UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Andres Teder

The Problem Sets of Java Micro Edition Technology

A thesis submitted for the degree of
Magister Scientiarum in Computer Science

Supervisor: Prof. Jüri Kiho

Tartu 2006

The Problem Sets of Java Micro Edition Technology

Master's Thesis

Andres Teder

Abstract

Mobility can be considered one of the most important keywords of the present era making all research topics in this area interesting and important at the same time.

With the advent of powerful new mobile devices which utilise more capable wireless networks, carriers, content providers and handset manufacturers are bringing to market a range of new mobile software applications. Mobile handhelds of today carry different kinds of applications ranging from spreadsheet editors to TV-players.

One of the most common and the *de facto* development technology for the small mass-market handsets is Java Micro Edition, a technology that enables developers to address the biggest number of different mass-market devices. Unfortunately, it is very commonly the case that teams working with mobile software under-estimate or have false anticipations of the complexities, limitations and problem sets of Java mobile software development. One reason for false anticipations might be the special nature of mobile computing as well as a general lack of previous experience with mobile technologies. Another reason may be the fact that most of the available literature on the subject discusses the capabilities of the technology and development practices but fails to emphasise the diversity and show a larger picture of the problems and complexities that the development efforts may suffer.

The aim of this thesis is to give an overview of the problems and limitations of the Java Micro Edition software development. It also presents a framework of further knowledge building and addressing these issues.

Acknowledgments

I would like to thank all the people who supported to accomplish this thesis.

The information for this thesis has been collected over several years of active involvement in the rapidly growing mobile field. I would like to thank all my colleagues and good friends who have been by my side when exploring this interesting field.

I would also like to thank my supervisor Prof. Jüri Kiho from the University of Tartu for his encouragement to proceed with this research area as well as his support throughout my studies in the university.

Finally, I am very grateful to my mother who has always been supportive of my academic pursuit and whose patience and encouragement have helped me through the most difficult stages of this work.

Table Of Contents

1. Introduction

The history of the first mobile phones dates back to the 1940s. The predecessors of the first generation of mobile phones were radiotelephones which connected to the public switched telephone network. The first generation of mobile networks (1G) appearing in 1980s and especially the introduction of second generation (2G) networks have been the disruptive force catalysing the innovation and expansion of the cellular communication field. The innovation has reached a stage where third generation networks (3G) have been widely deployed [21]. The quick advancement of the cellular infrastructure and innovation in other fields of technology have been able to turn the once brick-size voice-only devices into small portable computers.

Today mobile phones have become a pervasive part of our everyday lives. While most of the phones in 1990s were mainly used for voice calls, sending out short messages and sometimes also as personal organizers, the recent innovations in the field have made it possible to expose mass-market mobile phones as open platforms for third party applications. Phones of today represent small personal computing devices and run applications ranging from games and television-players to business and productivity tools. An interesting development can be seen with the introduction of latest handsets that allow the user to switch off the wireless radio and use the phone just for listening to the music for example[1], not being connected to the mobile network at all. These handsets might be the sign of a new emerging trend that puts less emphasis to the *phone* part in the mobile phone and more emphasizes the *mobile* as a compact computing platform.

Mobile computing is fundamentally different from the more traditional Web-centric computing architectures. Although mobile networks are constantly improving, the lack of consistent bandwidth is still a reality in today's mobile networks. As a result mobile phones operate in a highly disconnected and varying environment. The phones are physically constrained and have limited storage capacity, they also have an extremely personalised nature and user expectance of the quality of interaction is very high.

The emergence and expansion of the mobile market has created many interesting opportunities for innovation for third party developers to bring applications to this platform. The challenges that mobile developers face are often misunderstood and result in failures of the projects in different ways:

[1] An example of such a device is Sony-Ericsson W810i.

- Fragmentation, implementation errors and other technical problems tend to increase the effort necessary for delivering an application to the market. As a result project timelines are extended and development costs increased.
- Management of user expectations is difficult – end-users have very high expectations of mobile applications despite of the varying environment the applications work in.
- Marketing mobile applications is difficult – many business models are still immature and time to market can take a while for new start-ups. Delivering a mobile application usually also requires the cooperation of several parties: development company, porting service provider, publisher, aggregator and distributor.

This thesis will give an overview of the development and technical aspects of Java mobile technology and points out several misconceptions that can be dangerous to development efforts.

The methodology used in this thesis can be divided into three separate steps.

- The first is to introduce Java mobile technology and show its future directions.
- The second is to position Java mobile technology in the current wireless development landscape by comparing it with other mobile technologies. This would present a reference for choosing the platform for mobile development.
- The third step is to explore the technology from a point of misconceptions and limitations that developers find troubling about the platform. The problems can be divided into two: the more general problem of fragmentation and the limitations of the Java Micro Edition platform. The data and prioritization of these pitfalls comes from author's practical experience, public developer forums and the trends and solutions the industry is using to reduce the complexity.

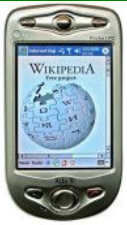
The layout of this thesis is organised as follows.

- The second chapter covers the background of mobility and its importance.
- The third chapter introduces Java mobile technology and its most important concepts.
- The fourth chapter positions mobile Java in the wireless landscape.
- The fifth chapter gives a coverage of fragmentation in the mobile space.
- The sixth chapter takes a look at several practical aspects of developing and running mobile Java applications.
- The seventh chapter concludes the thesis.

2. Background

2.1. Mobile devices

Generally, all devices that have something to do with mobile or wireless are classified as mobile devices. A wide range of mobile devices exist on the market today. These devices are aimed at both consumer and enterprise users and vary greatly in different capabilities. There are many ways to classify the devices, one of the ways is given below [18, 21]:

Pagers and RIM devices	Cellular phones	PDA devices and smart-phones	Tablet PCs	Laptop PCs
				

The devices that are one of the most widespread consumer oriented mass-market range devices, cellular phones, are the focus of this thesis.

In the early 1990s the core functionality of mobile phones was centred around voice services. The user interfaces were basic, the software platforms were usually closed for third party developers and the technology itself was very immature.

The mobile phones of today have evolved significantly from hardware as well as software perspective. The hardware progress has turned mobile phones into compact devices with colour displays and integrated cameras that have a wide set of accessories ranging from GPS readers to Bluetooth headsets. The software on modern mobile phones is usually a collection of various utilities from music and video players, games, calendars and to-do lists to spreadsheet editors.

The conventional definition of a mobile phone as a pure communication device has become somewhat misleading. Firstly, a mobile phone has traditionally been only a communication device, but nowadays phones have different non-communicational features which make them also a kind of smart-phones. Secondly, the connectivity part in a mobile phone might not be as important for the upcoming device releases.

A mobile phone with a music player, for example, could just be used in "offline" mode with all communication features turned off and only the music player (non-communicational functions) active.

2.2. The significance of the mobile platform

At the time of writing this thesis, the population of the world is reported to be 6.5 billion people [24, 25] and the number of mobile phone users more than 2 billion [26]. This means that about one in every three people has a mobile phone.
Analysts have predicted that there will be 3 billion mobile subscribers by the years 2009-2010, the main growth expected to come from emerging markets such as Africa, India, Russia and China [26].

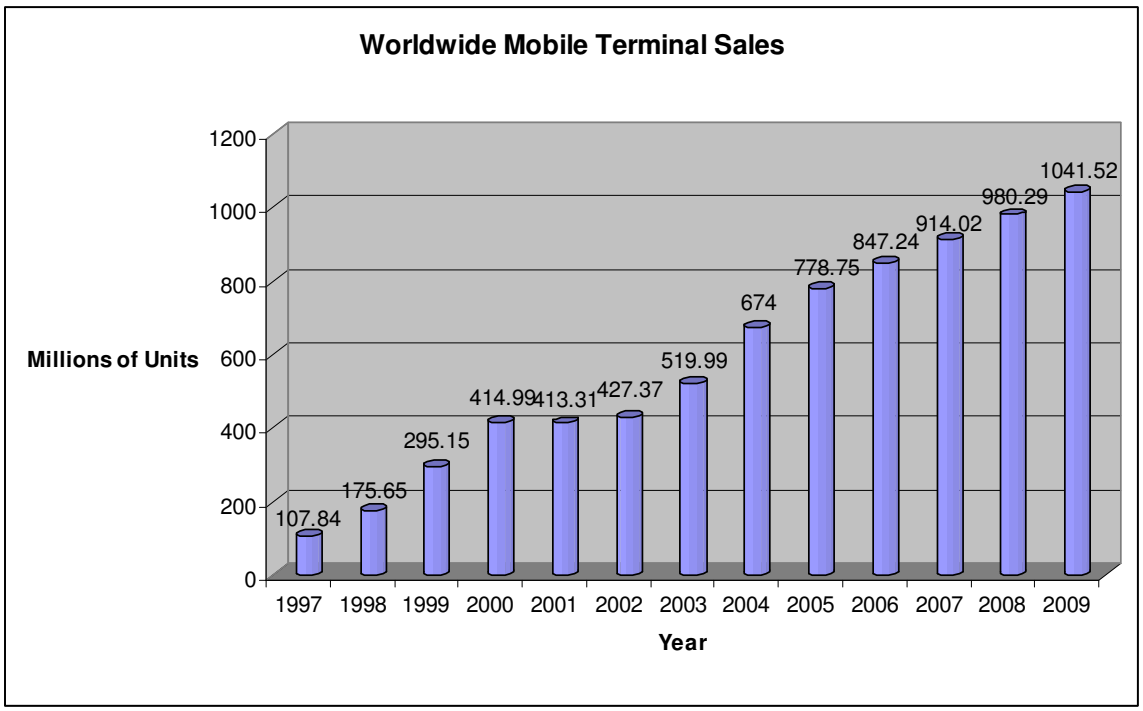


Figure 1 - Graph representing number of mobile terminal sales, based on [31]

It can be seen that mobile platform will become (or has become) the most accessible computing platform.
While countries with advanced economies are already using mobile phones in very different everyday activities, these devices can also have a tremendously positive impact on the economically disadvantaged nations. An experiment in Bangladesh where mobile entrepreneurs in a rural area purchased mobile phones to provide services to other villagers resulted in a micro-enterprise platform benefiting the

service providers as well as their clients [27]. The rural villagePhone idea has been picked up by Garmeen Foundation USA [28], a non-profit organization that uses innovative technology to fight global poverty and bring opportunities to the world's poorest people. A 2005 study by the Centre for Economic Policy Research found higher rates of economic growth in developing countries with high mobile phone penetration [27].

Mobile platforms have clear advantages over standard computing platforms such as personal computers by supporting access to extremely personalized information *anytime anywhere* and allowing for the first time in history a person's information access to be disassociated from the environment. Conflicting tasks such as going to see a doctor, booking flights for a vacation and checking the stock market can be done at the same time. While waiting in the doctor's office one can book flights online and also receive alerts of different movements of the prices in stock market.

Two primary benefits that rise from mobility for businesses can be outlined as follows [32]:

- Mobile users can make faster decisions. For example, users who can access corporate e-mail, documents and other data from anywhere, will be able to make faster decisions which will lead to greater customer satisfaction and increased revenue.
- Mobile users benefit from improved data accuracy. Business users can access corporate databases which have up to date information wherever they are. In addition to information access they can also capture information in the field, an operation that traditionally has involved filling handwritten forms. This leads to reduced cycles, labour and cost reduction.

2.3. Barriers faced by mobile development efforts

Mobile developers and development companies are finding it increasingly difficult to provide quality applications to satisfy the quickly growing market demand. From the development companies side, a considerable effort has to be made to plan resources as mobile projects tend to become more costly than several other types of software projects mainly due to the innovative (often immature) nature of the technology and the market.

From developers side, mobile application development requires unique skills extending beyond simple programming knowledge. Mobile developers often need to

have a good understanding of the technology's business aspects, the needs of carriers, phone manufacturers and end-users as well as the technical aspects of the specific technology chosen for development. It is also very common that mobile applications need to be interfaced with other enterprise systems which requires even a more diverse skill set from developers.

Wireless carriers have a unique power position in the mobile ecosystem. They provide access to wireless data services (and applications) and, more importantly, own the billing relationship with the customers. In order to stay successful it is a necessity for wireless carriers to differentiate from competitors. One of the key ways of differentiation is to provide unique value-added services, an example of which can be downloadable Java applications. At the same time the different complexities involved with the acquisition, management and delivery of these services must be managed. Carriers have to deal with a variety of handsets having different technical capabilities and user interface characteristics which are results of the quick evolution of the handset technology space. The effort required to port and verify the content services for different handsets is growing more expensive as the complexity is increasing and has thus formed one barrier for market development. Although carriers have provided different specifications that handset manufacturers must match as a way to address the issue, mobile technologies are changing and developing in a rapid manner and the fragmentation and complexities will still continue to increase.

End users have been very open to new mobile services, but their optimism has been clouded by applications failing to properly work on handsets. While the range of applications might be broad, especially in the area of gaming, many of the applications can be of poor quality or fail to correspond to the user expectations.

These kind of barriers seriously limit the mobile market to reach its full potential.

3. Java Micro Edition technology

3.1. Introduction

Java Micro Edition (Java ME, formerly also known as Java 2 Micro Edition[2] or J2ME) technology was introduced to the public in 1999 as a highly optimized Java runtime environment geared for consumer devices. It was the aim to apply the paradigm shift that Sun had introduced with Java to the small devices sector. Java is a programming language that does not have pointer types or complicated memory operations, is object-oriented, secure and has a widespread platform support. Java ME was set to target a wide range of devices from set-top boxes to "smart-cards" and automobiles. It forms a separate platform edition alongside other platforms such as Java Standard Edition (Java SE), a complete environment for desktop and server applications, and Java Enterprise Edition (Java EE), an environment for enterprise applications.

Figure 2 - Java Platform Editions

In the cell-phone sector, most of the phone manufacturers and carriers have realized the potential of the Java ME technology that offers a platform able to utilize the

[2] The naming convention change was communicated by the Sun Microsystems marketing department at the JavaOne 2005 conference.

already widespread Java developer skills and at the same time provide a portable application environment which spans different device categories.

At present over 180 mobile operators have deployed Java ME and over 1 billion Java ME devices have been shipped [11, 12]. Handset manufacturer support is strong - for example all Nokia Developer Platform devices are Java ME enabled and all together there are reported to be more than 86 Java ME licensees [1, 12]. The number of handset models supporting Java ME has exceeded 635 and unit shipments expected to exceed 600 million in the year 2006 alone [12].

The number of mobile Java downloads globally has exceeded 23 million per month and there are more than 45 000 Java applications on the market [44].

3.1.1. Evolution

Java Micro Edition technology is officially developed through a process called *Java Community Process* (JCP) that was established in 1998 [4]. The process can be described as follows [3, 4]: for a given task a set of standard APIs is defined by a standards committee. The standard API specifications are called *Java Specification Requests* (JSR). Individual vendors then provide competing libraries that implement those APIs. The application code using the API is completely decoupled from the specific implementation provider.

This kind of approach minimizes the developer's cost of learning and improves code portability, it also protects the freedom of choosing vendors. The core concepts of the JCP are:

- All specifications are defined collaboratively by the industry.
- All specifications go through a well-defined review process.
- All specifications have a reference implementation and a *Technology Compatibility Kit* (TCK).

A diagram explaining the process is shown below.

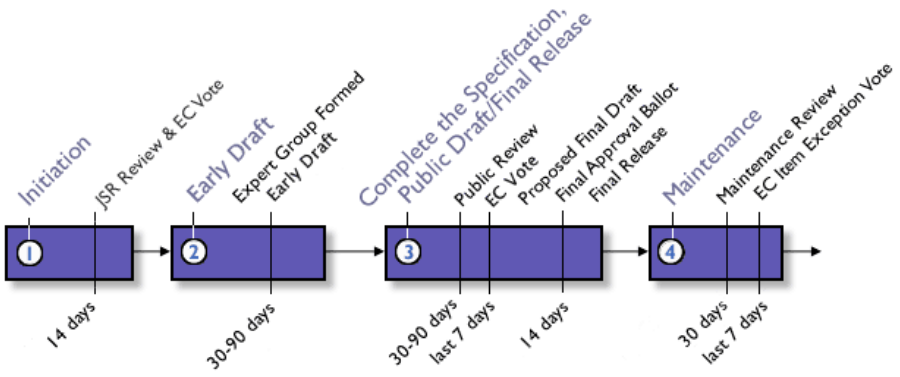


Figure 3 – Abbreviated JCP timeline view [4]

At present, there are 262 JSRs which address Java EE and Java SE technologies and 79 JSRs which can be categorized to address Java Micro Edition technology[3].

3.2. Architecture

Java ME technology has a specific architecture that makes the platform a good fit for a variety of small devices which have been designed for different purposes and have different features. The core components of the technology are *configurations*, *profiles* and *optional packages*.

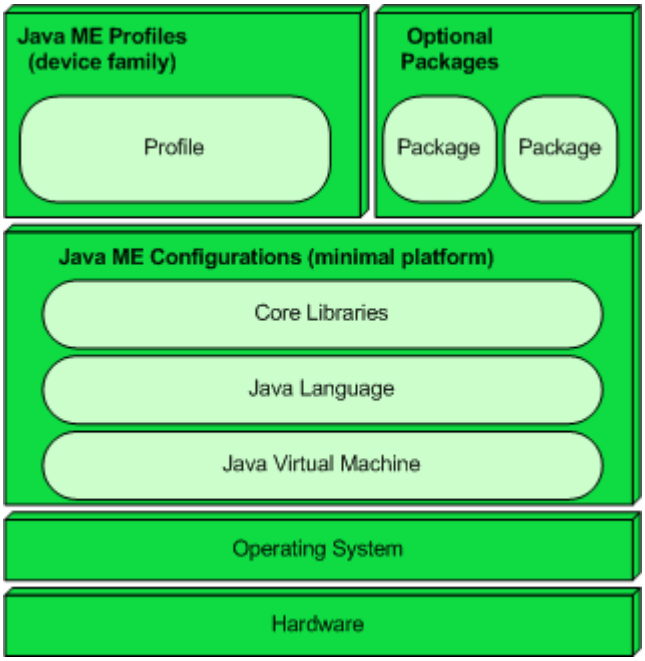


Figure 4 - High level Architecture of a Java Micro Edition environment

[3] May 2006, based on [4]

The different layers of the architecture are the following:

- The lowest layer of the architecture is the hardware of a mobile phone.
- On top of the hardware layer is the native system software which includes the operating system and native libraries.
- The layer on top of the operating system is called a *configuration*. Configuration is a specification that details a Java Virtual Machine (JVM) and a base set of APIs which can be used on a certain class of devices.
- On top of the Configuration layer there are [19]:
    - *Profiles* which complement a configuration specific to a device category. Profile is a set of APIs that further define the life-cycle model, user interface, storage and access to device specific properties.
    - Optional packages that extend the *Configuration* and/or *Profile* with functionality for specific application requirements and offer standard APIs for using both existing and emerging technologies.
    - OEM specific classes which address a certain functionality of a certain device (or device family) and are not specified through JCP.

3.2.1. Configurations

Configurations "limit" the Java environment to a certain target platform's capabilities. Two configurations have been defined for the Java ME platform.

