Mobile Applications with J2ME™

A White Paper

July 2001

Netflix v. VideoLabs
IPR2023-00628
Netflix. Ex. 1009

ERICSSON

1 Table of Contents

1 TABLE OF CONTENTS 1-1

2 INTRODUCTION 2-2

3 JAVA 3-2

4 J2ME – JAVA 2 MICRO EDITION 4-4

4.1 CONFIGURATIONS 4-5
4.2 PROFILES 4-6
4.3 STAND-ALONE JAVA APIS 4-10
4.4 PROVISIONING 4-10
4.5 PUTTING IT ALL TOGETHER 4-12
4.6 I-MODE/JAVA (JAPAN) 4-12

5 RELATED TECHNOLOGIES 5-13

5.1 PERSONALJAVA 5-13
5.2 EMBEDDEDJAVA 5-14
5.3 JAVA CARDS 5-14
5.4 MEXE 5-15
5.5 MICROSOFT .NET 5-16
5.6 BREW 5-16

6 MOBILE APPLICATION DEVELOPMENT WITH J2ME 6-17

6.1 MIDP FEATURES AND THEIR USE 6-17
6.2 LIMITATIONS 6-20
6.3 TESTING MOBILE JAVA APPLICATIONS 6-21
6.4 CONCLUSION 6-22

7 DEVELOPMENT TOOLS 7-23

8 REFERENCES 8-24

2 Introduction

Java is one of the most exciting things currently happening to the world of mobile devices and applications. This paper is providing insight into Java technology for the application development on mobile devices. It shall give an overview over the key aspects of Java on mobile terminals as it is today, but also over techniques that can be interesting in the near future. An important part is to present the background of J2ME, the Java platform for mobile devices. It should also provide an extensive set of resources to anyone who is interested in understanding what can be done with Java, or even better, who wants to develop her/his own Java applications.

This paper was written from the perspective of an application/service developer and tester. It presumes that the reader is already familiar with the concepts and terminology of Java, and aware of the trends in information and mobile technologies. For more detailed insights into what is actually needed in order to provide support for Java in mobile terminals, numerous references are mentioned throughout the document.

As an important source for many aspects of J2ME, which are described in this paper, we used Eric Giguere's book 'Java 2 Micro Edition: Professional Developer's guide' [1]. For further technical details of programming portable devices, we refer you to this book.

3 Java

Java boomed together with the Internet. Applets were soon after the introduction of the first Web Browsers an important feature to make the Web more attractive. These applications, that are embedded in HTML pages and are automatically downloaded from the network when opening the page, offer a great level of flexibility to service providers and the users. Today, all kinds of applications, from games to e-banking and all in between, are based on Java, whether they run as Applets in web pages or as regular, preinstalled Java Applications.

Not only e-commerce and banking, but also entertainment applications and many other services make Java very interesting for the use on mobile terminals. There are also many practical reasons that make Java a preferable technique:

- Large developer community
  Java is a programming language that is known by a very large programmer community. Besides that, the language is relatively easy to learn. These are important factors to provide the market with a flood of applications.

- Rapid development of applications
  By providing advanced APIs, Java enables rapid application development.

- Platform independence
  Especially in the mobile sector, there is a countless number of operating systems. Java as a platform-independent programming language, lets an application developer reach a much larger market with no extra effort.

- Dynamic content
  With the possibility of downloading new applications over the network as known from Java Applets and the Internet, applications don't need to be preinstalled on the device. Thus, the network presents the user with an enormous playground of applications that he/she can download and use on the go.

- Execution on the device
  By storing applications on the Mobile Terminal, they can be executed even if no network connection is present. This is a significant advantage when comparing Java to WAP.

- Security
  Java is based on a proven security model and offers the application developers with APIs for secure communication over the networks. This makes Java interesting for e-business and e-banking applications.

By providing extensive APIs for all sorts of tasks, the standard Java Runtime Environment grew over the years to an impressive CPU and memory consumer. The execution of big applications with complex graphical user interfaces becomes slow even with powerful hardware. How can this possibly work on very limited devices, such as mobile phones and PDAs? Recognizing, that one size does not fit all, Sun grouped its Java technologies into three editions, each aimed at a specific area of today's vast computing industry:

- Java 2 Enterprise Edition (J2EE)
  For enterprises, needing to serve their customers, suppliers, and employees with reliable, complete, and scalable Internet business server solutions.

- Java 2 Standard Edition (J2SE)
  For the popular and well-established desktop computer market.

- Java 2 Micro Edition (J2ME)
  For the combined needs of:
  - Consumer and embedded device manufacturers who build a diversity of information services
  - Service providers who wish to deliver content to their customers over resource-constrained devices
  - Content creators who want to make compelling content for resource-constrained devices

How these editions share the market is seen in figure 1.

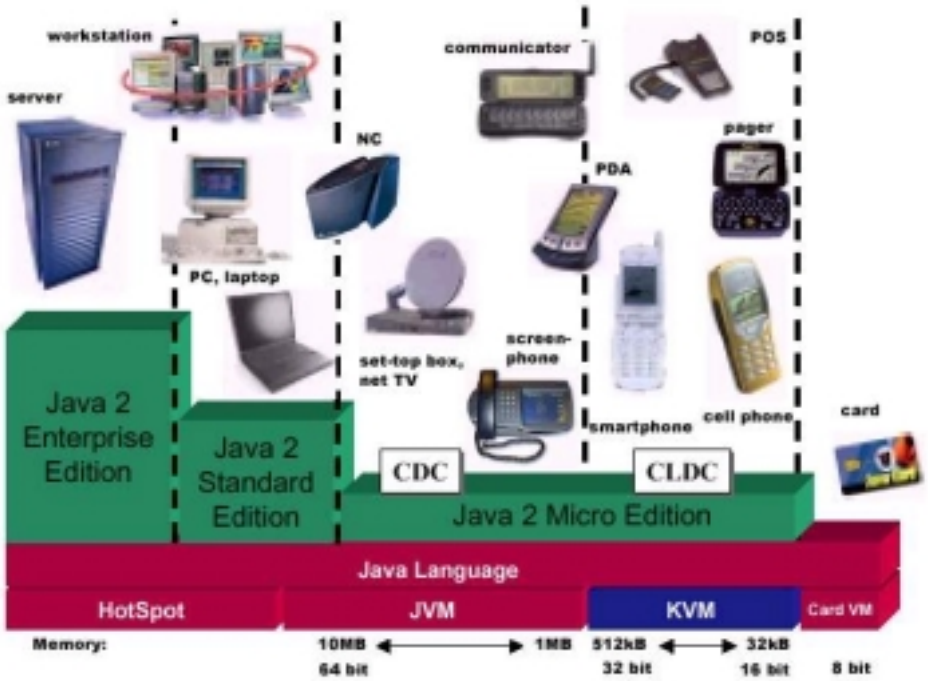


Figure 1: Java editions and their respective target products

Each Java edition defines a set of technology and tools that can be used with a particular product:

- Java Virtual Machines that fit inside a wide range of computing devices
- Libraries and APIs specialized for each kind of computing device
- Tools for deployment and device configurations

This paper looks closer at the Java 2 Micro Edition, its possibilities and limitations.

