diagrams. VS.Net supports debugging on both high-fidelity emulators and real devices. However, VS.Net is not cheap. As of today, no free command-line tool exists for .Net CF development. Nor does any third-party IDE product support .Net CF. More tools will be available in the future when .Net CF matures.

J2ME Development Tools

On the J2ME front, command-line tools and vendor-specific toolkits are readily available. Sun's J2ME Wireless ToolKit is a widely used MIDP development tool. Antenna is an open source project that extends Java's de facto Ant build tool to J2ME. But of course, for most developers, IDEs are still essential. All major Java IDEs now have J2ME modules or plug-ins:

- Sun ONE Studio Community Edition with wireless modules is free and has excellent support for enterprise features.
- JBuilder with MobileSet has a great visual UI designer and good UML design support.
- CodeWarrior Wireless Studio is bundled with many useful third-party tools. It supports development on both CLDC and CDC/PersonalJava.
- IBM WebSphere Studio Device Developer is based on the popular Eclipse IDE platform. It supports both on-device and emulator debugging. If you choose one of IBM's smart mobile middleware solutions, WebSphere Studio Device Developer is naturally the best tool.
- Oracle9i JDeveloper IDE helps integration between Oracle mobile servers and J2ME clients.
- Data Representation's Simplicity IDE has visual designers not only for UI components, but also for end-to-end communication logic components, such as an XML-based transaction engine. Simplicity supports integration with legacy (mainframe) information servers through a visual screen reader.

A big challenge for all J2ME IDEs is vendor SDK integration. Every device vendor provides SDKs for their device emulators and proprietary J2ME extensions. The Unified Emulator Interface (UEI) is designed to standardize the interfaces between IDEs and device SDKs. But the UEI is available only through a Sun licensing program. An open standard is needed.

Conclusion

.Net CF and J2ME are both excellent platforms for developing smart clients for mobile commerce applications. The .Net CF platform focuses on enterprise applications with rich UI, database, and XML Web services support. VS.Net is an excellent tool for .Net CF development. But .Net CF runs only on Windows-powered high-end PDAs. As a young platform, it currently lacks support for gateway servers and choices for mobile databases. Application life cycle management is also weak.

Java 2 Platform, Micro Edition (J2ME) Mobile Information Device Profile (MIDP) is undoubtedly the dominant mobile phone programming platform today. The number of J2ME-enabled phones and J2ME applications continues to rapidly grow.

J2ME sports a modular design and is portable across a variety of devices. The platform provides balanced support for both enterprise and consumer applications. Most important, J2ME APIs undergo rigorous standardization processes to ensure wide industry support and minimum learning for developers. J2ME vendors offer excellent selections of mobile databases and gateway application server products. However, keeping the J2ME platforms simple and avoiding fragmentation proves challenging. J2ME Web services tools still need improvements and standardization.

References:

http://java.sun.com/j2me/
http://www.microsoft.com/net/
http://www.develop.com/compactframework/ArticleArchive.aspx
http://www.enterprisej2me.com/J2MEvsdotNET/books.html#j2me
http://www.informationweek.com/story/IWK20020912S0011
http://www.javaworld.com/javaworld/jw-12-2002/jw-1220-wireless.html
http://www.javaworld.com/javaworld/jw-10-2002/jw-1025-j2mebenchmark.html
http://www.javaworld.com/javaworld/jw-06-2002/jw-0621-wireless.html
http://www.javaworld.com/javaworld/jw-10-2002/jw-1018-wireless.html
http://216.239.37.100/search?q=cache:zNUtxgyqlBwC:www.csse.monash.edu.au/~wiharto/docs/FinalPresentation.pdf+j2me+vs+.net+cf&hl=en&ie=UTF-8
http://www.imakenews.com/fourthgen/index000012657.cfm
http://www.eweek.com/article2/0,3959,971909,00.asp
http://216.239.53.100/search?q=cache:UmPZTmeGXlEC:astral.ct.monash.edu.au/cse3211/lectures/CSE3211-cfdotnet95.ppt+j2me+%22.net+cf%22&hl=en&ie=UTF-8