- *Connected Limited Device Configuration* (CLDC) – aimed at the smallest wireless devices with the biggest mass-market appeal. These devices typically have 160 kB or more memory and slow 16 or 32 bit processors. Only a limited small subset of Java SE libraries are available on the CLDC platform.
- *Connected Device Configuration* (CDC) – aimed at more capable wireless devices that have at least 2MB of memory and 32 bit processors. CDC supports a full JVM and most Java SE libraries.

Platform	**Device group**	**Example devices**
CDC platform [9, 10]	High-end consumer products	Smart-phones, PDAs, set-top boxes, medical equipment etc.
CLDC platform [5, 6]	Lower-end consumer products	Mobile phones, modems etc.

This thesis will focus on the CLDC platform which can be considered the most pervasive of the 2 categories in consumer devices sector. The CDC platform requires much more from the device than the CLDC and is not affected by many of the problems that are common to CLDC implementations. There are only a small number of CDC handsets available on the market today compared to the almost pervasive CLDC deployments.

3.2.2. Profiles

Profiles address a certain vertical market[4] segment or a device family. The main goal is to guarantee interoperability within this vertical market segment by defining the Java platform standard for that market. A device can support multiple profiles and the libraries in the profile are more domain specific than those defined on the configuration level. Profiles define complete, and usually self-contained, application environments.

Devices in the consumer market are not so homogeneous that end users can expect or require universal portability, ideally the portability means portability in the same device family.

An example of a profile is the user interface – as mobile phones have small screens, they do not need the full set of Java AWT libraries since the screens are just too small to display all elements. The most widespread CLDC-based profile is called *Mobile Information Device Profile* (MIDP) and targets mobile phones.

3.2.3. Optional Packages

Optional packages are layered on top of configurations or profiles and typically contain functionality that is independent of any vertical market segment or device family. They do not define a complete runtime environment. The main goal of optional packages is to extend the runtime environment to support device capabilities that are not universal enough to be defined as part of a profile or that need to be shared by different profiles.

[4] A vertical market segment is a set of products that share a common unique and specific nature that differentiates them from other segments, an example of different segments of mobile devices can be: PDAs and mobile devices

3.3. Connected Limited Device Configuration

The CLDC specification 1.0 (JSR 30), was released in May 2000. The next release of the specification 1.1 (JSR 139) was released in March 2003.
As general characteristics, the target devices for CLDC have 160 (192 kB in CLDC 1.1) to 512 kB of memory devoted to the Java platform, 16 to 32 bit processors, some form of connectivity and low power consumption.
The CLDC specification addresses the following areas of the on-device Java environment:

- Target device characteristics
- Security model
- Networking and input/output mechanism
- Language differences and core libraries
- JVM differences
- Internationalization

The specification does not define any requirements in the areas of high-level application model, application lifecycle management, event handling or user interface support - these are all defined by profile specifications.

3.4. Mobile Information Device Profile

The MIDP specification 1.0 (JSR 37) was released by the JCP in September 2000 [7]. The next version of the specification, MIDP 2.0 (JSR 118), was released in November 2002 [8]. Work has been started on the version 3.0 of the specification (JSR 271) which is expected to be released by end of 2006 [4].
The MIDP can be considered the most popular of the Java ME profiles since it is implemented on a huge number of Java-enabled handsets [16, 17].
The main goal of the MIDP specification is to define an architecture and the associated APIs to enable an open development environment for mobile information devices.
The MIDP specification defines:

- Application delivery and billing model, Over-The-Air (OTA) Provisioning
- Application lifecycle and management
- End-to-end security through HTTPS and secure sockets
- Application signing model for added security
- Push architecture (since MIDP 2.0 specification)

- Networking support (based on the HTTP protocol and CLDC)
- Persistent storage support
- Sound support
- User interface (Limited Capability Device User Interface, LCDUI)
- Miscellaneous classes such as timers and exceptions

The MIDP specification does not define system-level APIs or low-level security – both are considered to be out of the scope of the specification.

Applications written for the Java ME platform and Mobile Information Device Profile are called MIDlets[5].

3.5. Important Optional Packages

The configuration and the profile layers have their boundaries and much of the additional functionality that is required by real-world applications is specified in optional packages. The number of optional packages standardized through JCP is increasing rapidly and there are a number of optional packages available specifying functionality in the areas of multimedia, messaging, location awareness, communication technologies and others. After an optional package has been released by the JCP it can be implemented by device manufacturers under certain licensing conditions. Because implementing optional packages is time-consuming and often related to phone manufacturers' strategic decisions of product line development, the interval between the final release of an optional package JSR and the time it will be available to developers in real-world devices can be more than several months.

A list of important optional packages is given in the following table, many of the packages are available on today's devices although some of them are yet to be released by either the JCP or phone manufacturers. Varied sets of optional packages on different phones is one of the causes of the widespread fragmentation issue.

The increase in the number of optional packages is a good example of the quick expansion of the Java ME platform. The optional packages JSRs define the capabilities, building blocks of the architecture of the future Java ME devices.

[5] More specifically a class is a MIDlet when it extends the `javax.microedition.MIDlet` class

JSR	Name	Description
120, 205	Wireless Messaging API 1.0, 2.0 (WMA)	Support for wireless messaging including SMS. Version 2.0 of the specification also adds MMS capability.
135	Mobile Media API	Adds audio/video controls and support for streaming media.
172	Web Services Specification	Adds support for web-services related XML processing.
179	Location API	Adds support for location tracking of wireless devices.
82	Bluetooth API	Adds support for Bluetooth communication protocol.
177	Security and Trust Services API	Adds support for various security-domain services and allows the application to access smart-cards.
184	Mobile 3D graphics API	Adds support for 3D graphics.
180	SIP API	Adds support for the Session Initiation Protocol (SIP) which is commonly used in multimedia and VoIP applications.
190	Event Tracking API	Standardizes application event tracking on a mobile device and the submission of these event records to an event-tracking server via a standard protocol. Early draft review was started in March 2006.
226	Scalable 2D Vector Graphics API	Adds support for rendering scalable 2D vector graphics, including image files in W3C Scalable Vector Graphics (SVG) format.
229	Payment API	Adds support for initiating mobile payment transactions from Java ME applications.
253	Mobile Telephony API	Allows applications to access call-related functionality.
257	Contactless Communication API	Adds two packages to support contactless communication, one for bi-directional communication and the other for accessing read-only information. The Proposed Final Draft of the Specification was released in April 2006.

JSR	Name	Description
211	Content Handler API	Allows applications to invoke third party content handlers and/or register themselves as content handlers for different content types.
75	PDA Optional Packages for the J2ME Platform	Adds two separate optional packages for features commonly found on PDAs and other J2ME mobile devices, one for accessing PIM data and another for accessing file systems.
234	Advanced Multimedia Supplements	Adds advanced multimedia functionality which is targeted to run as a supplement in connection with MMAPI (JSR-135) in a Java ME/CLDC environment. More specifically, features added are: access to camera specific controls, access to radio and other media sources as well as advanced audio processing and media output direction.
238	Mobile Internationalization API	Adds an optional package containing an application program interface that allows MIDP application developers to internationalize their MIDP applications.
256	Mobile Sensor API	Adds a general Sensor API that extends the usability and choice of sensors for Java ME applications. It defines generic sensor functionality optimized for the resource-constrained devices like mobile devices.
258	Mobile User Interface Customization API	Allows software developers to query and manipulate the appearance of a mobile device user interface and introduces a common exchange format for appearance data to be used by content developers and distributors. Was passed to public review on the 9th of May 2006.
259	Ad Hoc Networking API	Defines an API that enables communication between mobile devices in a peer-to-peer ad-hoc network environment. The Early Draft Review stage was finished on the 1st of March, 2006.

JSR	Name	Description
266	Unified Message Box API	Defines an API to access and manage the message boxes of mobile devices and their content. The Early Draft Review stage was finished on the 14th of May, 2006.
272	Mobile Broadcast Service API for Handheld Terminals	Defines an optional package that provides functionality to handle broadcast content, e.g. to view digital television and to utilize its rich features and services. Finished Early Draft Review stage on the 19th of March, 2006.
280	XML API for Java ME	Provides a general purpose XML API for the next generation of mobile devices. Entered Early Draft Review stage on the 12th of May 2006.

3.6. Java Technology for the Wireless Industry

The MIDP specification is the most widely deployed Java ME profile. The downside of the 1.0 version of the MIDP specification is that it defines a rather simplistic environment which is not functional enough for the needs of many applications. The lack of functionality has driven handset manufacturers to define their own APIs for extended functionality in the areas of image manipulation, phone backlight flashing and sounds. At the same time the JCP has also released a number of new optional package specifications.

The result is a heavily fragmented space where devices implement a different collection of APIs and also have differences in the implementations of these APIs.

The Java Technology for the Wireless Industry (JTWI) specification (JSR 185) [40] was released by the JCP in July 2003. The main role of the specification is to reduce fragmentation and address various portability problems. The JTWI defines a component architecture which consists of fours JSRs:

- CLDC, version 1.0
- MIDP, version 2.0
- WMA, version 1.1
- MMAPI, version 1.1 (optional)

In addition to defining a standard architecture with components that provide a comprehensive set of functionality, JTWI also defines a number of other clarifications

and reduces optional clauses. The defined clarifications concentrate on size limitations, user interface, messaging, security model and push registry related functionality.

3.7. Mobile Service Architecture

The JTWI specification was the first attempt to address a number of problems that the emerging mobile Java industry has been facing. The *Mobile Service Architecture* (MSA) defined by JSR 248 for CLDC devices (and JSR 249 for the CDC platform) addresses the same issues as JTWI and is a continuation of the effort [39]. The specification efforts are lead by *Nokia* [62] and *Vodafone* [63] and the expert group represents the wireless industry across the world.

The CLDC-oriented specification reached a Proposed Final Draft stage in May 2006 [4]. The sister-specification also known as MSA Advanced environment that targets next-generation smart-phones has not published any drafts yet.

The importance of the MSA specification should not be underestimated. It can be considered to be one of the most efficient efforts that the industry has made to address fragmentation and various specification issues by gathering various JSRs under one umbrella.

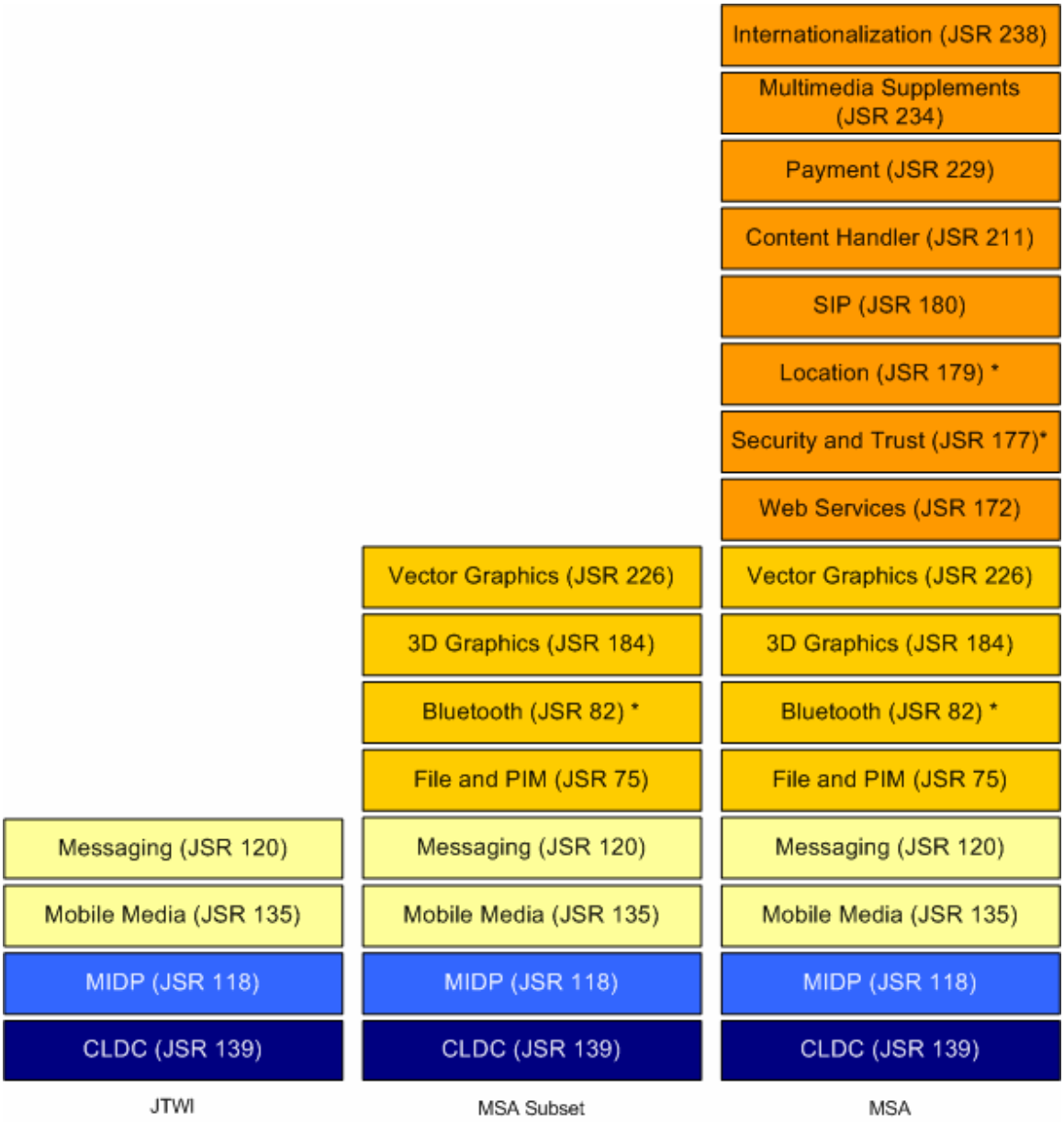


Figure 5 – Comparison of JTWI, MSA Subset and MSA architecture. * notes that a JSR or part of it is only conditionally mandatory.

The first and the main goal of the MSA specification is to reduce the number of optional clauses to minimum, reduce fragmentation and build a feature rich platform that reflects the market needs from 2006 onwards. It contains two sets of mandatory component JSRs, additional clarifications, additional requirements and recommendations for implementers. The second goal of the specification is the ability to use it in a wide variety of different markets and customer segments. To support

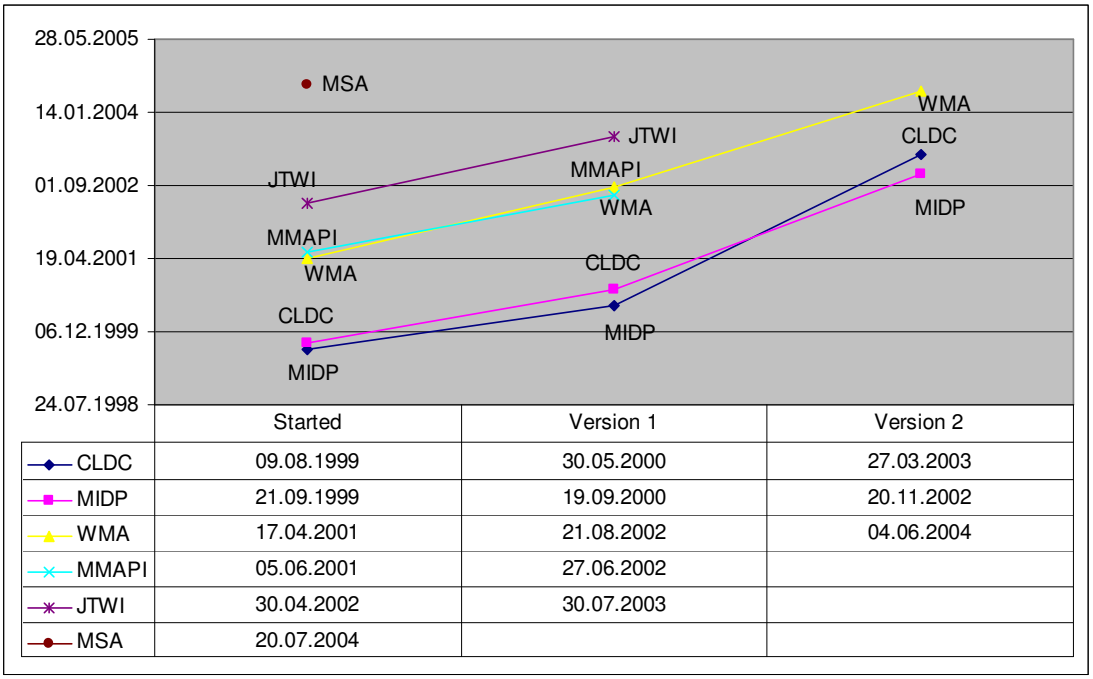
this, the MSA specification introduces two platform definitions – MSA and its subset[6] and supports using CDC as the underlying configuration.

The specification is expected to be reach final stage in August 2006. The first devices implementing the MSA specification are expected to be released in Autumn 2006 or Spring 2007.

3.8. Device generations

When developing mobile applications, it is important to understand the evolvement of the technology from its start to present. The field of mobile phones is rapidly evolving but the application developers are most interested in the handsets that are physically on the market today. The industry is looking into the future and defining the future architecture of the platform but it might take a considerable amount of time until these specifications reach full market potential.

For development efforts, especially the ones that are targeted to mass-market customers, it is very important not to focus on the latest and most innovative handsets, but try to utilize the handsets that are currently widespread.



	Started	Version 1	Version 2
CLDC	09.08.1999	30.05.2000	27.03.2003
MIDP	21.09.1999	19.09.2000	20.11.2002
WMA	17.04.2001	21.08.2002	04.06.2004
MMAPI	05.06.2001	27.06.2002	
JTWI	30.04.2002	30.07.2003	
MSA	20.07.2004		

Figure 6 – Important Java Micro Edition specifications in timeline, based on [4]

[6] The TCK for a specification costs about $50 000 (USD). As MSA is an umbrella specification consisting of 16 JSRs, the cost for implementers may start from $850 000 (USD) for just the TCKs alone. A subset of MSA consisting of 8 JSRs makes it possible to cut the costs by half so that it would more feasible for small companies to implement the specification [4].

A convenient way of analysing the current market situation is to divide the CLDC platform devices into 4 generations [11, 17, 43]:

Generation	Components	Devices available
1	CLDC 1.0, MIDP 1.0	2001
2	CLDC 1.0/1.1, MIDP 2.0	2003
3	JTWI	2004
4	JTWI, MSA	2006/2007

To target the widest audience, developers should address only first and second generation handsets. The first generation handsets included just the very basic functionality to be able to run simple Java applications on mobile phones. Since the specifications have retained backward compatibility, the simple applications can still run on the newest handsets but the latest handsets enable developers to utilize the mobile phones already as feature rich computing platforms.

4. Technologies for mobile commerce

4.1. Introduction

This chapter presents a number of different technologies that enable the delivery of content and applications to mobile devices. Each of the technologies is compared to Java Micro Edition, Mobile Information Device Profile where appropriate.

4.2. Applications for the enterprise and consumers

Typical mobile applications require a sophisticated set of features to be useful for solving mission-critical[7] enterprise problems as well as answering the increasing need from consumers.
The table below summarizes the usual requirements that enterprises and consumers would typically need from a more advanced mobile platform application [37].