4 J2ME – Java 2 Micro Edition

J2ME™ specifically addresses the large, rapidly growing consumer space, which covers a range of devices from tiny commodities, such as pagers, all the way up to the TV set-top boxes, appliances almost as powerful as desktop computers.

At a high level, J2ME™ is currently targeted at two broad categories of products:

- Personal, mobile, connected information devices
  In figure 1, this category is represented by the group labeled CLDC (Connected, Limited Device Configuration). Mobile phones and PDAs are examples of devices in this category. These devices have simple user interfaces, minimum memory budgets (starting at around 128 kB) and low-bandwidth intermittent network connections.

- Shared, fixed, connected information devices
  In the figure, this category is represented by the group labeled CDC (Connected Device Configuration). Typical examples of devices in this category include TV set-top boxes, Internet TVs, Internet-enabled screen phones, high-end communicators, and automobile entertainment/navigation systems. These devices have a large range of user interface capabilities, memory budgets in the range of 2 to 16 MB, and persistent, high-bandwidth network connections, most often using TCP/IP.

While connected consumer devices such as mobile phones, PDAs and set-top boxes have many things in common, they are also diverse in form, function and features. Information appliances tend to be special-purpose, limited-function devices. To address this diversity, an essential requirement for J2ME™ is not only small size but also modularity and customizability.
J2ME™ provides a range of Virtual Machine technologies, each optimized for the different processor types and memory footprints commonly found in the consumer and embedded marketplace.

For resource-limited products, J2ME™ technology supports minimal configurations of the Java Virtual Machine and Java APIs that include just the essential capabilities of each kind of device. As device manufacturers develop new features in their devices, or service providers develop new and exciting applications, these minimal configurations can be expanded with additional APIs and with richer complements of Java Virtual Machine features. To support this kind of customizability and extensibility, two essential concepts are introduced by J2ME: Configurations and Profiles[1].

[1] Configurations and Profiles are defined by the Java Community Process[SM] (JCP). The JCP program is the formalization of the open process that Sun Microsystems has been using since 1995 to develop and revise Java technology specifications in cooperation with the international Java community. The JCP fosters evolution of the Java technology in an open, participative manner. The Configurations and Profiles defined by the JCP are expected to take the role of de facto standards [13].

4.1 Configurations

The previous section describes the general concept behind J2ME, but it does not actually discuss in detail what classes are kept from J2SE libraries or what classes that are newly created. This is the purpose of configurations. A configuration is used to customize the Java™ runtime environment for a specific class of devices, and is therefore horizontal in nature, that is, it tries to address a certain family of devices that have similar capabilities. The idea is that an application developed for a particular configuration should run on any device implementing that configuration.

Sun's definition of a configuration reads as follows: A *configuration* is defined as the combination of a Virtual Machine (VM) and "core" APIs that represent an underlying development platform for a broad class of devices [15].

The Java Community Process (JCP) currently defines two configurations, both of them were already named in the previous section:

4.1.1 CLDC (Connected Limited Device Configuration)

The CLDC is particularly important for this paper since it targets devices that have around 128 kB or more of non-volatile memory, 32 kB or more of volatile memory, processors in the range of 16 MHz/16 bit, limited power, and a potentially slow-speed network connection. These are about the parameters we can expect from mobile phones and similar devices.

The main limitations of the CLDC compared to J2SE are:

- No support for JNI (Java Native Interface)
- No weak references supported
- No custom class loaders
- Limited support for internationalization
- No support for finalization
- No thread groups and daemon threads
- Limited error handling
- Simplified sandbox security model
- No certificates supported

There are different Virtual Machines on the market that implement the CLDC. Sun Microsystems' KVM is one of them [2]. KVM is dubbed by many as Kilo Virtual Machine, due to the fact that it works with memory in the range of kilobytes rather than megabytes, as the J2SE JVM does.

4.1.2 CDC (Connected Device Configuration)

The CDC is intended for the emerging higher-end next-generation devices. Typically, these devices run a 32-bit microprocessor/controller, and have more than 2.0 MB of total memory for the storage of the C Virtual Machine and libraries. The CVM is a full-featured, Java 2 Virtual Machine; designed for devices needing the functionality of the Java 2 VM feature set.

The CDC is a configuration that could become interesting for mobile terminals as their capabilities increase [3].

4.2 Profiles

The purpose of a profile is to add domain-specific classes to a particular J2ME configuration. Often, these classes address the user interface capabilities. While a configuration is horizontal in nature, a profile is vertical, and targets devices that are used "in the same way", rather than devices with similar capabilities as with a configuration.

Sun's definition of a profile: A *profile* is defined as a set of APIs for a specific vertical market and relies upon the underlying configuration's capabilities to create new, market-specific APIs [15].

Currently, there are two profiles in the final status of the JCP, the Mobile Information Device Profile (MIDP) and the Foundation Profile. At the moment, MIDP is probably the most relevant profile to application developers since it builds on the CLDC, and targets devices that actually exist on the market today. The Foundation Profile, on the other hand, is built on CDC, and aims at devices with higher processor, memory and network capabilities, than those that are available on the mobile terminal market today.

Besides those two, a number of other profiles are in a working status. To give a complete overview over what there is and what there is to come, these profiles are introduced as well.

4.2.1 Mobile Information Device Profile for the J2ME Platform (MIDP)

MIDP [15] addresses a family of devices that have a common set of characteristics: they have a small display (min. 96x54 pixels), contain a touch screen or keypad, and are capable of communicating via a mobile network with limited bandwidth. In essence, MIDP is built on top of CLDC, and adds classes (APIs) in the following areas:

- Defining and controlling applications - applications written according to the MIDP specifications are called MIDlets and are started, paused, and terminated according to certain conventions
- Displaying text, graphics and responding to user events
- Storing data in simple databases
- Network connectivity via a subset of HTTP
- Timer notifications

In this list, the term MIDlet is introduced. They represent the actual application based on MIDP. MIDlets, together with their support classes and resources, e.g. sounds and pictures, are packed into a jar file. In fact, a number of MIDlets can be contained in a single jar file. This is called a MIDlet suite. The distinctive feature of the MIDlet suite is that all applications that it contains run in the same sandbox. That means that they can invoke each other's public methods and access each other's public members. Apart from the class definitions and resources, the jar file also contains a manifest file, which describes the suite. A jar file is optionally accompanied by an application descriptor. An application descriptor is a text file containing selected information from the manifest file, together with the total size of the jar file. The file extension of this application descriptor is .jad, and below are listed its mandatory attributes:

- MIDlet-Name
- MIDlet-Version
- MIDlet-Vendor
- MIDlet-Jar-URL

- MIDlet-Jar-Size

Optionally, it can also contain the following attributes:

- MIDlet-Description
- MIDlet-Icon
- MIDlet-Info-URL
- MIDlet-Data-Size
- MIDlet specific attributes that do not begin with 'MIDlet-'

The application descriptor is used by the Application Management Software on the mobile terminal to analyze the MIDlets, and to determine whether the terminal is capable of running the application.