Feature	Characteristic
Standalone application	▪ Small footprint versions of application logic and local processing; thick intelligent, client/server or smart client applications ▪ Applications utilizing intermittent wireless connectivity ▪ Intelligent use of network bandwidth
Rich user interface	Ability to build simple but richer (than WAP browser based) user interfaces
Network aware	▪ Can be occasionally connected ▪ Client/server and distributed applications ▪ Direct device-to-device network communications ▪ Support for different web standards (HTTP/XML/..)
Reliable messaging	Guaranteed message delivery that ensures transactions are not lost due to lost of network coverage

[7] A system that is critical to the functioning of an organization and the accomplishment of its mission

Feature	Characteristic
Local persistence	Persisting data at the device locally. It is key to offline functionality and caching, guaranteed message delivery and better wireless bandwidth utilization.
End-to-End security	Well defined and scalable security model
Data synchronization	Synchronize data when needed
Notifications	Real-time asynchronous notifications
Multi language support	Target application to different languages
Cross platform and low cost of development and ownership	Can be developed once and deployed to any mobile device with a compatible Java ME implementation, appropriate profile
Access to corporate servers	Integrate and interoperate with existing corporate servers

Current Java ME platform provides almost all the features required by enterprise and advanced consumer applications. Several missing functionality domains (smart-card based security and various others) have been added by new JCP specifications.

4.3. WAP/WML (i-Mode/CHTML, XHTML)

Wireless Application Protocol (WAP) and also i-Mode's compact HTML (CHTML) [13] and XHTML enable the creation of thin client applications (i.e. micro-browser applications). These platforms are declarative and thus do not compare well to the procedural rich-client nature of the Java ME technology, but as the difference of WML/CHTML technologies and Java ME is often misunderstood, it is important to clarify the main aspects of both technologies.

	WAP	***Java ME MIDP***
Orientation	Thin clients (browser technology), analogue to HTML.	Rich client (non-browser based). Analogue to desktop applications.
Offline/disconnected operation mode	Not available	Available
Availability	High	Medium, High

	WAP	***Java ME MIDP***
User interface	Limited customization	Small number of limitations for customization
Maintenance and setup	Server-centric	Client-centric (optionally also server)
Supports leveraging device extensions	No	Yes
Device support	Widespread (since 2000)	Widespread (since 2003)

4.4. Binary Runtime Environment for Wireless

Qualcomm's Binary Runtime Environment for Wireless (BREW) is a technology which supports rich client development and provisioning. It is important to note that BREW is not just an API, but also a certification and a distribution model. BREW applications are written in C++ and run natively on BREW devices.

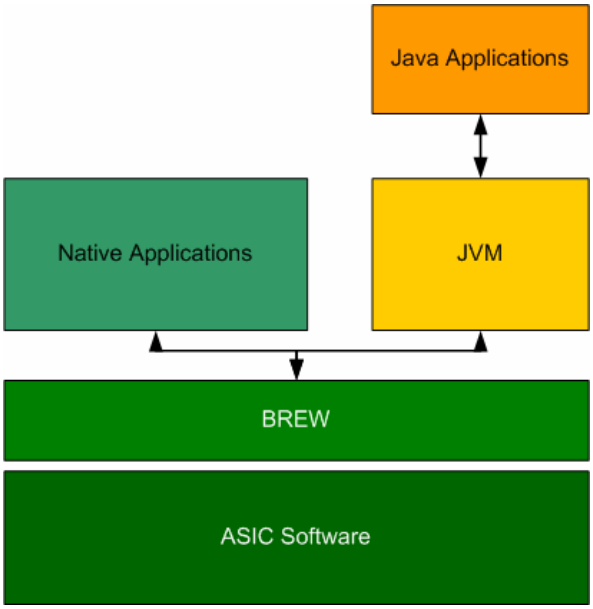


Figure 7 – BREW Architecture

Brew is air interface independent and can support GSM/GPRS, UMTS, cmdaOne & CDMA2000 1X/1xEV-DO [20].

Application development using BREW usually involves the following steps:

1. Application is developed and tested on a BREW emulator

2. Application is then tested on a BREW device. For testing, developers need to use Qualcomm's commercialization tools and test signature generator to be able to load applications on a specific handset for testing.
3. Application needs to get BREW certified, signed by the content provider.
4. Application can be distributed through BREW partner operators.

Brew development has somewhat steep entry costs - developer has to pay for testing and additional costs might occur for various tools such as ARM compiler licence. This tends to exclude many hobbyists from developing software for BREW handsets [21].

In comparison to Java ME, BREW has the following characteristics [3, 21]:

- Steep entry costs mean less market dilution, on the same time prevent market entry for low-budget start-ups or hobbyists.
- BREW applications do not execute in managed environments and are thus more prone to programmer errors.
- BREW API is more standard across handsets, there is less fragmentation than in the Java ME space, but as BREW devices also have different screen sizes, memory options and other characteristics, applications might still require considerable porting effort.
- There are usually no limitations to application size on BREW handsets.
- Brew features location information, multimedia, telephony services and file system access.
- There are Java ME and Flash Lite implementations available for BREW devices.

Unfortunately, because the process which precedes application delivery to end customers is costly and support on handsets is not as widespread as with Java ME, BREW plays only a minor role in today's mobile application development.

4.5. .NET Compact Framework

.NET Compact Framework (.NET CF) is Microsoft's [48] proprietary technology for mobile application development. .NET CF has a strong focus on enterprise applications and is geared towards high-end Windows CE and Pocket PC devices. Applications are usually programmed using C# or Visual Basic .Net (VB.Net).

.NET Compact Framework has been very often described as one of the most serious competitors to Java ME technology. Their architecture is somewhat similar as can be seen in Figure 9.

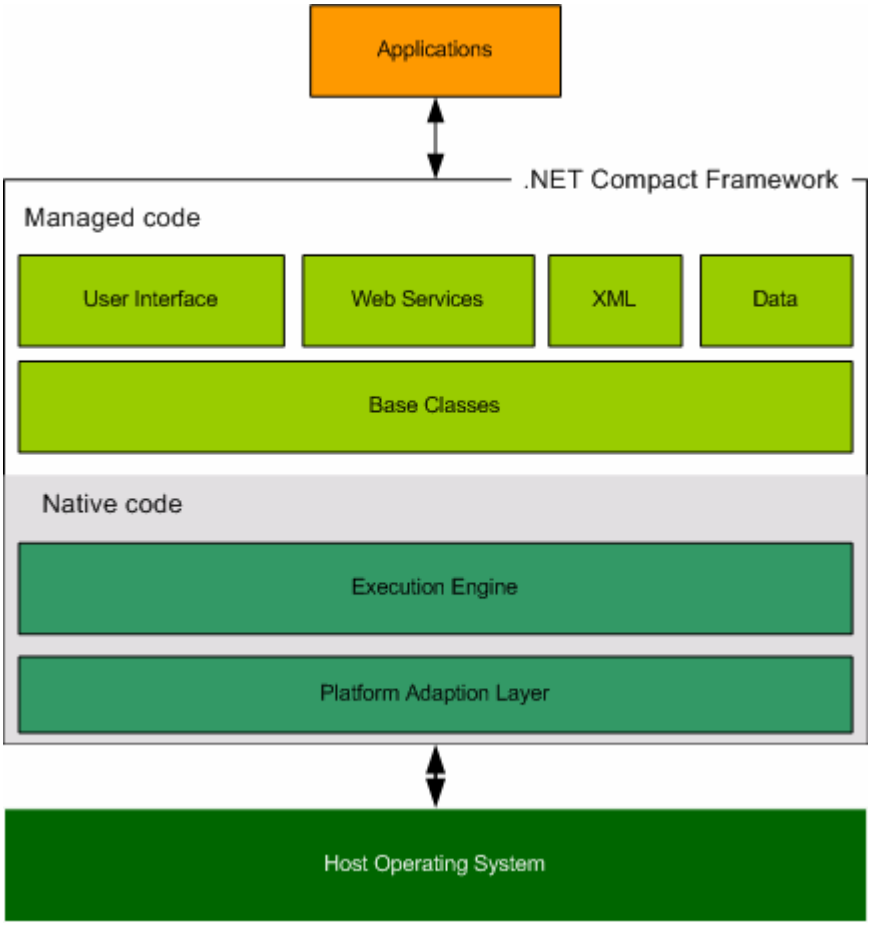


Figure 8 – Architecture of the .NET Compact Framework

The main components of the architecture are [33]:

- Execution engine – low level code that deals with loading, Just-In-Time compiling, executing managed code and managing memory, written in C/C++ and compiled down to native instruction set of the processor (analogue to a JVM).
- Base Classes - common types and functionality that developers can use to implement most data-processing algorithms. The base classes include core types, file I/O, streams, sockets networking, reflection and globalization (analogue to Configurations in Java ME).
- User Interface libraries – User interface libraries enabling developers to take advantage of high level interface controls such as buttons and lists as well as use lower level drawing operations such as drawing 2-dimensional objects (analogue to Profiles in Java ME).
- Web Services libraries – easy development of web services (analogue to Optional Packages in Java ME).

- XML libraries - classes to manipulate XML (Analogue to Optional Packages in Java ME)
- Data libraries - .NET Compact Framework offers an ADO .NET object model to work with relational data. ADO.NET gives developers classes to manage a set of related relational tables in memory along with classes to provide views onto this data. (Analogue to Optional Packages in Java ME)

Further comparison of the two platforms is given in the table below.

	.NET CF	**Java ME**
Programming language	C#, Visual Basic.Net	Java
Runtime environment	managed	managed
Availability of development tools	Extensive	Extensive
Specification process	Microsoft (single company)	Java Community Process
Devices targeted	Microsoft Pocket PC type devices	Wide range of handsets from several manufacturers
Market focus	Enterprise market	Consumer and enterprise market
Byte code compatibility	Standard .NET CLR	Not compatible with Java SE or Java ME CDC
API compatibility	.NET subset	Java SE subset
Native APIs	Available, consistent APIs	Not available
Security model	Simplified .NET	Limited Java SE security model and OTA Provisioning
Client installation	Internet Explorer based installation, ActiveSync	OTAP specification

The following table summarises the different features of both platforms:

	.NET CF	Java ME CLDC
User Interface	Windows Forms	MIDP LCD UI API and vendor specific UI libraries
Database access	ADO.Net	Vendor specific APIs
XML APIs	Build into ADO.NET and other standard APIs	third party libraries
Web Services	Built in to the platform	JSR 172 where available or third party libraries
E-mail and PIM	P/Invoke[8] Outlook APIs	PDA Optional packages (JSR-075) on some devices
SMS	P/Invoke SMS device stack	Wireless Messaging API (JSR 120, JSR 205)
Cryptography	Third party libraries	Third party libraries
Instant Messaging	P/Invoke MSN and other APIs	Third party libraries as well as JSR being defined.
Multimedia	P/Invoke Windows Media Player APIs	MMAPI as well as MIDP standard APIs
Background running applications	Yes	No. Exceptions are vendor extensions (VSCL), expected in MIDP 3.0

As a conclusion, .NET CF offers a very competing environment to Java ME for mobile application development. On the other hand, the main focus of .NET CF is the enterprise market, while Java ME is more oriented towards consumer applications. Both platforms can be used to deliver mobile applications, but require very different skill-sets.

[8] .Net CF programs invoke methods in specifically formatted Win32 native libraries (P/Invoke).

4.6. Messaging

The SMS and related technologies (EMS, MMS) can be considered one of the most important mobile technologies for value-added consumer applications at present, spanning most user groups and handsets. Mobile messaging brings the benefits of reliability (gateway servers store and forward messages), asynchronous nature and flexibility in the end-to-end architecture (loosely coupled components). Compared to Java ME they are complementary technologies and use each other in an interesting symbiosis.

- SMS and WAP SI push messages can be used to deliver Java ME application download alerts to the mobile handset.
- MMS technology can be used to deliver MIDlets to devices that support it.
- There are optional packages for Java ME which enable developers to make use of different messaging technologies including SMS, EMS and MMS.

4.7. Symbian OS

Symbian OS is an open standard mobile operating system developed by a group of leading mobile handset manufacturers, each owning a stake in Symbian. The manufacturers are: Nokia, Ericsson, Sony-Ericsson, Panasonic, Siemens, Samsung [22]. Analysts predict that Symbian OS will account for 76 percent of all smart-phone shipments by 2008 [29].

Symbian is an open OS standard which enables third party developers to write and install applications independently from the device manufacturer.

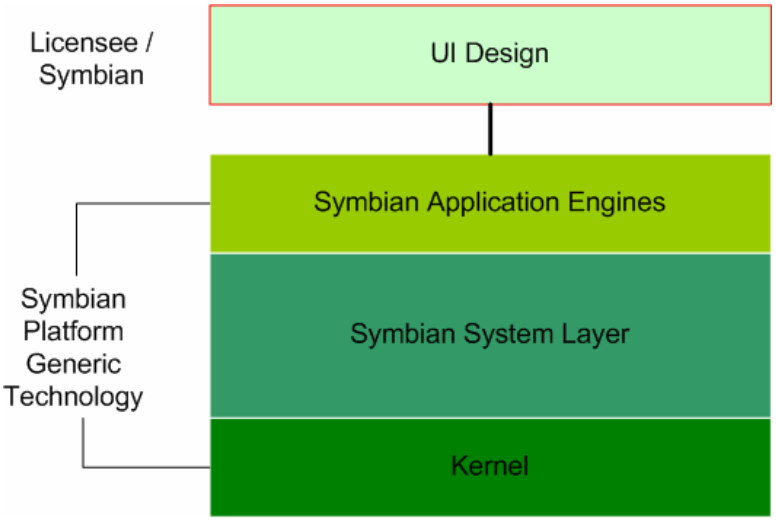


Figure 9 – Symbian Architecture

Symbian architecture can be broken down as follows:

- Kernel layer
- System Layer provides system APIs to handle various OS tasks.
- Application Engines provide access to functionality such as a calendar and contacts, but just the data not its presentation.
- User Interface layer deals with the presentation and can be very specific for a particular device. It is the responsibility of Symbian or the licensee to provide this layer.

Symbian OS v 7.0 introduced support for Java MIDP 1.0 alongside the existing support for other Java based technologies such as PersonalJava and JavaPhone. Since version 7.0s the Java platform on Symbian is completely based on Java ME (CLDC and MIDP) [23].

The native language to develop applications to Symbian devices is C++. The main comparison between native Symbian development and Java development either on a non-Symbian device or a Symbian device is given below. It is a common misconception that native Symbian development in C++ does not have limitations. There are a number of limitation in Symbian C++ standard such as: limited template support, no C++ exceptions and fixed stack size.

	Symbian OS environment	**Java ME MIDP**
Development language	C++ and others (Mobile Python, Perl, OPL)	Java
Direct access to native OS functionality (e.g. memory management)	Yes	No
Access to calendar, contacts etc.	Yes	No (standard access without extensions)
Possibility to extend system APIs	Yes	No (there is no JNI support)
Performance considerations	Runs natively in the device	Runs on top of a Java Virtual Machine
Portability	Low	High

Symbian is a very widely deployed operating system and has gained considerable momentum in the mobile OS market. Symbian applications are often developed

instead of Java ME applications when native and low level functionality is required. On the other hand, Java ME applications are much more portable than Symbian applications and can address a vastly greater customer base.

4.8. Palm OS

Most of Palm OS applications are written using C but development environments are also available for developing in a variety of other languages including C++, VisualBasic or Java [34].

Many PDA type devices like Palm do not usually have the JVM preinstalled by the factory. There are two options to get a Java runtime:

- Require the users to download and install the Java runtime.
- Distribute the runtime with applications (special licensing conditions).

Currently, Palms officially support Java ME and MIDP and the support has been integrated into developer tools. Java ME CDC based profiles are not officially supported because of the limitations of the Palm OS. MIDP environment is seriously more limited than native Palm OS and as a result not used very commonly for Palm device oriented application development.

4.9. Mobile Flash (Flash Lite)

Flash Lite is a lightweight version of the Macromedia Flash Player, a multimedia and action player supporting rich internet applications and streaming video and audio, that has been optimized for use on mobile devices. According to Adobe Systems, company who owns the Flash trademark, the Flash technology has been shipped pre-installed on more than 45 million consumer devices ranging from mobile phones to toys [42]. Flash adoption has been especially fast in Asia, particularly in Japan.

The development steps of a Flash Lite applications usually involve the following:

1. Purchasing and installing Macromedia Flash Professional.
2. Acquiring a supported handset.
3. Registering with the Macromedia Developer Program.
4. Downloading and installing Flash Lite Content Developer Kit.
5. Development and testing of the applications.
6. Publishing the content applications for delivery to customer's handsets.

Flash player is not installed on some phones, therefore distributors might need to take care of helping customers to purchase and download the Flash Lite player from Adobe online store before they can use the applications.

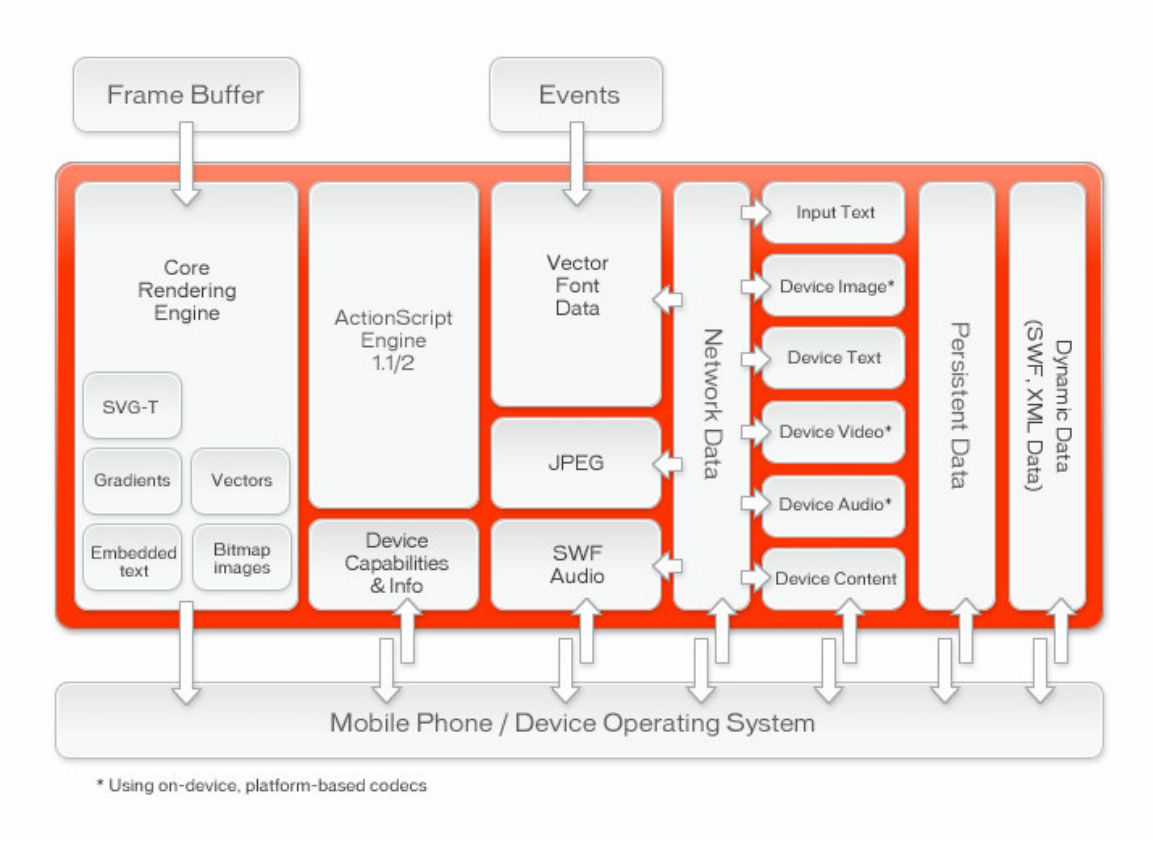


Figure 10 – The architecture of Flash Lite [42]

Flash technology and Java ME are compared in the following table:

	Flash	**Java ME CLDC**
Device support	Niche	Covers most mobile phones
Ability to update the runtime in the device	Usually not possible[9]	Usually not possible[10]
Fragmentation problems	Yes	Yes

[9] On many phones, the Flash Lite environment has been preinstalled, some phones allow the user to purchase and install the Flash player OTA.