4.2.2 Foundation Profile for J2ME™

This Profile [17] serves two purposes. First, it will provide a profile of the Java 2 Platform suitable for devices that need support for a rich, network enabled Java environment, but do not require a graphical user interface. Second, it will provide a base profile for inclusion by other profiles that need to build on the functionality it provides by adding graphical user interfaces or other functionality. This profile is built on top of CDC and requires therefore more advanced devices with the following characteristics:

- 1024k minimum ROM available (application memory requirements are additional)
- 512k minimum RAM available (application memory requirements are additional)
- Connectivity to some type of network.
- No graphical user interface, unless the GUI functionality is provided by an additional profile.

4.2.3 Mobile Information Device Next Generation (MIDP_NG)

This profile [18] is not released yet, but the JCP is working on it. Especially the following points are addressed:

- Domain security model, including signing of applications and verification of certificates
- HTTPS and secure networking
- Network connectivity via sockets and datagrams
- Formal inclusion of OTA (Over The Air) Provisioning (i.e., Recommended Practice 1 for MIDP 1.0) [16]
- Push architecture: external events and messages routed to appropriate MIDlets
- User Interface - extensions to low-level LCDUI to allow greater game functionality and layout control for larger screen sizes.
- A small, efficient XML parser to enable platform-independent data exchange
- Base sound API

4.2.4 Personal Digital Assistant Profile (PDAP)

This profile [19] is also in the process of definition. It is built on the CLDC, and is meant for mobile devices like handhelds or personal organizers. These devices have typically the following hardware parameters:

- No less than 512 KB total memory (ROM + RAM) available for Java runtime and libraries, and no more than 16 MB for storage.
- Limited power, typically battery operated.
- User interfaces of varying degrees of sophistication, but having at least: displays with a total resolution of at least 20,000 pixels, a pointing device, and character input.

To the class libraries of the CLDC, this profile adds mainly the following APIs:

- A display toolkit suitable for limited size and depth displays. In order to support the existing base of developer expertise, this toolkit will be a subset of the Abstract Window Toolkit (AWT).
- Simple persistent data storage for applications, data, and configuration information.

4.2.5 RMI Profile for J2ME™

The J2ME RMI Profile [20] will provide the Java Remote Method Invocation (RMI, [24]) environment for devices with a need of invoking remote services, e.g. printing services, or distributed computing. This Profile is intended to be interoperable with J2SE RMI and has the explicit requirement of providing compatibility with applications developed for J2SE 1.2.x or higher versions (for those J2SE features implemented in the RMI Profile). This profile is also still in a processing stage. The J2ME™ RMI Profile is targeted for devices characterized with the following capabilities:

- 2.5 M minimum ROM available *
- 1 M minimum RAM available *
- TCP/IP connectivity to the network.
- Supporting a complete implementation of the J2ME™ Foundation Profile and the J2ME™ Connected Device Configuration.

* Application and localization memory requirements are additional.

4.2.6 Personal Profile Specification for J2ME

The J2ME Personal Profile [21] provides the J2ME environment for those devices with a need for a high degree of Internet connectivity and web fidelity. This Profile is intended to provide the next generation of Sun's PersonalJava environment [28], and as such has the explicit requirement of providing compatibility with applications developed for versions 1.1.x and 1.2.x of the PersonalJava Application Environment Specification. A closer look at PersonalJava is following in chapter 5.1, PersonalJava. The J2ME Personal Profile provides a profile of the Java 2 Platform, Micro Edition in devices characterized as follows:

- 2.5 M minimum ROM available *
- 1 M minimum RAM available *
- Robust connectivity to some type of network.

- Graphical user interface, providing a high degree of web fidelity and the capability of running Java web applets
- Supporting a complete implementation of the J2ME™ Foundation Profile as mentioned above and the J2ME™ Connected Device Configuration.

* Application and localization memory requirements are additional.

For many developers, the fundamental question will be whether to develop a specific application for the Personal Profile or the MIDP. The most important criteria will then be for what kind of hardware platform they are targeting their application. Higher-end PDAs will support both profiles, but will probably become the domain of the Personal Profile due to its richer graphical capabilities. Devices like mobile terminals with their very small screens and simple user interfaces will be limited to only MIDP. The next criterion will be the features required for building up the applications. As an example, MIDP does not have an XML parser, and RMI is only supported on the CDC configuration. The Personal Profile would have to be used by the developer if either of these features was required in the application.

4.2.7 Personal Basis Profile Specification

The aim of J2ME Personal Basis Profile [22] is to provide a standard J2ME application environment for network-connected devices with relatively limited resources and level of graphical presentation. Such devices were previously served by the "minimum" PersonalJava Application Environment configuration. This Profile will serve as the basis upon which the Personal profile, which has the additional requirements of full web fidelity and legacy PersonalJava application support, can be built on. It is anticipated that the proposed Profile will be useful in markets where a basic graphical interface is required, but a full AWT support is inappropriate, e.g. interactive digital television, Java-based automobile services, information kiosks, etc. The J2ME Personal Basis Profile provides a profile of the Java 2 Platform, Micro Edition in devices characterized as follows:

- 2 M minimum ROM available *
- 1 M minimum RAM available *
- Robust connectivity to some type of network.
- Basic graphical user interface, permitting AWT applications written using only lightweight components.
- Supporting a complete implementation of the J2ME Foundation Profile and the J2ME Connected Device Configuration
- Application and localization memory requirements are additional.

4.2.8 Java Game Profile

This J2ME profile specification [23] will cover nine areas of game development:

- 3D Modeling and Rendering for Games
- 3D Physics Modeling for Games
- 3D Character Animation for Games
- 2D Rendering and Video Buffer Flipping for Games
- Game Marshalling and Networked Communication
- Streaming Media for Games
- Sound for Games
- Game Controllers
- Hardware Access for Games

These nine areas provide the core facilities of a game platform. To build such a platform, the expert group developing this Java specification intends to leverage existing APIs whenever possible. Game developers are also extremely concerned about performance and often look for ways to determine the capabilities of a particular platform This expert group will also investigate metrics that are relevant to games, and look for ways to use those metrics in characterizing implementations.

A goal of the Games Profile is scalability across a range of devices. This profile is targeted at high-end consumer game devices (based on the CDC and Foundation Profile) and desktops (J2SE). However the expert group intends to focus on J2ME platforms with a likely reference implementation on the CDC. The standardizing forum for this profile has only recently been formed.

4.3 Stand-alone Java APIs

Apart from APIs delivered for the J2ME platform with the different configurations and profiles, there is a large number of stand-alone APIs, which address needs of some very specific horizontal and vertical markets. These are also standardized within the Java Community Process, parallel with the configurations and profiles. Unfortunately, they are too numerous to be treated here individually. For developers with specific interests, here is a list of the most interesting APIs:

- Java Multimedia API (JSR 135) – provides basic sound and multimedia capabilities
- Java Speech API (JSR 113) – incorporates speech technology into user interfaces
- Java APIs for Bluetooth (JSR 82) - allows Java-enabled devices to integrate into a Bluetooth environment

For a complete list of APIs developed through the Sun's Java Community process, please, refer to Sun's web page [14].

4.4 Provisioning

An often addressed question is the problem of provisioning of MIDlets. As some manufacturers solve the problem today in their Java enabled phones, applications are downloaded from the Internet to a PC and then transferred to the phone using a serial cable. This solution questions the concept of mobility heavily. As in i-Mode, the user should be able to download applications to the mobile device on the go to keep the wireless experience.

A so called Over the Air (OTA) User Initiated Provisioning Recommended Practice [16] was pre-released in April 2001 by the Java Community Process (JCP). This recommendation is an appendix to the MIDP specification, and describes how MIDlet suites can be deployed over the air, and the functions the devices should provide to support these deployments. The aim of the recommendation is to provide a de facto standard that will support full, interoperable use of MIDP devices. Devices are expected to provide mechanisms that allow users to discover MIDlet suites that can be loaded into the device. In some cases, discovery will be via the devices resident browser (e.g., a WAP browser). In other cases, it may be a resident application written specifically to identify MIDlet suites for the user to download. To be considered MIDP-compliant in terms of the recommended OTA provisioning, a device must be capable of:

- Browsing, or otherwise locating MIDlet suit applications descriptors in the network

- Transferring a MIDlet suite and its associated application descriptor to the device via HTTP or WAP
- Responding to a server 401 (unauthorized) and 407 (proxy authentication required) response to an HTTP request by asking the user for a user name and password and re-sending the HTTP request with the credentials supplied
- Installing the MIDlet suite on the device

How a download of a MIDlet suite to the device should look like, is illustrated figure 2.

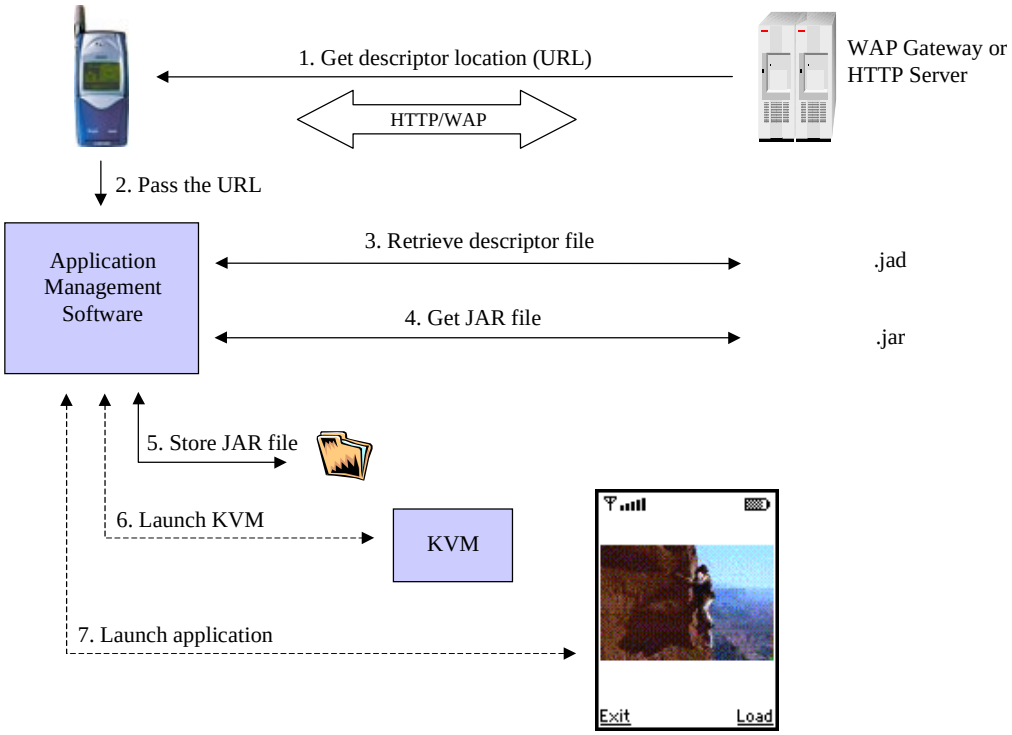


Figure 2: Over-the-air provisioning of MIDlets

1. Applications are found on the network by either the user browsing the web or by a discovery application checking dedicated servers.
2. The descriptor URL is passed to the Application Management Software (AMS)
3. The AMS retrieves the application descriptor from the server.
4. In the application descriptor file, the AMS finds the URL to the MIDlets Jar file. The AMS downloads the Jar file.
5. The AMS stores the Jar file on the device.
6. The user selects the application to run and launches thus the KVM…
7. …and finally the application (a MIDP MPEG player in this case).

4.5 Putting It All Together

The KVM, CLDC, and MIDP, which are the most interesting modules for application developers at the moment, all put together are presented in figure 3.

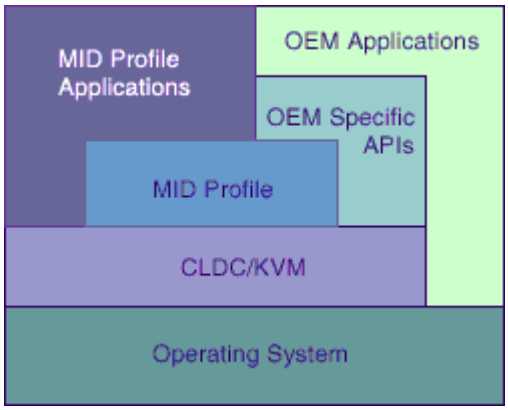


Figure 3: Software architecture of a Mobile Information Device

The figure also introduces the concept of OEM specific APIs. The idea is that terminal manufacturers will, for example, be able to add their own set of classes if they feel that MIDP (or CLDC) do not cover all their needs.

4.6 i-Mode/Java (Japan)

When introducing the profiles defined by the JCP as probable de facto standards, an impressive exception must be named. NTTDoCoMo had launched its i-Mode service in February of 1999 enabling its subscribers to use the Internet on their mobile phones as well as receive and send e-mails. In January 2001, an additional i-Mode service called iAppli was added. NTTDoCoMos customers were able to download and run Java applications for the first time on their Java equipped mobile phones.

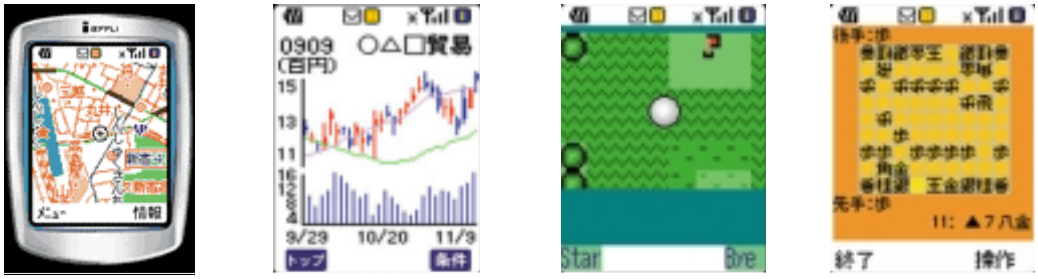
The concept that NTTDoCoMo used to realize its Java service corresponds to a large extent to description of MIDP in this document. iAppli is also built on CLDC, but works with a proprietary profile that has been developed by NTTDoCoMo. This profile stems from the MID profile, but has significant enhancements:

- More complex user interface components
- Support for GIF, whereas MIDP only supports PNG
- Additional sound and animation capabilities
- Secure connections via HTTPS (NOT supported by MIDP)
- S-JIS encoding/decoding (Japanese Language Support)
- DoCoMo proprietary characters

Java on mobile phones proved to be a big success in Japan. With its richer user interface capabilities than WAP and its (although limited) networking APIs, especially gaming applications, stock trading and navigation applications were most welcomed by the customers.