[10] It is very uncommon that Java environments are installable by the user into a mobile phone. Although cases when upgrading firmware also upgrades the JVM are possible.

	Flash	Java ME CLDC
Language orientation	More interface oriented (applications designed like movies by designers), limited access to device capabilities, rich multimedia support, SVG and vector-graphics rendering	More application development oriented (access to devices, data persistence)
Connectivity	HTTP based	HTTP, messaging and other protocols

4.10. Embedded Linux

Currently only 17 percent of embedded system designers are using embedded Linux. About 66 percent of the designers say that they are either not interested in using the embedded Linux platform nor will be using it anytime soon [30]. Although not widespread, Java ME compatible JVMs exist which enable application development for the mobile Linux platform.

5. Fragmentation

5.1. Introduction

The number of devices surrounding us is rapidly growing. Every day new gadgets such as music players, photo cameras, camcorders and handsets enter the market each trying to over-perform other similar products. It is the nature of the field of consumer devices to be fragmented and it is also an important characteristic of the mobile phone space. Albeit being so evident, fragmentation and its significance for mobile development projects might be often underestimated.

Fragmentation is a very broad term and depending on a certain viewpoint taken, can itself be defined in several ways, one of which suitable for this thesis can be [15]:

Any activity in the entire business landscape that decreases possibilities of reaching critical volumes of adequately compatible Java-enabled devices to allow profitable business for all parties in the value chain.

Using this definition, a look is taken at the fragmentation from a wide viewpoint taking into account the business as well as technical problems that this effect brings. Fragmentation is divided into four separate domains: device-level, standardisation-level, API-level and implementation-level fragmentation, introducing a framework of analysing fragmentation issues and risks involved.

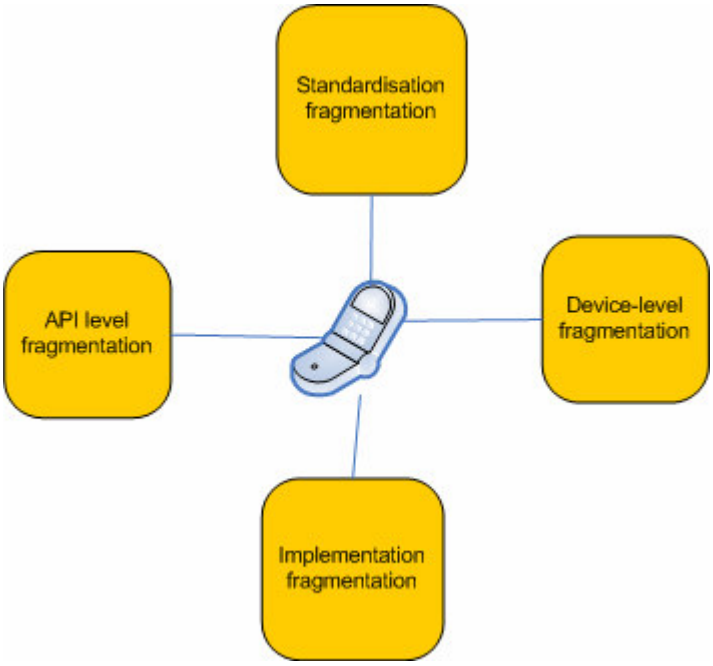


Figure 11 – Fragmentation domains

For each fragmentation domain we will look into the definition and the cause of the problem, some real-life examples and possible ways of addressing the issues.
Fragmentation has an effect especially on mass-market oriented software projects and is sometimes referred to as the biggest difficulty that the mobile industry is facing. On the good side, fragmentation is a normal by-product of innovation. As technologies improve, the diversity of new innovative products and therefore fragmentation caused is imminent.
For mass-market software applications there are no simple ways for a software developer to reduce the effect of fragmentation except limiting the addressable device set to just one device. Since the majority of mass-market projects usually aim to target as many devices as possible, it is crucial to understand the different aspects of fragmentation and plan the risks involved into the development process as early as possible.
When dealing with enterprise applications which tend to be addressed commonly to a single device type (model), fragmentation does not have a considerable effect.

5.2. Device-level fragmentation

5.2.1. Definition

Device-level fragmentation means that there is a huge number of devices with different capability sets. The diversity makes it difficult to write portable software that would work on all of them. Device-level fragmentation is the most abstract of the fragmentation issues and not specific only to the mobile device field nor software platform technology. At the same time it is also the most perceivable of all fragmentation types.

5.2.2. Cause

Reasons behind device-level fragmentation vary, but are mostly business-related. The diversity of handsets in a product-line fills a business need of differentiation and helps to increase sales. The demands of consumers are different – some people want to use photo cameras, others music players while others are mostly interested in gaming. As consumers do not want to pay for features they do not use, the only way to satisfy all user groups is to produce different devices [14].
Another angle of device-level fragmentation stems from innovation and quick advancement of the field generally.

Analysis of the factors of device-level fragmentation can be divided into two sections: user interface related and device hardware related.
User interface related characteristics with descriptions are given below:

Characteristic	Description
Screen size	Screen sizes can vary from small screens of low-end handsets to high-end smart phones with large screen dimensions
Colour depth	Ranges from 1 bit (monochrome) to 18 (and more) bits
User interaction	While most of the mobile phones have standard keypads, some of them also allow to use a pointer device in addition to a keypad and others might have completely different user interaction mechanisms such as jog wheels.
Sound support	Phones might have dedicated sound hardware or have implemented software algorithms to play different kind of sounds making the sound capabilities varied.
Language support	The character sets supported by small devices may vary between handsets targeted to the Asian and European markets for example.
Operating system	Phones run a variety of operating systems from different manufacturers (Microsoft, Psion, Symbian, RIM and others).

Device hardware characteristics:

Characteristic	Description
Central Processing Unit (CPU)	The CPU determines the overall performance of the device and can range from 20MHz in pagers to several hundred MHz in PDA type devices.
Memory capabilities	The volatile and non-volatile memory budgets that devices can access are varied. Memory size is especially important as mobile phones do not usually have hard drives and thus rely on memory for data storage.

Characteristic	Description
Batteries and power	While all manufacturers try to increase the battery life, battery power is still a scarce resource and affects especially applications with heavy data processing needs.
Connection ports	Device connection ports can vary from Bluetooth, infrared, FireWire to USB and others
Peripherals (also integrated)	Peripherals that modern phones have can vary: cameras, barcode scanners, location scanners, printers, biometric scanners.
Network options	Devices can utilise different network standards: GSM, CDMA or WCDMA

As mobile phones have been traditionally centred around voice functionality, it is also very important to consider which carrier's network a device is working in. While this is not directly a device related characteristic, it is important to understand that carriers might have altered versions of some devices (additional applications) and also the network characteristics and service characteristics in a certain carriers network might vary.

5.2.3. Effect

As an effect of device fragmentation, the amount of information that developers and other project members need in order to make adequate decisions for device support is greatly increased. The lack of concrete information about technical device parameters is one major obstacle in forming correct strategies for device support. In addition to pure device-related characteristics the environment (network and service characteristics) in which the phones operate can also vary. From development point of view, device fragmentation is one of the main factors behind the porting effort required for application releases and the increased complexity of the framework that handles the delivery of different ports to devices. Proper development process also requires handset testing and developers need to acquire many different devices for in-house testing or use external testing partners. This increases the project costs substantially and increases the risk of exceeding the planned timelines.

5.2.4. Solutions

Device-level fragmentation together with the difficulty to find or the lack of information is a dangerous set of problems and the cause why mobile development efforts are sometimes considered to be more failure than success. There are no simple solutions to address device-level fragmentation's effect on mobile projects. Fragmentation increases with every device added to the supported devices list. Even limiting the application to only one phone manufacturer is not enough as the set of devices can be still overwhelming[11] and it is not usually an option with mass-market consumer applications. The crucial aspect of properly addressing the device fragmentation is to take into account the essence of this type of fragmentation and plan the project timelines, resources and costs accordingly.

5.3. Standardisation fragmentation

5.3.1. Definition

Standardisation fragmentation means that although official standards exist, there is either a large number of them or their evolvement process has not been clearly guided producing dependencies between increments and confusion among implementers.

5.3.2. Cause

Standardisation fragmentation has its roots in a number of sources [15].

- Increments in separate JSRs are loosely coordinated.
- Overall architectural frame is not very well managed.
- Specifications are loose allowing too much freedom for implementers.
- Lack of component package architectures (this is addressed by some new standards such as JTWI and MSA).

JCP has defined several specifications addressing different capabilities of devices. These standard specifications cover different capabilities of devices such as the

[11] Even choosing a common phone type – Nokia Series 40 platform, official Forum Nokia FAQ states the following: Applications may require modifications to run usefully on Series 40 devices that have screen resolutions that differ from the device resolution of the original target. In addition, if an application makes use of any of the technology extensions provided in a particular Series 40 Platform edition or device, its compatibility will be limited to devices of the same edition or with the same technology extensions. Specific Series 40 devices may have unique capabilities, features, or issues. The best way to maximize development investment is to design initially for the Series 40 Platform, and then optimize the design for specific devices [43].

ability to send short messages (SMS), play and record video or audio media (MMAPI) or access web services. Different versions of such standards and their evolvement is somewhat not coordinated. Because of licensing issues, many of the existing APIs cannot be reused when creating new APIs.
Examples of some of the specifications are given in the following table:

Standard	Description
MIDP and CLDC	MIDP 1.0 was designed to work on top of the CLDC 1.0 specification. A MIDP 2.0 usually works on top of CLDC 1.1, but the two are not related – as a result there are devices running on MIDP 2.0 and CLDC 1.0. So although being a MIDP 2.0 device, it still has CLDC 1.0 limitations such as lack of floating point arithmetic for example.
MMAPI	Multimedia API adds multimedia capabilities to a Java device, but does not for example require any specific video format. Developers would need to convert video formats on the fly for each specific device.
MIDP and CLDC specifications generally, optional packages	There are a lots of "optional" clauses in the specifications which result in some implementations adopting the "bare minimum", the others going for maximum defined in the specifications.

5.3.3. Effect

As an effect of standardisation fragmentation, it is not always clear if a certain Java-enabled device has adequate functionality available to run an application or not. Whilst a device might be WMA or MMAPI-enabled, it can be a version of the WMA standard which does not support a certain feature set required. Even MIDP and CLDC specifications on a higher level have many optional causes, some examples of which

are devices supporting different video or audio formats or failing to load images that exceed a certain size. Standardisation fragmentation considerably steepens the learning curve of understanding the different specifications and getting a picture of the functionality that could be planned into a product release.

5.3.4. Solutions

Some of the solutions to reduce standardisation fragmentation in the wireless Java space from the industry viewpoint can be as follows:

- Improve coordination between specification efforts and architecture planning.
- Improve the standardization process (JCP) to allow more easily building on top of other specifications.
- Increase the completeness of the specifications and to reduce optional clauses
- Create component APIs (like JTWI, MSA) which define category/market specific clarifications to the standards.

From developer's viewpoint it is crucial to understand that standardisation fragmentation steepens the learning curve and makes it harder to make strategic decisions.

5.4. API-level fragmentation

5.4.1. Definition

API-level fragmentation means that there is no minimum required API set available on mobile phones and the API set supported is very varied. Some devices also carry proprietary APIs alongside the ones standardised by the JCP. The timing of API implementation releases is too different between device vendors.

5.4.2. Cause

As there were no API component packages such as JTWI or MSA defined for several years, most current Java-enabled devices implement different APIs. Some phones might have messaging capability, others do not have any way for Java applications to send messages. As the first version of the MIDP specification did not define enough APIs to be productive in certain vertical markets, like game development, several manufacturers offer their own API extensions which are complementary to defined standard APIs (defined through the JCP). Examples of such APIs are Nokia's

UI API that expands graphics functionality for Nokia handsets, Messaging API providing messaging and Nokia SOA API for web services support[12].

Often the timing of supported APIs between the handset manufacturers is too different causing additional fragmentation.

Some examples of this type of fragmentation are introduced in the table below:

Manufacturer, API	Description	Handsets affected
Nokia, User Interface API (Nokia UI API)	Nokia UI API is a Nokia proprietary API to extend the standard MIDP 1.0 profile with several extensions such as ability to draw full-screen, play simple sounds, support for transparency in graphics (makes it possible to draw non-rectangular shapes) and low-level image access.	Available in Nokia devices supporting MIDP 1.0, deprecated in Nokia MIDP 2.0 phones. Interestingly, this API is also available on some Sony-Ericsson, Samsung and Sendo phones [16, 17].
Nokia, SMS API	SMS messages can be sent and received.	Nokia 3410
Siemens, Led API	Allows manipulation of LED lights on the handset	Siemens M55, Siemens CF62
Siemens, Game API	API to simplify games programming for Siemens' devices	Various handsets from Siemens.
Motorola, Lightweight Windowing Toolkit (LWT)	API to expand the graphics capabilities of J2ME	Various handsets from Motorola

5.4.3. Effect

The cause of this kind of fragmentation affects portability and results in applications being limited to a single device, manufacturer or a carrier or requiring porting effort to support different handsets.

[12] Nokia SOA API has been deprecated in favour of JCP specification JSR-172 a standardised web services access API for the Java ME platform.

5.4.4. Solutions

As with other types, there are no easy ways to address this kind of fragmentation. One way and approach to avoid being "locked" to one manufacturer or carrier is to keep applications always written to the "lowest common denominator" technology.

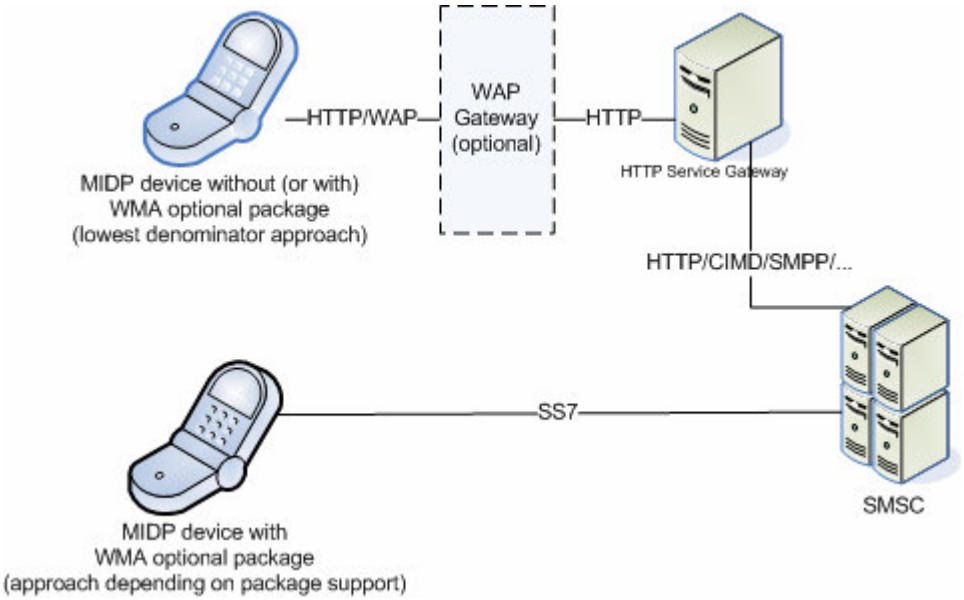


Figure 12 – Example of utilizing MIDP 1.0 to send SMS

A lowest common denominator approach might utilize the server-side backend and the only required connectivity protocol in the MIDP specification (HTTP) to add functionality that might be in some cases missing from a device. Other examples might be to use only low-level user interface elements to provide consistent user experience. Lowest common denominator approaches are not always possible or efficient (it might be much more costly to use low-level than high-level user interface components), especially areas such as multimedia rendering or performing device-specific functions.

As a future solution from the industry perspective, content developers, manufacturers and operators need to work closely together to establish feature-rich standard core APIs and lay out the future roadmap for the platform. JTWI and MSA specifications are good examples of such joint efforts.

5.5. Implementation fragmentation

5.5.1. Definition

Implementation fragmentation means that there are a number of different devices that implement the same standard APIs but behave differently due to implementation differences [14, 15]. Another aspect that can be categorized under the implementation fragmentation is the download mechanics, which determine how the applications are downloaded and installed on devices.

5.5.2. Cause

Java is an open standard available for licensing from Sun Microsystems. With Java ME, the JVM implementations as specified in the MIDP specification are produced by handset manufacturers (licensees). As a result, several companies produce their own JVMs which are built to conform to MIDP 1.0 or MIDP 2.0 standards. As standards have loose definitions (and optional clauses) and as devices are different, a lot of room is left for interpretation and the behaviour of the resulting environments differ.

5.5.3. Effect

Poor JVM implementations can result in applications behaving unexpectedly or not working at all. The implementation differences might even vary across different product lines of one handset manufacturer. Typically JVM fixes that cause new production firmware releases ultimately drive the creation of new devices, each with its own SKU.

Different download mechanics mean that even devices that all have proper standards implemented and tested might have different download mechanics resulting in applications failing to be delivered to handsets.

5.5.4. Solution

As a solution for implementation fragmentation, the whole industry should move towards defining more precise specifications and pay more attention to quality. Overall testing should be improved and attention paid to the quality of the TCKs.

6. Practical considerations of Java ME

6.1. Development and planning phase

6.1.1. Documentation

It is very common that development efforts may need to allocate significant amounts of time, especially in the planning phase, to gather documentation for making technical or business-level decisions. Various official JCP-produced documents are easily accessible, but it should be noted that:

- Documentation about handsets is hard to find and can require considerable amounts of time for research. The information that can be found usually lacks the technical depth which developers need (profile support, firmware, problems)[13].
- Documentation about different problems of devices is especially hard to find since there are no official sources that collect and publish this kind of information.
- Documentation about specific operator extensions might require signing contracts with operators and might not be freely available for developers. An example of a library which is accessible only for companies who have signed a partner agreement is Vodafone Service Class Library (VSCL) [55].