Classical console games were made very addictive by adding hall of fame servers in the networks and thus letting a huge player community compete while sitting on the subway on the way home from work. Users were all of a sudden able to download new levels to a game once they completed the ones locally stored. The first mobile network games were realized (Othello).

To illustrate the variety of possible Java applications for mobile devices, figure 4 illustrates some successful applications from Japan.



Dynamic map (with zooming and sliding functions)

Trading application with (scrolling) news ticker

Animated golf game

Meijin (traditional Japanese chess game)

Figure 4: Screenshots of successful iAppli applications.

5 Related Technologies

When developing applications for limited devices, it is important to not only see J2ME, but also to take a glance at other possibilities. That is the purpose of this chapter. The first three subchapters shall show that J2ME is not Sun's first attempt at making Java work on small devices. JavaCard, EmbeddedJava and PersonalJava already existed by the time that KVM made its first public appearance. After those, a few more subchapters shall give a short overview over technologies that are often named in context with J2ME.

5.1 PersonalJava

PersonalJava [28] is the closest in spirit to J2ME. It defines a subset of the core Java APIs (an optimized version of the Java runtime library) that can be combined with a fully compliant JVM in order to produce a smaller Java runtime environment that is suitable for use on consumer devices. PersonalJava also defines some new APIs, of which the most important is the Truffle Graphical Toolkit, which defines user interfaces with a customizable look-an-feel.

The biggest difference between PersonalJava and J2ME is that the latter supports a much broader set of devices, mostly by allowing specific changes to the JVM by using different Configurations. As such, PersonalJava is really a subset of J2ME and is in fact being redefined as a J2ME profile, the Personal Profile that was introduced in section 4.2.6, Personal Profile Specification for J2ME.

5.2 EmbeddedJava

At the first glance, PersonalJava and EmbeddedJava [29] seem similar. In fact, EmbeddedJava seems even more flexible than PersonalJava, because every class, method, and field of the Java runtime library is optional. This makes it very difficult for 3rd party developers to write applications for EmbeddedJava, since only the implementor of the embedded system is aware of which APIs are included. Unlike PersonalJava or even J2ME, EmbeddedJava is specifically meant to be an unexposed, embedded runtime environment. In other words, only the implementor of the embedded system can write applications for the EmbeddedJava environment, unless rigorous embedded-system specific documentation is provided for the 3rd party developer. Taking this into account, EmbeddedJava does not affect the mobile terminal market, and is not affected by it.

5.3 Java Cards

JavaCard is an architecture for running Java programs on smart cards. A smart card is a set of electronic circuits packaged as a thin device that looks like a standard credit card. Already today, all GSM phones contain SIM cards, which reside on smart cards. SIM cards have a standardized interface, but the internal execution environment is proprietary. Smart cards can, however, host a tiny Java virtual machine together with a limited amount of memory. Java cards with memories of up to 32 KB are available already today. This opens up the possibility of developing JavaCard-compliant applications that would run on smart cards from all vendors. JavaCard specification differs to a large extent from all the other Java platforms, because of the severe constraints on computational power and memory capacity provided by today's smart cards.

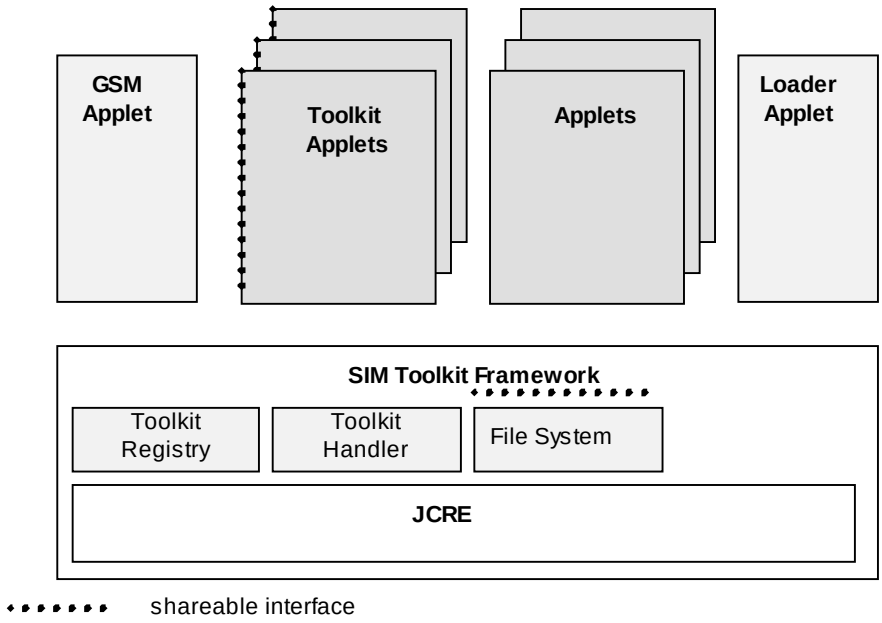


Figure 5: GSM Java Card Architecture

GSM Java card architecture, which is depicted in figure 5, is built on both the JavaCard 2.1 specification [30] and the GSM specifications for SIM-ME interface [31] and SIM application toolkit-ME interface [32]. In this architecture, the SIM toolkit framework together with the GSM

applet carry out the functions of conventional SIM cards, but this time powered by Java. Apart from the mandatory GSM applet, a number of other applets can be loaded into the card's memory. The Toolkit applets differ from the conventional, JavaCard 2.1 compliant, applets by implementing the Toolkit interface. This allows them to be registered in the Toolkit registry and then be invoked by messages and events from the mobile terminal. The toolkit applets can also augment the terminal's menu system, do call control and provide a variety of authenticating and encrypting functions.

The main advantage of the GSM Java Card is the flexibility of the architecture, which supports the well-established standardized GSM interfaces for interaction with the SIM. Java applications can be rapidly developed by a broad programmer community, and JavaCard applications can be used on Java cards from different vendors without reprogramming. Another advantage is the standardized approach to JavaCard application OTA provisioning via SMS [33]. On the other hand, JavaCard environment is suitable only for a specialized suite of applications. It, for instance, does not contain any support for graphics, user interaction (apart from selection from the terminal's menu) memory management, multi-threading and most of the class libraries.

5.4 MExE

Mobile Execution Environment or MExE [26] is a specification first created by ETSI, and now maintained and developed by the 3GPP (3rd Generation Partnership Project). MExE provides a standardized application execution environment for Mobile Terminals. It also provides the ability to negotiate the Terminal capabilities with a MExE service provider, allowing applications to be developed independently of any Mobile Terminal platform. To cater for the wide variety of possible Mobile Terminals, ranging from the small and limited to the larger and more sophisticated, MExE includes a number of different classmarks. A classmark specifies an application environment based on an existing and already proven technology like WAP or Java. MExE is not a new technology in itself, but rather incorporates these existing technologies into an open standard that will allow interoperability between device manufacturers and application developers. There are currently three Classmarks defined in the MExE specification with more to follow in the future.

MExE Classmark 1 – WAP Environment
Classmark 1 MExE devices are based on Wireless Application Protocol (WAP). The Wireless Application Protocol is a world standard for presentation and delivery of information and telephony services on mobile phones as well as other wireless terminals. Supporting mandatory features of WAP, WAP enabled devices provide access to the World Wide Web content without requiring substantial resources from the mobile terminals.

MExE Classmark 2 - PersonalJava Environment
Classmark 2 specifies PersonalJava enabled devices with the addition of the JavaPhone API. The Personal Java application environment is the standard Java environment optimized for consumer electronic devices designed to support World Wide Web content including Java applets. The Personal Java API is a feature level subset of J2SE with some Java packages optional and some API modifications necessary for the needs of small portable devices (for example an optimized version of the Abstract Windowing Toolkit targeted to small displays). JavaPhone is a vertical extension to the Personal Java platform that defines APIs for telephony control, messaging, address book and calendar information, etc.

MExE Classmark 3 - Java 2 ME CLDC Environment
Classmark 3 MExE devices are based on CLDC/MIDP as it is described in this document.

MExE devices are required to support at least one of these classmarks while support of multiple classmarks is optional. The most common configuration, depending on device sophistication, will most likely be a device supporting Classmark 1 and one of the Classmarks 2 or 3.

5.5 Microsoft .NET

Microsoft .NET is Microsoft's platform for XML Web services. XML is a widely accepted standard [27] for data exchange developed by World Wide Web Consortium (W3C). Thanks to XML, applications can be modularized and then loosely connected into functional suites. The objective is to spread tasks over the network in such a way that tasks are always carried out most efficiently and in the most natural place. This approach also allows for an efficient code reuse.

Microsoft .NET adopts the XML model for distributed computing, and provides a complete object-oriented infrastructure needed to develop XML Web services. The corner stones of .NET are the .NET framework and Visual Studio .NET.

.NET framework consists of the common language runtime and class libraries. The common language runtime provides a code-execution environment that manages code targeting the .NET Framework. Code management can take the form of memory management, thread management, security management, code verification, compilation, and other system services. The .NET Framework class library is a collection of reusable classes, or types, that tightly integrate with the common language runtime. The class library builds on the object-oriented nature of the runtime, providing types from which your own managed code can derive functionality. This not only makes the .NET Framework types easy to use, but also reduces the learning curve associated with using a new piece of code. In addition, third-party components can integrate seamlessly with the classes in the .NET Framework.

Visual Studio .NET is Microsoft's multi-language development environment for .NET. Three languages are supported by default - Visual Basic, C++ and C#, but in theory nothing prevents developers from developing .NET applications in any other object-oriented language, for instance, in Java.

Microsoft is a very strong advocate of .NET. A complete portfolio of their upcoming products, including operating systems, corporate servers and smart terminals is going to be .NET enabled. Microsoft even tries to incorporate .NET into MExE by introducing Classmark 4 with .NET environment. Wide acceptance of .NET should be secured through extensive support for 3rd party developers and open specifications of all levels of the .NET infrastructure.

5.6 BREW

BREW (Binary Runtime Environment for Wireless) is a platform that is designed by Qualcomm [34], an American company working in the wireless communications sector. Brew is an application environment that lies between the handset chipset and the applications running on the phone. It makes the phone's functionality available to the application without requiring the developer to have the chip system's source code, or be in a direct relationship with a device manufacturer.

Applications running in the Brew environment are called Applets (not to confuse with Java Applets!) and are written in C or C++. Similar to Java Applets that inherit the Applet specific capabilities from the Applet class, Brew Applets support the IApplet interface. Through this interface, the Brew AEE (Application Execution Environment) gains control over the application.

Brew offers applications a number of APIs allowing them to interact with the resources of the device. A few examples of the interfaces are:

- Memory management
- Notifications
- Network and Sockets
- Download and Billing
- Display
- User Interface and control
- Sound players

Just as it is thought with Java MIDlets, Brew Applets can be downloaded Over the Air (OTA) to the device. This makes the set of applications on a terminal very flexible.

These aspects raise the discussion whether Java and Brew are in a competitive or in a complementary relation. On one hand, Brew allows applications to be downloaded to the device and execute it locally just as Java does. On the other hand, the device must support the Brew platform to use these services whereas a Java application only requires a JVM (or a KVM respectively) no matter on what platform this is implemented. This puts Java in a much more generic context. Qualcomm argues, however, that any Java Virtual Machine can be implemented on top of Brew and therefore is Brew just as generic as Java. How the Java–Brew relation will develop and if Brew is capable to really challenge Java remains to be seen. It is, however, a topic that is worthwhile to keep an eye on.

It is also worth mentioning that KDDI (Japanese operator) intends to make its upcoming cellular phone handsets Brew-compatible within 2002, whereas NTTDoCoMo only supports Brew in its high-end devices.

6 Mobile Application development with J2ME

How Java has evolved to a suitable programming language/technique for mobile terminals, has been discussed to a certain extent so far. To go more into detail and to highlight also a few development aspects, this paper picks up some important features that MIDP offers in the following section. Note, that some of the features described below, such as MIDP features, are in fact more specific to CLDC/KVM, but were not mentioned as such, since MIDP is the interface which the application developer will be dealing with for now.

6.1 MIDP features and their use

6.1.1 Applications/MIDlets

Feature
Applications in MIDP are called MIDlets. MIDlets, or MIDlet suites, can be downloaded via a network and run on the mobile device. The terminal should also provide for permanent storage of MIDlets. In addition to this, MIDlets can be updated or removed.

MIDlets are described by so-called ADFs (Application Descriptor File with .jad extension), which is basically a file that the phone will download before it downloads the actual application

and which tells the J2ME™ environment how big the application is, what name it has, etc. MIDlets themselves are downloaded together with all related resources (images, sound files, etc.) as one single JAR file. MIDlets can be started, paused, restarted and terminated. The terminal may provide the option to run several MIDlets simultaneously.

Use
Downloading MIDlet suites from applications servers is one major advantage of Java on Mobile Terminals. Permanently storing them on the device is another one. Once stored on the phone, a MIDlet can be run anytime and anywhere. For example, games can be played off-line and the high score can be uploaded to the high score server once the device is connected to the network.

6.1.2 Graphical User Interface

Feature
MIDP provides an extensive collection of elements to create rich graphical user interfaces. The APIs are logically divided in high-level and low-level functions. The high-level interface provides easy-to-use elements, and guaranties compatibility across any MIDP device, but is limited in terms of flexibility. The low-level interface is more complicated to use but also more efficient, whish is, in general, what developers of game applications are looking for. Using the low-level API might also result in incompatibilities.

Use
MIDP introduces the basic graphical user interface elements that we know from any PC-based Java application. Image labels, text boxes, list boxes, buttons, ticker lines, all those are included and offer a rich set of elements to choose from. In addition to this, implementing the user interface locally means that from a user point of view the application seems to be more "smooth" and to have a better response time than typical browser based solutions.
MIDP defines functions that allow a Java application to retrieve display characteristics such as screen height and width. This provides a solid mechanism for adapting the look of an application to the terminal to which it is downloaded.

6.1.3 Security

Feature
MIDP supports the Java Sandbox Model very much like the applets that run in web browsers. In this context each MIDlet runs in its own environment and cannot affect other MIDlets. This concept also applies to the Record Store and to network connections. The former restricts access to particular information in the Record Store to the MIDlet that stored this information, whereas the latter ensures that network connections can be setup only to the server where the MIDlet originated.