Some projects which do provide a selection of information about devices and in some cases also possible issues are J2ME Polish [16], Phone Scoop [52], Wireless Universal Resource File [53], Nokia Developer Forums and the known issue documents of devices [43], Sun Developer Network Device Database [17] and Mobile Research [56].

6.1.2. Development approach

It is a common misconception that wireless development is simpler than developing other kinds of software (desktop or server solutions). While the difficulty of programming is hard to measure objectively, mobile development projects often need knowledge that spans the whole enterprise – from mobile client side to backend security issues.

[13] Conclusions made by searching through several phone manufacturers sites, such as Nokia (http://www.nokia.com), Sony-Ericsson (http://developer.sonyericsson.com), Motorola (http://developer.motorola.com/) and others.

It is also important to consider the resource constrained nature of the devices, therefore implementing systems using a proper object oriented design is often unnecessary and in many cases not recommended. Due to various size limitations of handsets, much of mobile developer's attention should be paid to optimization of the code so that it would run faster and also be as small as possible.

- Instead of using encapsulation and other object-oriented design principles, mobile developers usually aim to limit the number of classes to make the resulting application more compact.
- Objects use heap memory and destroying them requires additional CPU cycles. Garbage collector takes a while to run and could slow down application drastically. The recommended approach for mobile applications is to try use less objects and also make use of object pools[14] when possible.

6.1.3. Platform limitations

The language differences of the Connected Limited Device Configuration are the following:

- There is no floating point support in the 1.0 version of the specification. The reason for this limitation is the absence of floating point hardware and the complexity and resource consumption of emulating it software-wise. Floating point support is required in the CLDC 1.1 specification.
- There is finalization for objects (no callback for object finalization from the garbage collector).
- No support for the `java.lang.Error` exception class hierarchy.

The class files bundled with the CLDC specification are divided into two: CLDC-oriented classes and classes that form a subset of Java SE. The class files that are used in a CLDC environment from the Java SE are added using the following rules:

- The package name must be identical to Java SE counterparts.
- There cannot be any additional public or protected methods or fields.
- There cannot be changes to the semantics of the classes and methods.

As a result, the Java SE classes used in CLDC environment can only have methods or fields removed and there cannot be changes to existing methods.

The JVM differences in the CLDC are the following:

[14] Object pool enables to save the runtime cost of creating and destroying and garbage collecting object by storing frequently used objects in memory all the time.

- No support for weak references in the 1.0 version of the specification. Weak references let the programmer to keep a reference to an object which can still be garbage collected. Version 1.1 of CLDC adds weak references support.
- No support for reflection.
- No support for thread groups and daemon threads.
- No support for accessing native functionality (Java Native Interface).
- No support for user-defined class loaders.

6.1.4. Security model

The security model defined by CLDC is divided into two: virtual machine security and application security.

The role of the virtual machine security is to protect the underlying device from any damage that executable code could cause. This level of security is added by validating the class-file byte-code to ensure its correct execution. The standard Java SE byte-code verification process requires more than 50 kB of code and 100 kB of heap which is not feasible on small resource-constrained devices. The CLDC specification offers an alternative model which only requires 100 bytes of run-time memory and about 10 kB of binary code. The reduction in resources comes from the removal of the iterative dataflow algorithm from the in-memory verification and means that the process is divided into two steps: on the development machine step and on-device step.

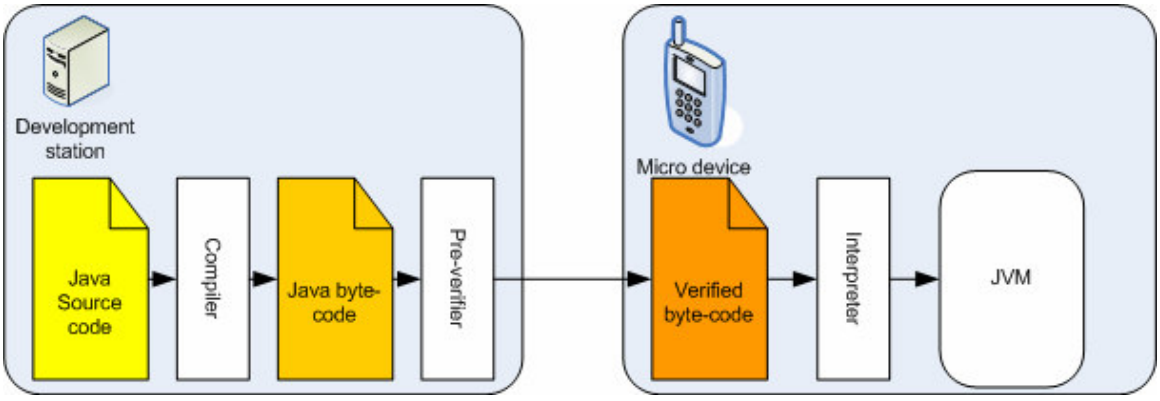


Figure 13 - Virtual machine security in the CLDC environment

On the development machine, the source code is compiled and then pre-verified. Pre-verification essentially prepares the code to be executed by KVM by inserting additional attributes (stack map entries) into the class file. It should be noted that even after going through the verification process, the resulting class file is still a valid

Java byte-code. The stack map entries enable the on-device JVM to check the class file efficiently and reject it if necessary.

Application-level security makes sure that all programs run in a pre-defined "sandbox" – small, controlled environment.

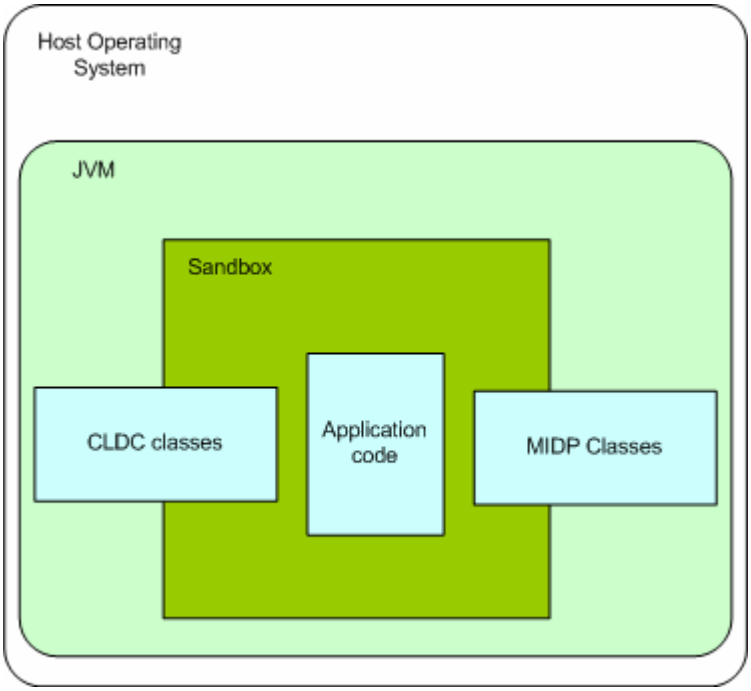


Figure 14 - CLDC Sandbox

Due to sandbox restrictions, developers are not allowed to change the base API set on the device.

6.1.5. Emulation and on-device debugging

Emulator is an off-device software that enables developers to test their applications without owning a physical mobile device. While being very important for the rapid development process, usage of emulators has its side-effects.

- Many emulators are not built on top of the phone's firmware environment and the behaviour of the application might be different on a real handset. Some emulators only use a PC-implementation of core MIDP classes and expose different devices using different skins.
- Emulators often run on powerful PC platforms and are often not able to emulate the resource-constrained phone environment properly. An application that has no performance issues in an emulator might have performance problems on the real device.

- Off-device emulation environments might give developers a false impression of high-speed network connectivity and other connection characteristics that are not commonly present in most current wireless networks.
- Emulators, like other software tools, usually come with their own limitations and full functionality might not be available. As an example, the emulation of certain optional packages might not be available. There are also special cases which are difficult to emulate – messaging or Bluetooth emulation might be just useful for code flow verification as the emulation is different from real messaging implementations on target handsets.
- Emulators, like other software tools, are threatened by problems of poor design issues and code errors that can cause differences with real-device environments.

Although emulation proves a valuable tool for application development, the importance of testing on real devices should not be underestimated.

On-device debugging is not available for most of the phones and even if available, it usually requires specific IDE support.

6.1.6. Porting effort

Due to no real workarounds for several fragmentation issues, development efforts that have to address a large number of devices often need to plan for considerable porting effort. Device specific features such as screen backlight, vibration, sounds etc. can only be developed using proprietary APIs and the application needs to be built for a certain platform. It is also common that problems with different firmware versions might need separate workarounds.

Device and sometimes even firmware-specific ports can be referred to as *platforms* that an application is able to run on. Usually a *platform* consists of a set of building instructions, resource files and source code. In addition to mobile phone environment related changes, applications might also need to be localized for different markets. Considering that an application is addressing 12 platforms and 9 local markets, it would result in: **12 platforms * 9 localizations = 108 ports**.

The number of ports might be even larger when different distributors need to use proprietary logic for operations like billing or event tracking. A huge number of ports can be extremely hard to manage. Adapting solutions which are usually based on pre-processors like J2MEPolish [16] can be used to help on the user interface design level. In many cases developers have their own porting frameworks to manage the

complexities of the process. It is important to understand and plan the porting strategy early into the development process.

6.1.7. Packaging

Before applications can be delivered to handsets they must be packed into a special format. MIDP specification defines an application package called a MIDlet Suite which is in the form of a JAR file. A MIDlet Suite consists of the following:

- A manifest file describing the contents.
- Java classes for one or more MIDlets and classes shared by the MIDlets.
- Resource files used by the MIDlets.

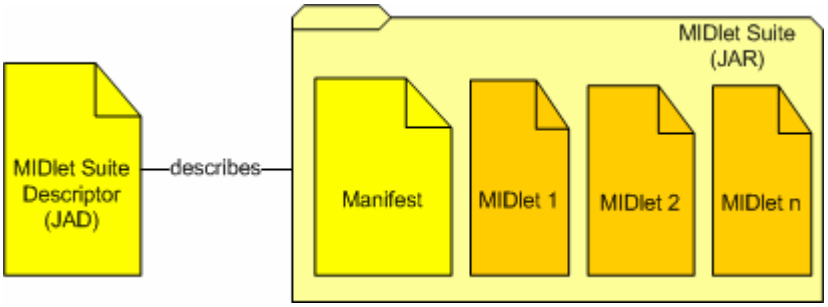


Figure 15 – MIDlet Suite packaging

The MIDP specification does not define how many MIDlets an implementation must be able to support in a single suite. In most cases developers should try to have only one MIDlet per suite for compatibility reasons. Future MSA devices are required to support between one and five MIDlets in one suite.

The MIDP specification also defines an accompanying descriptor file *Java Application Descriptor* (JAD) which can be optionally provided with the MIDlet. It is recommended in all cases to provide the file as it helps the mobile phone to verify if the MIDlet is suitable for the device. During installation, the device is then able to abort the download when, for example, the size of the MIDlet is inappropriate. This would result in a better user experience as the whole application does not need to be downloaded. JAD and JAR files can contain also developer defined parameters such as the server address, service parameters and others.

6.1.8. Obfuscation

The main role of obfuscators is to go through the Java classes in a MIDlet Suite and replace all class, method and variable names with short (and illogical) symbols. As the relationship between the symbols (or renames) stays the same, the processed code is not different for the execution environment. In addition to reducing

application footprint by replacing different names with short symbols and making MIDlet packages hard to reverse-engineer, some obfuscators such as *Proguard* [57] also perform other services – remove classes, methods, fields and attributes that are not used (also known as file shrinking), merge interfaces and classes and apply other optimizations to the byte-code.

Obfuscation has two main benefits.

1. Obfuscation reduces the size of the application and makes a MIDlet smaller. Smaller MIDlets are easier and more fail-safe to download and install. Obfuscation might sometimes be the only way to make a MIDlet work on a device that has a strict limit on the size of the MIDlet Suite. Typically obfuscators can reduce the size of a MIDlet Suite by 10 to 50 % .
2. The byte-code format of the Java classes is standardized and decompilers can be used to translate the byte-code classes back into source code. Obfuscation helps to secure the code from being reverse-engineered as the decompiled code would be very hard to understand. Reverse-engineering obfuscated code is usually too time-consuming to have any real value.

It should be noted that not all obfuscators work correctly in all cases and might produce byte-code which does not work on a real handset, therefore it is important to test the code on the real handset also after obfuscation.

Another aspect of the effect of obfuscation is that obfuscated code is harder to debug because the stack traces are not very informative – class and method names have been replaced by short and meaningless strings and source file names as well as line numbers are missing.

6.1.9. Signing MIDlets

The MIDP 1.0 version treated security issues using a well-known "sand-box" model. Version 2.0 of the MIDP specification introduces the concept of trusted and untrusted MIDlets. Trusted MIDlets are allowed access to sensitive and restricted APIs. As an example, HTTP connection is a sensitive operation because it may involve monetary costs for the user. Trusted MIDlets can acquire permissions automatically while untrusted MIDlets (MIDlet Suites for which the origin and integrity cannot be verified) have limited access to sensitive APIs and require explicit user approval depending on the security policy of the device. All MIDP 1.0 MIDlets are by default untrusted.

In association with trusted MIDlets, the MIDP 2.0 specification introduces a concept of *Protection Domains*. A *Protection Domain* is a set of permissions and interaction modes. These permissions can either be automatically granted or deferred until user approval.

User can grant permissions in three possible ways:

- Blanket – the MIDlet Suite acquires the permission for as long as the suite is installed.
- Session – requests the user to authorize the MIDlet first time the API is invoked and validity is guaranteed until any of the MIDlets in the same suite are running.
- Oneshot – requests user authorization every time a MIDlet is invoked.

Permissions are requested by MIDlets declaratively in the application descriptor or manifest file.

Each protection domain (except *untrusted* domain) is associated with a set of root certificates. A root certificate is associated with only one domain, but domains can be associated with several root certificates.

As recommended by the specification, the following domains should be present on compliant GSM/UMTS devices:

- Manufacturer – uses root certificates belonging to the device manufacturer.
- Operator – used by operator's MIDlets and may use root certificates available on storages such as SIM cards.
- Trusted third party – consists of well-known Certificate Authority's root certificates.
- Untrusted domain – does not have a certificate and is used for unsigned or MIDP 1.0 MIDlets.

The amount of domains and their associated permissions may vary among devices and networks, application developers or distributors should be aware of them when deciding which certificate to use and which permissions to request.

The MIDP and JTWI specifications have defined the following usage groups:

- Net Access – permissions related to network data connections.
- Messaging – permissions related to sending and receiving messages.
- Auto invocation- permissions related to automatically starting a MIDlet, for example by Push Registration.
- Local connectivity – permissions related to connectivity via local ports like Bluetooth or infrared.

- Multimedia recording – permissions related to image, audio, video recording.
- Read user data – permissions related to data like calendar entries or phone book.
- Write user data – permissions related to writing user data.

The availability of these function groups depends on the capabilities of a device.

The behaviour of a web browser when encountering a signed Java applet with an unknown certificate depends on the browser and local settings. Users are usually able to accept to trust an applet even with an unknown certificate. When a mobile device encounters a signed MIDlet with an unknown certificate, the MIDlet will not be installed at all. This makes it especially important to use a public certificate which can be validated to one of the root certificates on the device when signing MIDlets. For example Nokia mobile phones have root certificates from *Thawte* [60] and *Verisign* [61]. Developers would need to obtain a code-signing certificate from either of these authorities to be able to create trusted MIDlets for Nokia devices. In some special cases when partnering with either the operator or manufacturer, developers can also sign their MIDlets using an operator's or manufacturer's certificates.

The signing operation assures the integrity of a MIDlet application JAR file and protects the users from maliciously or accidentally modified binaries. When the JAR file should change during transmission or installation, the verification check would fail and the MIDlet Suite would not be installed.

A potential problem might be caused by the mismatch of permissions requested by a MIDlet Suite and a protection domain. For example, if a required permission is misspelled in the descriptor, the certificate can be validated on the device, but as the requested permissions cannot be fulfilled, the installation will be aborted. However, if the permissions are declared as optional, the MIDlet would still install even if a domain does not contain the permissions or they are misspelled.

6.1.10. Testing effort and certification

Publishing and sales are usually preceded by testing and/or certification. Testing first takes place on emulator platforms and later on real devices. Due to implementation differences, development teams should aim to conduct real-device testing on every target handset. To reduce the number of devices on which to test, the handsets can

sometimes be grouped into device families[15]. Sometimes testing is done through third party testing houses and in some cases even handled by distributors or publishers.

In addition to testing in-house or through testing partners, most distributors also require the application to be certified. The main goal of certification is to reduce the number of poorly tested applications on the market. Certification is required when the application developer wants to sign the MIDlets or when using carrier-specific APIs. Some sales channels require the application to be certified through the Java Verified Initiative [49]. The purpose of the Java Verified Program (JVP) is to provide developers with a central facility for quickly and easily getting their MIDP applications to market while having their work tested for a level of quality defined by the Unified Testing Initiative[16] (UTI). The developers can test their applications through different testing partners who have partnered with the JVP. The pricing can very different but usually ranges around $150 per device [49]. After the application has passed testing, the application will be digitally signed by the tester and the application has the opportunity to be showcased in the online catalogues of the Member Companies.

6.2. Marketing phase

6.2.1. Understanding the business model

The business models of mobile applications have taken a long time to establish and are still considered immature and often not understood well.

In general, applications can be divided into two main categories:

- Mass-market oriented applications
- Enterprise-oriented applications

Examples of enterprise-oriented applications can be various rich clients accessing backend business systems. Enterprise-oriented applications usually work in a very controlled environment, can utilize the delivery mechanisms set by organization's policies and target certain devices. The revenue models for enterprise applications are simpler and more similar to traditional software revenue models, examples of

[15] The testing criteria that is used by third party testing companies can also be adopted for in-house testing. Testing process could use lead-devices. If a lead device passes the test, several "secondary" devices can also be considered to pass. [49]

[16] A cooperative effort formed by Sun Microsystems and mobile device manufacturers such as Motorola, Nokia, Siemens, Sony Ericsson, Vodafone Group and LG Electronics. The UTI has developed specific testing requirements related to the operating characteristics of Java applications for mobile devices such as handsets [49].

such can be a licensing fee for the developer or even just development fee for the application.

Most common examples of mass-market applications are games and various utilities such as email clients, video players, world clocks, dictionaries etc. The end environment and often delivery mechanisms in case of mass-market oriented applications are not usually directly controlled by developers. An understanding of the various problems of the MIDP platform when producing applications with a mass-market appeal is very important.

6.2.2. Sales channels

It is very often the case that developers do not have direct relationships with end-users, carriers or even some content provider portals. The main sales channels for developers can be divided into distributors, aggregators and publishers.

- Distributors include carriers, content portals and any other parties having direct channels to end-customers. As a distributor possesses the channel to end users, dealing with distributors directly means fewer delays and being more close to end-users.
- Aggregators are a certain type of distributors who have access to more than one channel. An example of an aggregator can be a mobile content portal allowing application downloads for subscribers of different mobile networks.
- Publishers are more novel than distributors or aggregators and have emerged due to the market reaching a state of making publishing profitable. The need for publishers rises with the expansion of the market and rise in the number of distributors. Publishers are similar to a traditional PC applications (or games) publishers. It is often the case that publishers work very closely with development companies providing new ideas and even funding which in form of recoupable advances against royalties. Publishers and developers usually work on a revenue share basis: publishers distribute the application through various channels, gather all revenues and then share an agreed percentage with the developer.

6.2.3. Revenue models

This thesis focuses mainly on mass-market applications. The importance of understanding the revenue models lies in the design decisions that have to be made in early application development stages. Examples of such design questions are

deciding if the application should try to utilize the connectivity or minimize network traffic usage or be designed for impulse purchases.

The different revenue models for this type of applications are presented in the table below [2]. The term *application provider* refers to any party who precedes the carrier in the mobile application value chain. It can be the development company, aggregator or a specialized publishing company.

Revenue method	**Analysis**
Carrier downloads	**Description**: The most popular of the different methods is offering application downloads for users through carrier's portal (also known as "carrier's deck"). Typically a mobile phone user can browse a list of applications and then download them over-the-air (OTA) to his/her mobile phone directly. Purchases from operator portals are often impulse purchases and applications should be designed accordingly.
	***Advantages**:* ▪ There is a trust relationship between the end-user and the carrier. ▪ The download mechanism is simple for the end-user and usually well-managed by the carrier. ▪ The billing is easy and quite fraud-safe.
	***Disadvantages**:* ▪ Carriers often take a considerable revenue cut from developers and other involved parties – usually around 40-60% of the sales price [50]. ▪ Users are often limited to the carrier's choice of applications.
	***Examples**:* most operators have decks for downloadable content

Revenue method	Analysis
Carrier revenue shares	**Description:** Some Java ME applications utilize the wireless connection to fulfil their core functionality. For example a game might download new levels each time an old one has been completed or an on-device portal might periodically download new content. The revenue from data volume sent and received or in some cases, airtime, might then be shared with the application providers. In this scenario, the application itself could be made available for free or for a reduced price. Revenue share applications should be designed to take the maximum of the user experience retaining the user's attention for longer time spans.
	Advantages: This kind of charging is common with other GPRS services, thus it feels natural for users[17].
	Disadvantages: ▪ Revenue sharing is not very widespread. ▪ Revenue sharing requires very close working relationships between the carriers and the application providers and might be difficult to set up.
	Examples: multiplayer games, on-device portals
Downloads from the application developer directly	**Description**: When downloading an application from the carrier, the choice of applications is often limited to what a carrier has decided to offer. In direct downloading scenario, the user downloads the application to the phone from an application provider's website. For this, the operator must allow internet access outside of the operator's portal. Alternatively the application might be downloaded to the end-user's PC and then delivered to the phone using infrared, serial cable or other phone-specific medium.

[17] Not all carriers, who have GPRS installations, are charging on the basis of volume of data sent and received. There are carriers who are also basing the charges on airtime (For example: Vodafone France).

Revenue method	Analysis
Downloads from the application developer directly (continued)	**Advantages**: ▪ Direct downloading makes it possible for the application provider to partially cut the carrier out of the revenue chain. ▪ Direct downloading might mean less trust for the end-users.
	Disadvantages: ▪ It is difficult to set up a billing relationship with the end-user. ▪ There might be copy protection problems, especially when users download applications to their PCs for off-line installation. ▪ The content provider has to manage OTA provisioning backend. ▪ Access to third party content portals might be blocked by the carrier (walled-garden[18]). ▪ Downloads for offline installation are very difficult for the application developers to support[19].
	Examples: different non-carrier content portals

6.2.4. End-user billing

End-user billing is one of the most problematic aspects of selling mobile applications and usually involves a specialized billing service provider. The most common of the revenue models has been getting paid per download, other options available for payment are Monthly Recurring Charge (MRC) or pay per use and if available, channel-specific billing APIs. In case of developer using a distributor or publisher to market the application, the billing is usually not handled by the developer. Most commonly it is handled by the carrier in case of which the charges will appear on the end-user's phone bill.

[18] A walled garden, with regards to media content, refers to an closed set of exclusive set of information services provided for users [21]. In this context a walled garden denotes a closed network in which the users are only allowed to access certain web addresses.

[19] While OTA is a defined standard for Java ME MIDP, other methods for installing applications might include Bluetooth, IrDA, serial cable and others all having device-specific aspects (usage of special software, cables etc.)

The payment methods that are accessible to developers are premium SMS or on-line credit-card payments and usually require partnering with an SMS aggregator[20] or a micro-payment solutions provider such as Sharewire [51]. In come cases premium phone numbers might also be used.

From the technical side, the JCP has been working towards having a generic way to conduct payment transactions. It has released an optional package for developers to be able to gain control of payment operations from the Java ME applications directly by using a standard Payment API [45].

Another specification which has not been officially released yet, is named Event Tracking API [46] and enables developers to access carrier-side event processing systems in a standard way. While the Web Services API [47] can also be used to achieve the same, the support for web services is not so prevalent on Java ME devices [17].

6.2.5. End-user education and support

Educating end-users can be a difficult task but, at the same time, is very important in delivering applications to consumers.

Some key areas where user's might need support are:

- Configuring the phone to be able to access the internet (CSD or GPRS settings).
- Downloading and launching the applications.
- Providing support for application functionality related problems and removal.

In case of using distributors and publishers, the support for downloading and for other areas is usually not handled by the developer directly.

6.3. Discovery and installation phase

6.3.1. Availability of OTA installation

MIDP 1.0 specification does not require an Over-The-Air installation mechanism to be available for MIDlet installations, it instead defines a recommendation document called *"OTA User Initiated Provisioning Recommended Practice"* which states that devices are only expected to have OTA capability. MIDP 2.0 specification requires MIDlets to be installable using OTA provisioning.

[20] SMS Aggregators serve as gateways for companies who want to use SMS services. Usually SMS aggregators act as SMS resellers and support a number of different carriers.

Most MIDlet portals have applications downloadable from the web page, if the user does not have an OTA Provisioning capable phone, the only way to deliver an application would be to use proprietary tools and other mechanisms – infrared, Bluetooth or COM port connection. Non-OTA methods are not standardized by the MIDP specification and the behaviour of the handset after installation or its ability to read a descriptor or notify installation status cannot be relied upon in these cases.

6.3.2. Discovery and Installation

The CLDC specification requires all devices to have *Application Management Software* (AMS) available that can retrieve, launch and delete applications on the device. The AMS is implemented natively in the device and can be sometimes be a integrated into the web-browser on a phone.

Application discovery and installation usually can be divided into the following steps[21].

1. The end-user's mobile phone is sent a link to the provisioning server via WAP SI or SMS. This step might be omitted if the user has clicked on a direct download link. The link usually points to a provisioning service and not directly to the application descriptor as provisioning usually is preceded by selecting a correct version for a handset.
2. When the link is triggered a request is made to the provisioning server
3. The provisioning server authenticates the user and looks up the most appropriate port for the user and sends back a JAD file.
4. The user device then presents a download dialogue stating the name of the application, the vendor, size and other parameters.
5. If the user chooses not to install the application or the device finds the parameters inside the JAD inappropriate, the installation can be cancelled without downloading the application JAR itself. If the user chooses to install the application, a request is made back to the provisioning server to download a JAR file from the location specified in the descriptor file.
6. The on-device Application Management Software downloads and installs the application.

[21] The provisioning process is described in detail in the MIDP 1.0 and 2.0 specifications. If the provisioning backend is based on Java, it could be based on JSR 124, J2EE client provisioning [4].

7. The AMS then reports the installation status back to the provisioning server for billing purposes and reporting. It is also possible to specify a URL which is used to report deletion status of the MIDlet.

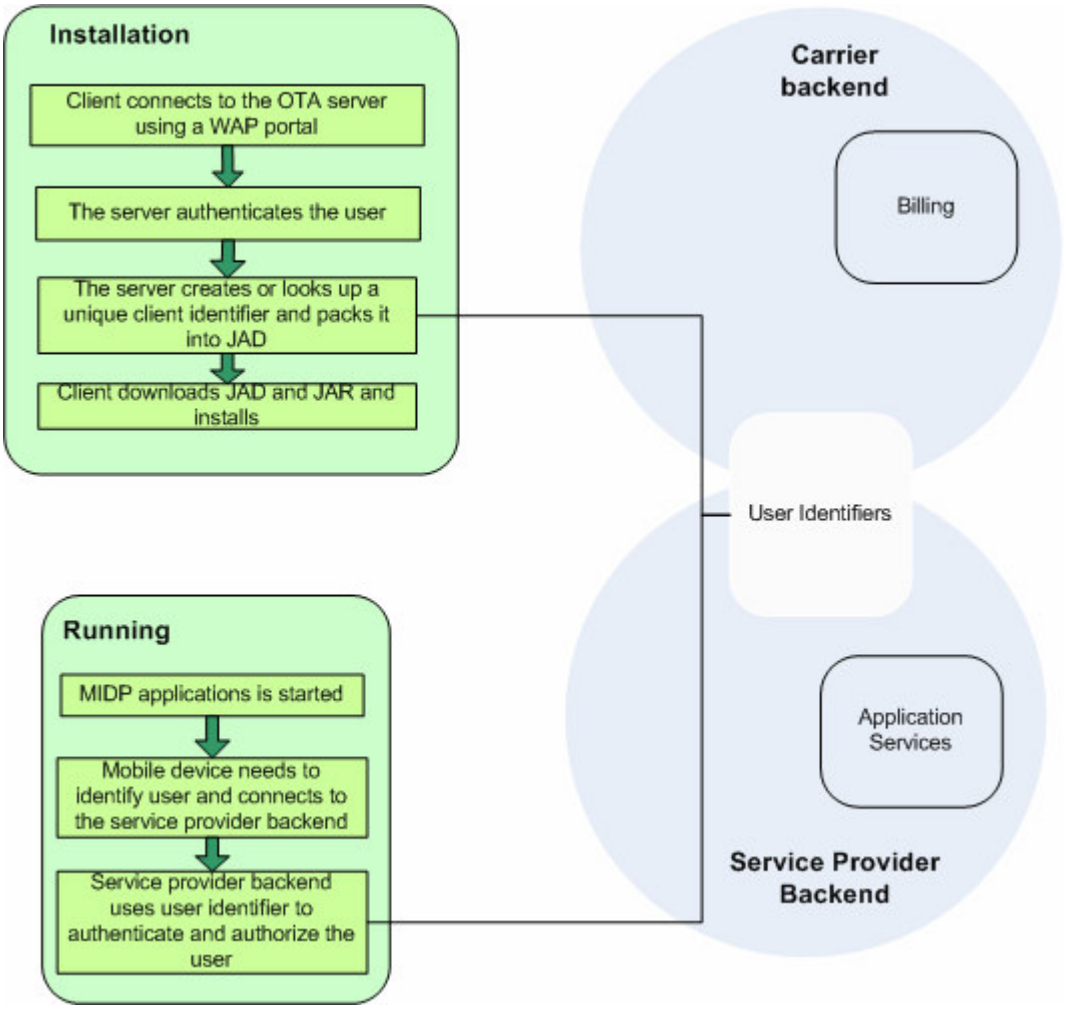


Figure 16 – Example of an OTA installation process

After the application download has completed, most handsets present a dialogue for the user to start the application. Since the MIDP specification does not mandate the behaviour for MIDlets after they have been downloaded on the device, the exact behaviour can vary.

6.3.3. Memory limitations

There are three kind of size limitations which can occur when installing mobile Java applications:

- OTA downloading limitations may be imposed by operator gateways. Some operators have a policy to limit the maximum download size through the WAP gateway. These limitations are usually results of the limits on the WML pages and also results in a carrier specific limit on the size of the Java application that can be downloaded.
- The MIDP does not specify the minimum JAR file size that the device implementers must support. As a result, a variety of devices exist with various limits. Most common limitations are: 32 kB, 64 kB and 128 kB. The OTA specification [8, 38] states that a MIDP compatible device must reject the installation if the device has insufficient memory for the MIDlet Suite.

  Most mobile devices use storage space dynamically and a device might reject a MIDlet installation even if there are no manufacturer limitations but the user has several MIDlets installed and there is not enough room to install another.
- The available memory budget for persistent data is dynamic and can vary runtime (for example, the amount of free memory will increase when a MIDlet is removed from the system). Still sometimes a MIDlet might require a certain amount of memory to be available for persistent data. The OTA Provisioning specification defines an application descriptor attribute[22] for the MIDlet to be able to reserve a certain amount of memory upon installation. A recommended approach is to require as little data from the MIDP environment as possible because it will greatly increase the likelihood of successful MIDlet installations.

Some of the phones and the maximum supported size for the application are given in the table below, based on [54].

Phone	**Maximum Size of JAR**
Nokia S40 v1 (3300 etc.)	64 kB
Nokia S40 v2 (6230 etc.)	128 kB
Sharp GX22	100 kB

About 15 % of the games for handsets are smaller than 64 kB and about 35% are smaller than 128 kB [54].

[22] The descriptor attribute: MIDlet-Data-Size, p. 434 [8]

Size limitations can cause different and often unexpected problems during MIDlet installation. The MSA specification addresses these issues and offers the following clarifications to earlier specifications.

Parameter	Required support
Minimum supported JAR download size	300 kB
Minimum supported JAR install size	300 kB
Minimum supported MIDlet Suite JAD size	10 kB
Minimum supported size of attribute name	512 bytes
Minimum supported size of attribute values in JAD or JAR manifest	2048 bytes
Minimum supported JAD attributes	128
Minimum number of MIDlets in a MIDlet Suite	5

6.3.4. Client identification

A limitation of the MIDP platform is the inability to access network or phone-specific unique identifiers. This affects:

- application installation (client needs to be identified for billing purposes)
- application communication with the backend (client needs to be able to authenticate itself to the server-side with the least possible inconvenience for the user).

The different means of identification are also outlined in the table below.

Identifier	Description
IMEI number	IMEI number, although accessible for GSM operators, is not accessible programmatically from MIDlets and thus cannot be used as an identifier.
IMSI number	IMSI is not accessible programmatically and cannot be used to identify a mobile device.
MSISDN number	MSISDN number (commonly referred to as the user's mobile phone number) would be the best way to identify a mobile phone client. It is not accessible programmatically.

Identifier	Description
IP address	The IP addresses in mobile networks are commonly assigned dynamically and cannot be used as unique identifiers. Identifying mobile clients based on IP addresses would require tight integration with a carrier. There are also no standard ways of access IP addresses programmatically from the MIDP environment.

MIDP does not provide any simple ways to identify mobile clients and the only viable workarounds require integration with carriers. A common approach is to use the carrier's WAP portal to identify the user and then communicate the identifier to the MIDlet via JAD or other configuration files dynamically during installation. Client identification places additional demands on the OTA provisioning server to be able to dynamically embed identifiers into the MIDlet descriptor as well as raises some security concerns. When the user identifier is created dynamically during installation on carrier's side, it also needs to be communicated to service providers in case the MIDlet only installs from the operator but requires data from third party service providers. The problem with this approach is that since the MIDlet does not have access to phone or network-specific identifiers, it is not later able to verify that it is working on the handset that it was downloaded to. Ways to identify clients in this case would be to use SMS messages or phone calls if such functionality is available. While HTTP basic authentication could also be used during MIDlet installation, it is often not viable for operational reasons – it is difficult to distribute credentials and not very convenient for the user.

6.3.5. Copy protection

Copy protection in this context is defined as duplication of the application between end-users. While mobile applications work in a closed and more controlled environment than other types of applications (e.g. desktop applications), some protection can be offered by device manufacturers. Some of the devices do not allow the user to access the on-device Java application storage and prevents users from copying and redistributing. Unfortunately this behaviour is not the same on all devices.

Although several preventive measures might be taken, none of them can ensure full copy-protection. This problem might be alleviated by the *Security and Trust Services*

API (SATSA) [36], which provides smart-card communication, digital signatures and basic user credential management with an addition of a general purpose cryptography API. A copy protected MIDlet using SATSA is delivered in two parts by the operator:

1. a MIDlet application handling user interaction and resource-intensive tasks
2. a smart card application which manages the security of business logic

When an application is copied over to another phone using IrDA for example, the user who receives just a copy of the application, does not have a corresponding smart-card application and the MIDlet would not work.

6.3.6. Application removal

The MIDP specification requires the handset vendors to provide users with means to remove installed applications. Because devices have very different user interfaces for MIDlet handling, the screen-flow and other details are device-specific. The specification leaves the concrete behaviour of how a device should inform the user of the consequences of the operation to be provided also by the implementers. Application providers can specify an optional attribute in the application descriptor during installation that the handsets can use to report the removal to the backend server. The specification requires the implementation to report the deletion to the backend server, but leaves it up to the concrete implementation to provide retry mechanism, therefore the status reports are not very reliable.

6.4. Running phase

The application running phase starts when the application has been successfully downloaded and installed on the device. This is the point when the application will be used and the efforts spent on ensuring that the application would work on specific handsets is put to a test.

MIDlet applications can also specify an icon to be displayed by the handset next to the name of the application for better user experience. The MIDP specification although does not define any concrete requirements for the icons, causing the problem that when the icon is of unsupported size for a specific handset, it might not be displayed. The MSA specification requires implementations to support sizes of 16x16 and optionally 24x24.

6.4.1. Lifecycle

The lifecycle of MIDlets is managed by the on-device Application Management Software that calls the MIDlet's lifecycle methods. The MIDP specification has defined the following lifecycle methods:

- `startApp()` method is called by the AMS when the MIDlet needs to start. At this point the MIDlet should acquire any resources it needs and begin performing the service.
- `pauseApp()` method is called when the AMS wants the MIDlet to significantly reduce the amount of resources it is consuming so that they may be temporarily used by other functions on a device such as a phone call or running another MIDlet. The MIDlet should reduce the resource as much as possible and stop performing the service. The *pauseApp()* method is only available since MIDP 2.0 and has not been reliably implemented across devices [50].
- `destroyApp()` method is called when the AMS has determined that the MIDlet is no longer needed or when memory is needed for a higher priority application. The MIDlet should then optionally save its state and release resources. This method might also be called by the user if the exit key is pressed accidentally which makes it a good place to save state. It is critical to be able to save state quickly, some implementations allow up to 5 seconds for exiting, if the process does not complete in this time, it will be killed by the AMS [43, 50].

A MIDlet can signal the application management software whether it wants to run or has completed but has no knowledge of other MIDlets through the MIDlet API.