Use
As later discussed, MIDP doesn’t support secure network connections which is of course a main area of interest when talking about Security. However, by adding third party APIs to an application, HTTPS connections can be provided. Having this level of security on the connection side, and the described level of security from the Sandbox Model on the device side, a good all over security standard can be reached for e-banking and e-commerce applications.

6.1.4 Record Store

Feature
MIDP defines a local and permanent storage object called the "Record Store". It provides a limited amount of memory to any MIDlet and is accessed as a linear byte array.

Use
The Record Store access mechanism is very basic and resembles that of accessing a one-dimensional byte array. This is sufficient enough to store preferences, profiles, game levels, sounds, images. And it's secure: MIDP's sandbox security model ensures that no other MIDlet may access the storage area.

6.1.5 Event Handling

Feature
MIDlets may setup different types of events and event listeners. In fact, most user interactions are handled via events. One particular type of event is the timer.

Use
Events from specific keys can be caught and reacted upon. This allows a good interaction between the user and the application. An interesting feature of MIDP is the possibility to setup timers and catch its notifications (also events). Very nice features can be implemented this way, such as updating stock share information in regular intervals without requiring the intervention from the user. Examples of timer triggered network connections are of course only of real value when the application uses packet switched connections.

6.1.6 Networking

Feature
MIDP provides simple networking through the HTTP protocol. The security model restricts network access to the server from which the MIDlet itself originated. Only terminal-initiated requests are possible, i.e. there is no server push functionality (see discussion in 6.2.3 Push Services).

Use
The networking capabilities of Java introduce the client-server paradigm to the mobile world. This is especially important for services that involve complex calculations or access to large databases. You might want to implement a Java application, which basically provides a nice and responsive user interface, but leaves all the complex computing to the server. All this is possible with MIDP today, but not in a very efficient way.

Because the MIDP implementation of HTTP does not provide server push, the application has to be proactive, and repeatedly check content on the network, for interactive services. There are timer and event objects in MIDP, that make this easily achievable, but it is a very inefficient way of doing it. Not to mention, that all communication with MIDlets has to be carried through HTTP, which imposes a great deal of overhead to each request. Changes to the networking capabilities of MIDP are, however, on the way. The MIDP_NG, presented in section 4.2.3, aims at introducing an SSL layer to HTTP and a sockets model for datagram communication. It will also take care of server push. In the meantime, deficiencies of MIDP will be compensated by existence of proprietary APIs. For instance, Motorola terminals already today provide proprietary APIs for datagram communication via sockets.

6.1.7 Threads

Feature
MIDP offers threading, locking and synchronization.

Use
Threads are a powerful tool for designing user-friendly applications. Operations that are expected to take a significant amount of time (because of network access or demanding computing requirements) are simply started in a separate thread. As a result, the application as a whole remains responsive to user interactions.

6.2 Limitations

Of course, the world is not perfect, and even MIDP does not solve all problems that one faces while implementing mobile applications. The discussion in this section highlights some issues that have been found to be problematic when developing services.

6.2.1 Launching an Application

With MIDP, applications are typically launched through Application Management Software. What this software in detail looks like, is left to the device manufacturer. In the SDKs that are on the market today, the simulated mobile keeps track of a list of applications stored on the device. This would typically take form of a WML or X-HTML list. The user can choose one of the applications by selecting it. How the application is then downloaded and installed on the device is not standardized, but the recommended practice is described in paragraph 4.4.

This mechanism is fine for applications such as games that will typically be started by the users themselves. However, when thinking about e-commerce, it could be very useful, to start, for example, an e-wallet application from within a browsing session. Such interaction between the WAP browser and the MIDlets is unfortunately not supported.

6.2.2 Downloading Large Applications

MIDP does not specify any limitations concerning application size. The limits will probably be set by the operators. Developing communities talk about highly differing numbers. Numbers somewhere between 10 and several 100 kB per application are expected. It was mentioned, that it will be the RAM consumption (not enough heap to allocate to the KVM) that limits the applications size rather than the storage memory. In NTTDoCoMos iAppli service today, an application jar file must not be bigger than 10 kB including all pictures and sounds. This is obviously a severe cut-back and requires serious efforts in code optimization.

In this context, a company named AppStream [8] has come up with a proprietary solution. A streaming server located in the network takes a Java application and divides it into chunks. Downloading an application takes place by "streaming" the chunks of code to the terminal. A special piece of functionality in the terminal is then capable of starting the application before the complete code is downloaded. This technology can achieve very impressive acceleration of the download process (or, to be more precise, the application initiation procedure, but from the end-

user point of view this is perceived as the download time), but it requires adoption of AppStream's proprietary technology both in the network and in the terminal.

6.2.3 Push Services

One thing that is definitely missing in MIDP is the capability to push messages to an application, be it active or not. As mentioned before in the section 'Launching an Application', it would be interesting to let for example an e-commerce page launch your e-wallet application. This topic is addressed in the Java Community Process for the development of MIDP_NG (next generation).

6.2.4 Security

Unfortunately, MIDP supports only very low level security. MIDlets can only open HTTP connections to the server from which they originated. It is also, unlike in i-Mode Java, not possible to encrypt messages on this connection using for instance SSL. It is, however, possible to implement this level of security in the application itself. There are commercial products available that offer APIs for SSL-connections [7].

To improve the overall security model it would furthermore be necessary to authenticate the Java application itself, for example by verifying a signature applied to the JAR file as known from J2SE.

6.2.5 Graphics support

MIDP does not support common graphic formats as GIF or JPEG. To display pictures with MIDP, they have to be transformed to the Portable Network Graphics format (PNG). This is no major drawback, but is mentioned here since it is a disadvantage compared to iAppli's GIF support.

6.3 Testing Mobile Java Applications

Like any other piece of software, Java applications have to be tested to insure usability and functionality under all working conditions. The importance of testing of mobile application is further emphasized by the fact, that the working conditions vary a lot more than they used to for fixed applications. The main areas of testing can be summarized into four bullets:

- Implementation validation
- Platform portability
- Network-independent stability
- Validation of special features

Implementation validation is normally an iterative process carried out by the developer during the implementation phase. Java implementation is facilitated by advanced software development kits (SDKs) for J2ME. All major vendors of mobile equipment provide SDKs that allow 3rd party developers to tailor their applications to their products. By testing simultaneously with a number

of different SDKs, platform portability can be ensured. A number of Java SDKs is listed in section 2 of this paper.

Network-independent stability is an essential area in mobile application testing. Mobile terminals with J2ME are going to be connected to bandwidth limited, intermittent packet networks. Applications for such terminals have to be tested for:

- **Usability under bad network conditions** - applications should always inform the user of their activity and stay in control of the user even if the network connection is temporarily lost.
- **Security of transactions** - consistency and atomicity of banking transactions, e.g. e-payments and money transfers, and other billable transactions, e.g. MIDlet downloads or access to content with added value, should be insured
- **Robustness** - fast recovery from temporary network outages, possibly including implementation of available-bandwidth probing mechanisms is vital.