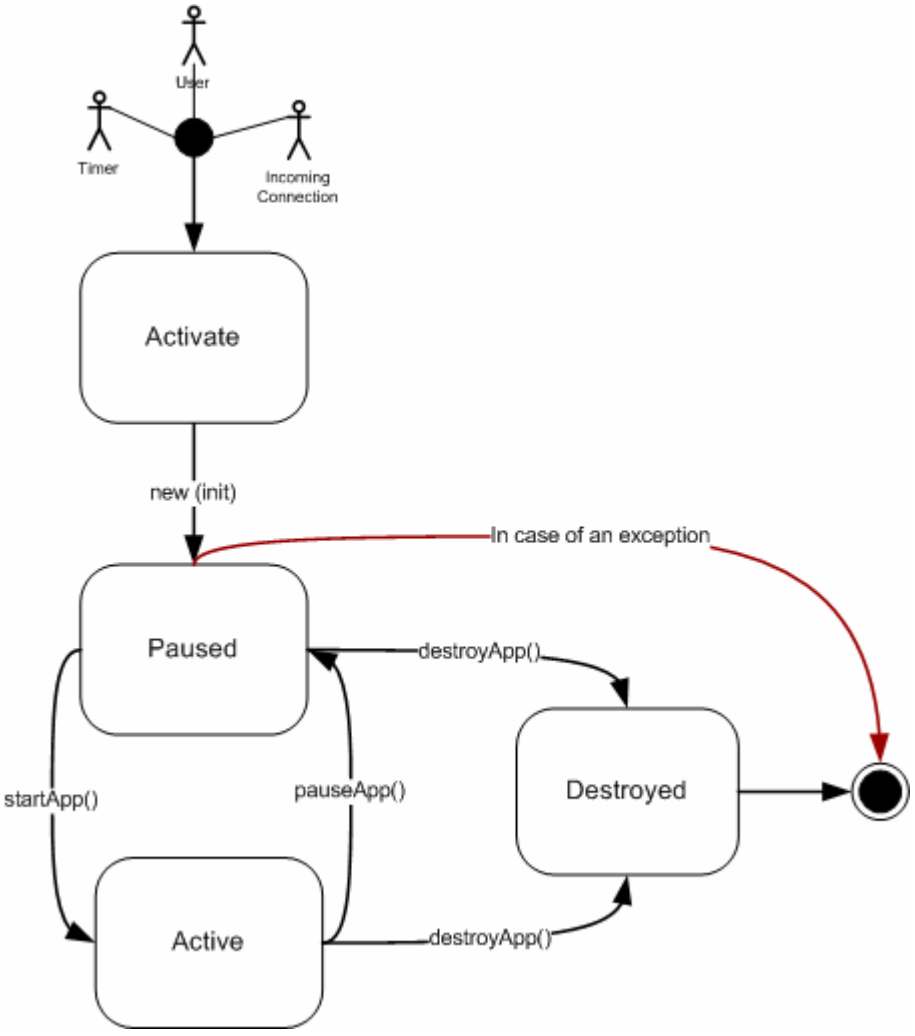


Figure 17 – The lifecycle of a MIDlet

6.4.2. Connectivity

The Java SE has an extensive networking API but is too complex for small resource-constrained devices. The CLDC specification adds a set of classes named the *Generic Connection Framework (GCF)* which forms a lightweight networking API for mobile devices. The GCF supports most popular data protocols on wide area networks, including the IP network and wireless messaging network. It also supports local connectivity protocols such as Bluetooth and the infrared port. The GCF has been so successful that it is now also available under J2SE as an optional package (JSR 197) [1, 4]. All network connections and data streams in the GCF are represented as Java interfaces. Device vendors can develop the most efficient implementation of those interfaces in their Java ME runtime. CLDC layer does not itself provide any implementations which are provided by the profile layer.

MIDP 1.0 implementations are required to support only access to HTTP 1.1 servers and services[23]. The HTTP protocol subset may be implemented either using IP (TCP/IP) or non-IP (WAP, i-Mode) protocols. The MIDP specification requires the underlying protocol to be made transparent from the client application as well as internet server. If the mobile phone is only able to use WAP or i-Mode protocols, from the application developer's point of view, it would still be using just HTTP communication. What stems from here is that the developer has no access to more native connectivity (such as socket connection) that might be the only way to talk some protocols that backend services support. In such cases, intermediary gateways might be used to translate HTTP into other protocols.

Developers should also account for the characteristics of mobile networks such as slow data rate, unreliable connections and the possible costs that might occur for the user when sending or receiving data. All network operations are suggested to be made in a separate threads not to block the user interface. Carrier gateways might also sometimes cache responses or even add, modify or remove requests from HTTP headers, as a consequence, developers should rely on own protocols not on certain headers in HTTP requests.

It is also important to note that selection of bearers or modifying connection settings in the handset is not possible from the MIDP environment. There is no way from a MIDP application to change the GPRS access point for example.

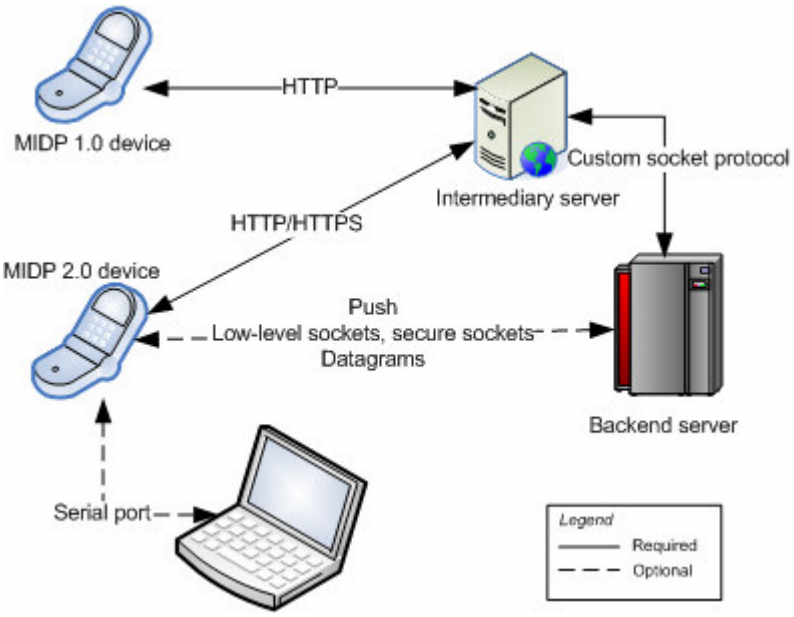


Figure 18 – The various connectivity options for MIDP applications

[23] There are also exceptions to the rule. Early Samsung devices sold by US carriers might have their networking layer restricted and in some cases even blocked. [50]

In case a Wireless Messaging API optional package is available, MIDP devices can also communicate with the backend servers using messaging. Neither the MIDP nor WMA specifications require a certain connection protocol to be implemented. The situation is clarified by JTWI that requires the support for SMS or CBS messages (as specified in WMA 1.1) in appropriate networks. In addition to requiring connectivity using different messaging protocols, JTWI requires support for Push SMS for MIDP 2.0 GSM/WCDMA phones.

Another set of clarifications will be brought by the MSA specification. The different connectivity options are summarized in the table below.

Protocol	Required/Optional (MIDP)	Required/Optional (MSA)
HTTP client	Required	Required
HTTP over SSL v3.0 client	Optional	Required
HTTP over TLS v1.0 client	Optional	Optional
HTTP over WTLS client	Optional	Optional
TCP socket client	Optional	Required
TCP socket server	Optional	Optional
TLS v1.0 client	Optional	Optional
SSL v3.0 client	Optional	Required
UDP	Optional	Optional
SIP	Optional	Required
SMS	Optional	Required
MMS	Optional	Required
CBS	Optional	Optional
Bluetooth L2CAP client	Optional	Required
Bluetooth L2CAP server	Optional	Required
Bluetooth RFCOMM client	Optional	Required
Bluetooth RFCOMM server	Optional	Required
Bluetooth OBEX client	Optional	Conditionally mandatory
Bluetooth OBEX server	Optional	Conditionally mandatory
IrDA OBEX client	Optional	Conditionally mandatory
IrDA OBEX server	Optional	Conditionally mandatory

The MIDP 2.0 specification introduces an new feature called push registry which mandates that MIDlet applications can be auto-launched by listening to either

incoming events or using a timer. On the other hand, the MIDP 2.0 specification does not require support for any protocol and developers cannot count on a certain protocol or service being available on a device. The situation is improved by the MSA specification which requires Push Architecture support for the following connections:

- Bluetooth RFCOMM
- Bluetooth L2CAP
- SMS
- MMS
- SIP

6.4.3. Memory

The types of memory that a MIDlet application is usually interested in are:

1. Heap memory (object storage)
2. Non-volatile memory for on-device data storage accessible to MIDlets
3. Device memory that can be used for MIDlet installations

MIDP specification mandates the following memory requirements:

Parameter	**MIDP 1.0**	**MIDP 2.0**	**JTWI 1.0**	**MSA**
Volatile heap for the Java runtime	32 kB (should[24])	128 kB (should)	256 kB (should)	1024 kB (required), 2048 kB (recommended)
Non-volatile memory for data persisted by the application	8 kB (should)	8kB (should)	30 kB (should)	64 kB (required)
Minimum number of record stores	N/A[25]	N/A	5 (required), if memory available	10 (required)

[24] Notes that the support is recommended but not required
[25] Not specified

Parameter	MIDP 1.0	MIDP 2.0	JTWI 1.0	MSA
Minimum supported download/install size	N/A	N/A	64 kB (should)	300 kB (required)

It is easy to observe that the MIDP specification itself just recommends the minimum memory budgets which devices need to conform to. The availability of heap in reality varies and developers cannot count a certain amount of memory to be available.

Phone	Heap Memory
Nokia S40 v1 (3300 etc)	370 kB
Nokia S40 v2 (6230 etc)	512 kB
Sharp GX22	512 kB

Most mobile devices do not have a traditional file system. However, the MIDP specification requires implementations to provide persistent storage. The persistent storage mechanism can vary from device to device but the programming interface does not change. This way MIDlets are more portable than when using native storage mechanisms. The MIDP storage facility is based around a concept of *Records* and class `javax.rms.RecordStore`.

- A *RecordStore* is a collection of *Records.*
- *Records* in essence are just *byte* arrays which have unique identifiers. This means that all data for storage must be packed into byte arrays for storage, then unpacked to rebuild and use as needed by the MIDlet.
- The programming interface provides methods to create, remove RecordStores and perform operations to add, remove or update the records in a *RecordStore*.

Important considerations with local data storage are:

- Data access speeds may greatly vary
- The amount of RecordStore space depends wholly on the device and is generally shared with the MIDlet storage
- *RecordStores* can in most cases only be shared by MIDlets that belong to the same MIDlet Suite. The MIDP 2.0 also adds support for sharing *RecordStores* between MIDlet Suites.

Many current mobile phones have more than several megabytes of memory. Although, it is common that MIDlets are allowed to only access a small portion of it. On-device storage space might greatly vary and *MIDlet-Data-Size* properties in the *JAD* file should be used to communicate the memory requirements to the AMS. Although commonly used, developers should not rely on the runtime memory information through the API to verify the amount of memory available due to implementation errors. The most sure and recommended way of verifying that a certain amount of memory is available is to fill the memory until the memory is full, then delete and base further calculations on this knowledge [50].

First MIDP devices lacked a compacting garbage collector. As a result, after an application has been running for a long time, the heap can become so fragmented that allocation of larger objects is not possible. The latest JVM implementations (CLDC Hotspot) use much better garbage collection algorithms and a compacting garbage collector. Still developers cannot rely on a compacting garbage collector to be present and need to optimize object allocation accordingly.

6.4.4. Media content

There are 4 main types of media that can be used in a MIDP application:

- Text
- Image
- Sound
- Video

The only available image format that is required to be supported by both MIDP specifications is Portable Network Graphics (PNG) version 1.0 without support for transparency. Writing to the lowest common denominator, developers can only rely on PNG support. JTWI mandates that compatible devices must also support ISO/IEC JPEG together with JFIF noting that lossless PNG might consume too much memory space than is necessary for most cases. JTWI although defined that if JPEG format would become encumbered with licensing requirements, the support is optional. The MSA specification requires JPEG to be supported by all compatible handsets. The JTWI further clarifies and requires support for PNG image files with a bit-depth of 1, 2, 4, 16, 24 and 32 bits per pixel and PNG as well as JFIF image formats to support images up to 32768 total pixels.

One other cause of different problems is the failure of the MIDP or JTWI specifications to require a certain size of image objects to be supported. This might

mean that a PNG image which is 10kB of size, works fine on one device, but fails to show on another. MSA specification clarifies image object and requires minimally images of at least 32 kB to be supported. To optimize image delivery, a common approach is to create film strips or tile maps for images instead of many small ones. JPEG format required from JTWI is also important for capturing shots from video playback.

Audio content is supported on MIDP platforms with the addition of a MMAPI optional package. It should be noted though that the support for different content types has not been specified and developers cannot assume a certain content type to be supported. MIDP 2.0 defines WAV format[26] to be supported by device platforms but **only if** the device platforms support sampled sounds, additionally SP-MIDI format is required but **only if** synthetic sound support is provided. The MSA specification clarifies the situation requiring compatible implementations to support sampled audio as well as the wave format. JTWI requires the support for MIDI feature set as application are greatly enhanced by suitable audio services. Another aspect of playing sounds is the ability to control the volume. While sound formats might be supported, volume controlling through the Java implementation might not be supported. The situation is clarified by JTWI specification which requires all compatible implementations to support volume control.

Video support has the same kind of issues as audio. The specifications lack a lot of clarifications and can cause a lot of confusion. Media support which has been added since MIDP 2.0 with the addition of the MMAPI, is protocol and content format agnostic. Video support is completely up to implementers and often does not have any correlation with the natively supported formats on the handset.

The MIDP 2.0 specification also leaves the access to different content types to be decided by implementations making the supported for various content types differ on a large scale. JTWI mandates that all MIDP implementations must provide HTTP 1.1 support for all supported media types. For example, if a device supports *audio/x-wav* (wave audio), *audio/mpeg* (mpeg standard audio) or others, HTTP 1.1 must be supported to access them.

6.4.5. User interface

The user interface API (LCDUI) of MIDlets is divided into two: high-level and low level. When using high-level user interface elements, a lot of control is given to the

[26] 8-bit, 8KHz, mono-linear PCM wav format

native implementation. While implementations provide native look and feel to common elements such as text fields, lists and forms, in many cases their usability is not good enough. For example, the process a user has to go through to enter text into a text-field using a Nokia Series 40 phone[27] consists of three steps. Firstly, a user would need to select *Edit* from a menu, then enter text and after that press *Save*. It is easy to see that if the number of text fields is more than one, usability is a very big concern. This is just one example why many developers revert to low-level user interface and choose to draw the elements on the screen themselves.

One major limitation of the user interface available that developers often face is the inability to use the full screen of the device. The MIDP 1.0 specification did not require any full screen access from implementations. As a result, full screen capability was added by third party APIs (for example, Nokia UI API). Some implementers also allow the developers to draw out of the bounds of normal application screen area to accomplish full screen drawing. The MIDP 2.0 specification introduces a special full-screen mode, which developers can use. The specification states that when the canvas is in full screen mode, the ticker and title must not be displayed but leaves it up to the implementations to decide if status bars etc. would still be displayed even if the MIDlet is in full-screen mode.

Soft key mappings are often a point of concern for developers. Different devices map the soft-keys differently causing differences in the user experience. As an example, some phones switch left and right soft keys - when a user would need to press the left soft key to cancel the active function, on some phones a right key press would be required [35]. Until the MSA specification, it has also been not defined how soft-keys should be handled in full-screen mode.

When developers use text boxes and text fields they cannot rely on a certain set of character support for different types of text fields. The situation is somewhat clarified by the JTWI specification by defining required characters that need to be supported by text boxes and text fields as well as URL and Email type fields.

The minimum text size has not been specified in MIDP specifications, thus application developers do not always know how much text input a text box or a text field can accommodate. MSA specification states that the minimum storage capacity of a text box or field must not be less than 100 characters. No consistent behaviour has been specified for cases when the MIDP implementation has truncated some text. MSA

[27] Nokia 3100 was used for the test

requires the visual indication of the truncation to be consistent with the devices native user interface.

Support of native input mechanisms, for example T9 Predictive Text[28] input technology is up to the implementation. Developers cannot assume that any device specific input technology is available to Java graphics elements.

The MIDP or JTWI specifications do not require the MIDP implementation to support canvas, pointer or motion events even if the device hardware does so. The MSA specification requires the support in case the device hardware supports these features. Also the handling of repeat events (when the user holds down a key) on the phone display canvas and simultaneous key presses are optional for implementers. The MSA specification requires this functionality to reduce fragmentation. The MIDP specification also fails to define the specifics of other types of input mechanisms than a keypad and a pointer device. The MSA specification also clarifies this issue and defines the necessary support for jog dials.

MIDP specifications do not specify the layout directives for *Form* and *Item* screen elements in a way that the implementers would properly and uniformly implement the layout directives. MSA specification requires all implementers to implement all layout directives. It should be noted that vertical alignment of text is not included in MIDP API as it is considered too burdensome for implementers to implement.

Double buffering of graphics is optional both in MIDP and JTWI specifications, but required in the MSA specification.

6.4.6. Error handling

While MIDlet developers can make use of standard Java exception handling to cater for exceptions and sometimes errors in the program flow, the specifications do not define the behaviour in case an exception is thrown but not caught. If the device implementation does not properly handle the exception in such a case, the MIDlet can enter into an incorrect state. The MSA specification clarifies the behaviour

6.4.7. Access to native functionality

It is very commonly the case that applications might require access to the native functionality of the device. Such native functionality could be call logs, calendars, to-do lists or any other utility which the handset manufacturer usually provides to

[28] T9 Predictive Text technology supports typing text messages with a single key press for each letter (http://www.t9.com/)

users. As MIDP is a very high-level specification and cannot address native utilities which can greatly vary across different handset families as well as the situation that MIDlets work in a constrained sandbox, they are not allowed access to any such functionality (unless such functionality is provided with an optional package). In the MIDP 2.0 specification, a new platform request method[29] has been defined which can request the device to handle a provided URL natively and could be used to either request the launch of a native browser to update the MIDlet itself or for example to initiate a call. The MIDP 2.0 specification does not require device vendors to implement the platform request functionality which decreases portability and increases fragmentation. The MSA specification aims to require the platform request implementations to support at least browser launching as well as call initiation. As other aspects, the behaviour of the MIDlet when using native functionality can vary. Some handsets do not support concurrent browser and MIDlet applications, others have different behaviour during call initialization and can either stop MIDlets or switch them to the background [1].

In some circumstances, mobile applications might need to use functionality that goes beyond the capability of the platform request method. For example, MIDlets might have need to access a GPS device attached to a handset or utilize device specific operating system features.

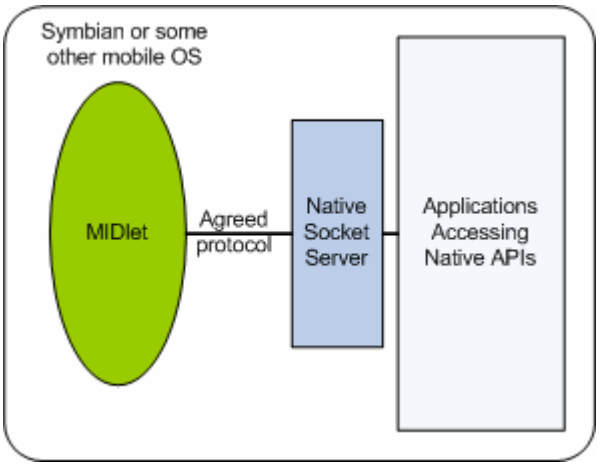


Figure 19 - Accessing native functionality from Java ME applications

Many of the mobile operating systems have closed APIs and there are no options for the developer to access anything beyond the functionality defined in the MIDP specification unless a vendor extension or an optional package adds the needed capabilities. On mobile platforms which are open, specific workarounds might be

[29] boolean platformRequest(String URL), MIDP 2.0 specification, p. 447 [8]

available to bridge MIDlets with native functionality [41]. These approaches can be used in parallel to the platform request functionality provided by MIDP or extend this. For example on the Symbian and Windows Mobile platforms MIDlets can use native proxies through socket communication with the underlying platform.