Testing scenarios for the aforementioned problem areas are simplified for the MIDP by the fact that it only supports HTTP connections. The only challenge is to make the terminal-server interaction as smooth as possible. All other types of communication, e.g. terminal-terminal communication for messaging, will demand a proxy in the network. However, other, more advanced, profiles like MIDP_NG or Personal Profile will also support RMI, XML or sockets for TCP/IP. Some of these new techniques will appear even before the advent of MIDP_NG, in the form of proprietary APIs. With these mechanisms, a variety of new test cases must be invented. The main areas will be the combining of terminal requests with server push, security features and bandwidth-adaptive algorithms for raw datagram services using sockets.

Finally, many viable applications will require some special features to ensure the application's usefulness. The most prominent of those features is undoubtedly server PUSH. Another example is the previously mentioned technique for step-wise MIDlet download developed by Appstream [8].

The best way to test for network-independent stability, and validate the special features, is to combine mobile network simulators with testing on real systems, if possible. Testing on real systems can provide valuable insights into the real application's response times, together with a subjective evaluation of the user's experience. Mobile network simulators, however, must still be used for creating controlled testing environment, for cases when repetitive tests have to be carried out or to achieve extreme conditions, like high network load or network failure. Within Ericsson, it is the Mobile Applications Initiative (http://www.mobileapplicationsinitiative.com) that can help 3rd party developers with both facilities and technical expertise for application testing.

6.4 Conclusion

Java technology is definitely breaking the current frontiers of mobile Internet technologies, and opening up a new world for mobile applications. From an application developer's point of view, the above mentioned limitations should be considered in an early phase of design, as they will have significant impact on the capabilities of the developed application. It is, however, expected that most of these limitations will eventually be addressed and solved. Therefore it can only be stressed how important it is to look into this technology already today.

7 Development Tools

Getting started with developing applications is made easy by a number of development kits. This section introduces a few of those that are available as freeware on the market today.

7.1.1 Suns J2ME Wireless Toolkit

A development kit that is itself written in Java. It uses a Toolbar as a easy-to-use graphical user interface, and lets MIDlets run on different visionary emulated phones with different screen qualities. This kit [9] is tightly coupled to Suns JForte, a Java IDE that is freely available on the net.

7.1.2 Motorola J2ME SDK Components Developer Edition

Motorola [10] was one of the first companies coming out with an SDK for J2ME. It enabled developers to run MIDlets on the Java enabled Motorola phones (i85s and i50sx). For unknown reasons, Motorola is at the moment not offering the download to the public anymore.

7.1.3 Zucottos WHITEboard SDK

This is a freely available [12], fully-grown development environment written in Java featuring debugging, code formatting, image editing, Java Documentation generating, packaging and more. It resembles very much Sun's JForte IDE. When using the default installation, Motorola's i85s is emulated.

7.1.4 i-JADE (phone emulator for iAppli)

Zentek [13] offers a very simple emulator for NTTDoCoMos Java phones.

7.1.5 Java Card 2.1.2 Development Kit

Based on the Java Card 2.1.1 Specifications, it features a series of new or enhanced tools, aimed at improving both development experience and platform security:

- New off-card verifier
- Executable implementation of the Java Card Runtime Environment (C-JCRE)
- Improved converter

8 References

1. Mobile Applications with Java, White Paper about Java in NTTDoCoMos i-Mode (http://inside.jp.ao.ericsson.se/ra/java/java.asp)
2. CLDC and the K Virtual Machine (http://java.sun.com/products/cldc)
3. CDC and the C Virtual Machine (http://java.sun.com/products/cdc/)
4. Mobile Information Device Profile (JSR-37), JCP Specification (http://java.sun.com/products/midp)
5. Java 2 Platform Micro Edition Technology for Creating Mobile Devices, White Paper (http://java.sun.com/products/cldc/wp/KVMwp.pdf)
6. MIDP APIs for Wireless Applications, A Brief Tour for Software Developers (http://java.sun.com/products/midp/midp-wirelessapps-wp.pdf)
7. Legion of the Bouncy Castle, company offering an encryption API for J2ME™ (http://www.bouncycastle.org)
8. AppStream, company offering a download enhancement solution (http://www.appstream.com)
9. J2ME™ wireless toolkit and JForte by Sun, (http://java.sun.com/products/j2mewtoolkit/, http://www.sun.com/forte/ffj/)
10. Motorola's Developer Community (https://commerce.motorola.com/idenonline/ideveloper/index.cfm)
11. Java Card 2.1.2 Development Kit (http://java.sun.com/products/javacard/dev_kit.html)
12. WHITEboard SDK by Zucotto Wireless (http://www.zucotto.com/whiteboard/index.html)
13. Zentek Technology, company offering a SDK for iAppli (http://www.zentek.com)
14. The Java Community Process Program, (http://java.sun.com/aboutJava/communityprocess/)
15. JSR #000037, Mobile Information Device Profile, (http://java.sun.com/aboutJava/communityprocess/jsr/jsr_037_mid.html)
16. Over The Air User Initiated Provisioning Recommended Practice (Addendum to MIDP)
17. JSR #000046, J2ME™ Foundation Profile, (http://java.sun.com/aboutJava/communityprocess/jsr/jsr_046_j2mefnd.html)
18. JSR #000118, Mobile Information Device Next Generation, (http://java.sun.com/aboutJava/communityprocess/jsr/jsr_118_midpng.html)
19. JSR #000075, PDA Profile for J2ME™, (http://java.sun.com/aboutJava/communityprocess/jsr/jsr_075_pda.html)
20. JSR #000066, J2ME™ RMI Profile, (http://java.sun.com/aboutJava/communityprocess/jsr/jsr_066_rmime.html)
21. JSR #000062, Personal Profile Specification, (http://java.sun.com/aboutJava/communityprocess/jsr/jsr_062_pprof.html)
22. JSR #000129, Personal Basis Profile Specification, (http://java.sun.com/aboutJava/communityprocess/jsr/jsr_129_pprof.html)
23. JSR #000134, Java Game Profile Specification, (http://java.sun.com/aboutJava/communityprocess/jsr/jsr_134_pprof.html)
24. Java Remote Method Invocation, (http://java.sun.com/products/jdk/rmi/)
25. Personal Java Application Environment, (http://java.sun.com/products/personaljava/index.html)
26. MExE Forum (http://www.mexeforum.org/)
27. Extensible Markup Language (XML) (http://www.w3.org/XML/3GPP)
28. PersonalJava Application Environment, (http://java.sun.com/products/personaljava/)
29. EmbeddedJava Application Environment, (http://java.sun.com/products/embeddedjava/)
30. Java Card Technology, (http://java.sun.com/products/javacard/)
31. GSM 11.11 – Specification of the Subscriber Identity Module – Mobile Equipment (SIM – ME) interface.
32. GSM 11.14 – Specification of the SIM application toolkit for the Subscriber Identity Module – Mobile Equipment (SIM – ME) Interface
33. GSM 03.48 – Security Mechanisms for the SIM application toolkit
34. QUALCOMM, (http://www.qualcomm.com/)

© Ericsson Radio Systems AB - July 7, 2001
With comments and suggestions, please, contact the authors.

Christian Schläpfer (christian.schlapfer@era.ericsson.se)
Michal Kubik (michal.kubik@era.ericsson.se)
Guido Zavagli (guido.zavagli@nrj.ericsson.se)