7. Conclusions

Mobile platforms have become one of the most widespread computing platforms accessible to billions of people. While cellular technologies have been through several evolvement stages, it has been just recently that mobile phone platforms have opened up for third party applications. Java Micro Edition is one of the main technologies that enables the delivery of applications for enterprise as well as mass-market end-users. Since its introduction, the mobile Java platform has gained considerable momentum in the industry and currently about three of every four phones released are able to run Java applications.

Java ME technology provides a lot of functionality through a well-known programming API and allows development companies to leverage the existing Java developer skills.

This thesis has presented an overview of the Java ME technology for small resource-constrained devices, its main concepts and positioned it by comparing it to other wireless technologies.

Mobile platforms are different in nature from more-conventional web-centric software environments. The differences range from the way applications are installed to end-devices to the general environment they are working in. Mobile applications also need to balance the existing technologies capabilities with end-user expectations. As a result, mobile programming efforts are clouded by misunderstandings and false anticipations which can result in poor user experience or project failure.

This thesis has offered an approach to analyse fragmentation and its effect on mobile development projects.

The thesis has also taken a look into several areas of Java ME technology which are important but not usually covered in specifications or other available literature. Java ME has a number of limitations ranging from application installation issues to user interface problems and a selective view of these was presented in the thesis.

Java Micro Edition tehnoloogia probleemistik

Magistritöö

Andres Teder

Resümee

Mobiiltelefone on tänapäeval raske määratleda kui pelgalt sidevahendeid. Pigem võib neid seadmeid käsitleda kui miljardite inimeste kasutuses olevaid arvutusplatvorme. Mobiilitehnoloogiad on teinud läbi pika ja kiire arengu ning jõudnud juba kolmanda generatsiooni andmesidevõrkudeni. Mobiiltelefonid aga on muutunud rakendustele avatud platvormideks alles hiljuti. Java Micro Edition (Java ME) on esimene tehnoloogia, mis on suutnud luua mobiiltelefonidele rikka funktsionaalsusega platvormi ärikriitiliste ja ka meelelahutusliku suunaga laiatarbe rakenduste jaoks. Tänaseks on Java ME tehnoloogia omandanud mobiilirakenduste arendamise valdkonnas juhtpositsiooni ja on installeeritud rohkem kui miljardisse telefoni.

Käesolev uurimistöö tutvustab Java ME tehnoloogia peamisi kontseptsioone ja arengut ning toob välja võrdluse teiste mobiilirakenduste arendamisel kasutatavate tehnoloogiatega.

Mobiilsed platvormid erinevad oma olemuselt konventsionaalsetest, sageli veebipõhistest, platvormidest. Erinevused ulatuvad rakenduste installeerimisviisist nende üldise töökeskkonnani. Mobiilsed rakendused peavad arvestama ka olemasolevate tehnoloogiate pakutavate võimaluste ja lõppkasutajate ootuste vahelist tasakaalu. On üsna tavaline, et ootused mobiiliprojektidele ja teadmised mobiilitehnoloogiatest on väga piiratud. Olemasolevad spetsifikatsioonid või kirjandus ei anna sageli piisavalt laiapiirilist ülevaadet ja ei käsitle olulisi probleempunkte. Valearusaamad võivad muuta rakenduse lõpptarbija jaoks halvasti kastutatavaks või viia isegi kogu arendusprojekti ebaõnnestumiseni.

Käesolev uurimistöö esitles ühte ulatuslikku mobiilitehnoloogiate probleemi, fragmentatsiooni, selle päritolu ja mõju mobiiliarendusprojektidele.

Lisaks on lahatud mobiilirakenduste elutsükliga seotud piiranguid ja probleeme, mille vältimine on mobiiliprojektide edukuse tagamiseks väga oluline.

Glossary

This glossary includes a number of terms and acronyms which have been commonly used throughout this thesis. The sources for the definitions are based on information from [21, 8, 58, 59].

Acronym or term	Term	Definition
.NET Framework	*.NET Framework*	The Microsoft .NET Framework, more commonly known as simply the .NET Framework, is a software development platform created by Microsoft. The .NET Framework is now in version 2.0, which was released in November of 2005 and is the successor to two major previous versions: 1.0 and 1.1. .NET is a Microsoft technology that allows cross-language development and provides a large standard library. Other competing approaches are cross-platform languages, e.g. Perl, using a cross-platform runtime like the Java Virtual Machine, or compile standard ANSI C to each platform.
ADO	*ActiveX Data Objects*	Component object model (COM) object for accessing data sources. It provides a layer between programming languages and OLE DB (OLE for databases, a means of accessing data stores, whether they be databases or otherwise, in a uniform manner), which allows a developer to write programs which access data, without knowing how the database is implemented. You must be aware of your database for connection only. No knowledge of SQL is required to access a database when using ADO, although one can use ADO to execute arbitrary SQL commands. The disadvantage of this is that this introduces a dependency upon the database.
AMS	*Application Management Software*	A generic term used to describe the software on the device that manages the downloading and lifecycle of MIDlets. This term does not refer to any specific implementation and is used for convenience only. In some implementations, the term Java Application Manager (JAM) is used interchangeably.
ASIC	*Application Specific Integrated Circuit*	A chip that is designed to fulfil a specific task in a computer system, e.g. for routing messages in a network.

Acronym or term	Term	Definition
Byte-code		Form of instructions that can be executed by JVMs
CBS	*Cell Broadcast Service*	The CBS service is analogous to the Teletex service offered on television, in that like Teletex, it permits a number of unacknowledged general messages to be broadcast to all receivers within a particular region. CBS messages are broadcast to defined geographical areas known as cell broadcast areas.
CDC	*Connected Device Configuration*	CDC is a smaller subset of Java SE, containing almost all the libraries that are not GUI related. It is also a superset of CLDC.
CLDC	*Connected Limited Device Configuration*	The CLDC contains a strict subset of the Java class libraries, and is the minimal needed for a Java virtual machine to operate. CLDC is basically used to classify the myriad of devices into a fixed configuration
CLR	*Common Language Runtime*	Common Language Runtime (CLR) is the name chosen by Microsoft for the virtual machine plus runtime library underlying their .NET initiative. The CLR is a generalized multi-language, reflective execution engine on which code originally written in various languages runs. As of 2005, over 40 languages were supported.
COM	*Component Object Model*	Microsoft software component technology introduced in 1993. It is used to enable inter-process communication and dynamic object creation in any programming language that supports the technology. COM is often used as an umbrella term encompassing OLE, ActiveX, COM+ and DCOM technologies
DCOM	*Distributed Component Model*	Microsoft proprietary technology for software components distributed across several networked computers to communicate with each other, extends COM technology. It has been deprecated in favour of Microsoft .NET technology

Acronym or term	Term	Definition
EC (JCP context)	*Executive Committee*	The Executive Committee is the group of Members guiding the evolution of Java technology in the Java Community Process. The EC represents both major stakeholders and a representative cross-section of the Java Community. The EC is responsible for approving the passage of specifications through key points of the JCP and for reconciling discrepancies between specifications and their associated test suites.
GSM	*Global System for Mobile Communications*	The Global System for Mobile Communications (GSM) is the most popular standard for mobile phones in the world. GSM service is used by over 1.5 billion people across more than 210 countries and territories.
HLR	*Home Location Register*	The Home Location Register or HLR is a central database that contains details of each mobile phone subscriber that is authorized to use the GSM core network. More precisely, the HLR stores details of every SIM card issued by the mobile phone operator. Each SIM has a unique identifier called an IMSI which is one of the primary keys to each HLR record.
IDE	*Integrated Development Environment*	An IDE is a type of computer software that assists computer programmers to develop software.
IMEI	*International Mobile Equipment Identity*	The IMEI is a 15-digit number which includes information on the origin, model, and serial number of the device.
IMSI	*International Mobile Subscriber Identity*	is a unique number that is associated with all GSM and UMTS network mobile phone users. The number is stored in the SIM. It is sent by the mobile to the network and is used to look up the other details of the mobile in the HLR or as locally copied in the VLR. In order to avoid the subscriber being identified and tracked by eavesdroppers on the radio interface, the IMSI is sent as rarely as possible and a randomly generated Temporary MSI (TMSI) is sent instead.
IrDA	*Infrared Data Association*	The Infrared Data Association (IrDA) defines physical specifications communications protocol standards for the short range exchange of data over infrared light, for uses such as personal area networks (PANs).

Acronym or term	Term	Definition
IEC	*International Electrotechnical Commission*	The International Electrotechnical Commission is an international standards organization dealing with electrical, electronic and related technologies. Some of its standards are developed jointly with ISO. The IEC charter embraces all electrotechnologies including energy production and distribution, electronics, magnetics and electromagnetics, electroacoustics, multimedia and telecommunication, as well as associated general disciplines such as terminology and symbols, electromagnetic compatibility, measurement and performance, dependability, design and development, safety and the environment.
ISO	*International Organization for Standardization*	The International Organization for Standardization is an international standard-setting body composed of representatives from national standards bodies.
JAD	*Java Application Descriptor*	The application descriptor is used in conjunction with the JAR manifest by the application management software to manage the MIDlet and is used by the MIDlet itself for configuration specific attributes.
JAR	*Java Archive*	In computing, a JAR file (or Java Archive) is a ZIP, RAR, or tar file used to distribute a set of Java classes. It is used to store compiled Java classes and associated metadata that can constitute a program.
Java EE	*Java Enterprise Edition*	A programming platform for developing and running distributed multi-tier architecture applications, based largely on modular components running on an application server. The J2EE platform is defined by a specification. (formerly known as Java 2 Platform, Enterprise Edition or J2EE up to version 1.4)
Java ME	*Java Micro Edition*	A collection of Java APIs targeting embedded consumer products such as PDAs, cell phones and other consumer appliances. (formerly referred to as Java 2 Platform, Micro Edition or J2ME)
Java SE	*Java Standard Edition*	A collection of Java Application Programming Interfaces useful to any Java platform programs (formerly known up to version 5.0 as Java 2 Platform, Standard Edition or J2SE)

Acronym or term	Term	Definition
JFIF	*JPEG File Interchange Format*	A standard, created by the Independent JPEG Group and specifies how to produce a file suitable for computer storage and transmission (such as over the Internet) from a JPEG stream
JPEG		A standard method of lossy compression for photographic images. JPEG is the format most used for storing and transmitting photographs on the World Wide Web.
JSR	*Java Specification Request*	A formal document that describes proposed specifications and technologies to be added to the Java platform
JVM	*Java Virtual Machine*	A virtual machine developed by Sun Microsystems which can run Java bytecode.
KVM	*Kilobyte Virtual Machine*	The KVM evolved from a research project at Sun Laboratories. It is a derivative of The Java Virtual Machine Specification, written from scratch in C. The KVM is designed to support standardized, incremental deployment of Java virtual machine features and Java APIs called for by the Java 2 ME architecture.
MIDP	*Mobile Information Device Profile*	A specification published for the use of Java on embedded devices such as cell phones and PDAs. MIDP is part of the Java Platform, Micro Edition (Java ME) framework.
MSISDN	*Mobile Station Integrated Services Digital Network*	The Mobile Station Integrated Services Digital Network (MSISDN) is the mobile equivalent of ISDN. Used as a value, MSISDN refers to the Mobile Subscriber ISDN Number, which is a max 15-digit number. There are mobile phones on the market that can have several MSISDNs in one SIM Card. E.g. a German MSISDN and a French MSISDN in one SIM card would allow the user to be reached under two numbers.
OLE	*Object Linking and Embedding*	Microsoft's distributed object system and protocol. An example of OLE might be allowing an editor to expose part of a document to another editor and then re-import it. Its primary use was for managing compound documents, but it was also used for transferring data between different applications using drag and drop and clipboard operations. OLE technology has been renamed to ActiveX in 1996.
OTA	*Over-The-Air*	Over The Air (OTA) refers to any wireless networking technology.
OTAP	*Over-The-Air Provisioning*	The deployment of applications to a wireless device using a wireless connection (over-the-air).

Acronym or term	Term	Definition
PIM	*Personal Information Manager*	A personal information manager (PIM) is a type of application software that functions as a personal organizer. As an information management tool, a PIM's purpose is to facilitate the recording, tracking, and management of certain types of "personal information".
PDA	*Personal Digital Assistant*	a small mobile handheld device providing computing and information retrieval capabilities for either personal or business use
Portability		Portability is the general characteristic of being readily transportable from one computer system to another.
SKU	*Stock Keeping Unit*	A uniquely identifiable line within a product range.
SMS	*Short message service*	Short Message Service (SMS) is a service available on most digital mobile phones that permits the sending of short messages (also known as text messages, or more colloquially SMSes, texts or even txts) between mobile phones, other handheld devices and even landline telephones. Other uses of text messaging can be for ordering ringtones, wallpapers and entering competitions.
TCK	*Technology Compatibility Kit*	A TCK is a suite of tests, tools and documentation that determines whether or not a product complies with a particular Java™ technology specification.
UI	*User Interface*	The user interface is the aggregate of means by which people (the users) interact with a particular machine, device, computer program or other complex tool (the system).
USSD	*Unstructured Supplementary Service Data*	Unstructured Supplementary Service Data is a capability of all GSM phones. It is generally associated with real-time or instant messaging type phone services. There is no store-and-forward capability that is typical of 'normal' short messages (in other words, an SMSC is not present in the processing path). Response times for interactive USSD based services are generally quicker than those used for SMS.

Acronym or term	Term	Definition
VLR	*Visitor Location Register*	The Visitor Location Register or VLR is a temporary database of the subscribers who have roamed into the particular area which it serves. Each Base Station in the GSM network is served by exactly one VLR, hence a subscriber cannot be present in more than one VLR at a time. The data stored in the VLR has either been received from the HLR, or collected from the MS. In practice, for performance reasons, most vendors integrate the VLR directly to the V-MSC and, where this is not done, the VLR is very tightly linked with the MSC via a proprietary interface.
VM	*Virtual Machine*	Virtual machine in its most general definition is a software that creates an environment between the computer platform and end user in which the end user can operate software.
WAP SI	*Wireless Application Protocol Service Indication*	The Service Indication (SI) content type provides the ability to send notifications to end-users in an asynchronous manner. Such notifications may, for example, be about new e-mails, changes in stock price, news headlines, advertising, reminders of e.g. low prepaid balance, etc. In its most basic form, an SI contains a short message and a URI indicating a service. The message is presented to the end-user upon reception, and the user is given the choice to either start the service indicated by the URI immediately, or postpone the SI for later handling. If the SI is postponed, the client stores it and the end-user is given the possibility to act upon it at a later point of time.
WCDMA	*Wideband Code Division Multiple Access*	W-CDMA (Wideband Code Division Multiple Access) is a type of 3G cellular network. W-CDMA is the technology behind the 3G UMTS standard and is allied with the 2G GSM standard.

Resources

[1] Michael Juntao Yuan, Kevin Sharp, "Developing Scalable Series 40 Applications: A Guide for Java Developers," Addison Wesley Professional, December 2004.

[2] Martin J. Wells, "J2ME Game Programming," Course Technology PTR, March 2004

[3] Michael Juntao Yuan, "Enterprise J2ME Developing Mobile Java Applications," Prentice Hall PTR, 1st Edition, October 2003

[4] Java Community Process, http://www.jcp.org, 19/03/2006

[5] "Connected Limited Device Configuration (CLDC), version 1.0a," JCP Specification, May 2000

[6] "Connected Limited Device Configuration (CLDC), version 1.1," JCP Specification, March 2003

[7] "Mobile Information Device Profile for Java 2 Micro Edition, version 1.0a," JCP Specification (JSR 37), December 2000

[8] "Mobile Information Device Profile for Java 2 Micro Edition, version 2.0," JCP Specification (JSR 118), November 2002

[9] "Connected Device Configuration, version 1.0b," JCP Specification (JSR 36), 2005

[10] "Connected Device Configuration, version 1.1", JCP Specification (JSR 218), 2005

[11] Martin Behovsky, "Java for the Mobile World. Java Platform, Micro Edition – Status and Future Directions," Sun Microsystems, November 2005

[12] Antero Taivalsaari, "Minne menet, Mobiili Java?" Sun Microsystems, December 2005

[13] David Fox, Roman Verhosek, "Micro Java Game Development," Addison Wesley, April 2002

[14] Allen Lau, "The Fragmentation effect," Javaworld, http://www.javaworld.com/javaworld/jw-05-2004/jw-0524-fragment.html, May 2004, 15/05/2006

[15] Pentti J Savolainen, "Fragmentation of Java 2 Platform, Micro Edition in Wireless Space: How to Live with It and Increase Compliancy," Nokia, 2003

[16] J2ME Polish, http://www.j2mepolish.org, 25/03/2006

[17] Sun Developer Network, The Java ME Device Database, http://developers.sun.com/techtopics/mobility/device/device, 25/03/2006

[18] Valentino Lee, Heather Schneider, Robbie Schell, "Mobile Applications: Architecture, Design and Development," Prentice Hall PTR, April 2004

[19] Java Platform Micro Edition, Sun Developer Network, http://java.sun.com/javame/, 14/05/2006

[20] Official BREW Developer Site, http://brew.qualcomm.com, 04/05/2006

[21] Wikipedia, the free encyclopaedia, http://en.wikipedia.org/wiki/, 04/05/2006

[22] "Symbian OS Basics", Nokia Developer Training, Course Pack, 2006

[23] UIQ Developer Community Web Portal, http://developer.uiq.com/devtools_java.html, 05/05/2006

[24] U.S Census Bureau, http://www.census.gov/ipc/www/world.html, 07/05/2006

[25] Geohive, http://www.geohive.com, 07/05/2006

[26] Wireless Intelligence, http://www.gsmworld.com/news/press_2005/press05_21.shtml, September 2005, 07/05/2006

[27] Rhett Butler, "Cell phones may help save Africa," http://news.mongabay.com/2005/0712-rhett_butler.html, July 2005, 07/05/2006

[28] Garmeen Foundation, http://www.gfusa.org, 07/05/2006

[29] "Nokia, S60 Platform Introductory Guide, version 1.1," November 2005

[30] "Interest in embedded Linux remains low, survey finds", http://www.eet.com/news/semi/showArticle.jhtml?articleID=184425894, EETimes, April 2006, 07/05/2006

[31] Gartner, Media Relations, http://www.gartner.com/press_releases/asset_132473_11.html, 08/05/2006

[32] Dan Fox, John Box, "Building Solutions with the Microsoft .NET Compact Framework: Architecture and Best Practices for Mobile Development," Addison Wesley, October 2003

[33] Ivo Salmre, "Writing Mobile Code: Essential Software Engineering for Building Mobile Applications," Addison Wesley Professional, February 2005

[34] Palm OS Developer Portal, http://www.palmos.com/dev/start, 16/04/2006

[35] Roger Riggs, Antero Taivalsaari, Jim Van Peursem, Jyri Huopaniemi, Mark Patel, Aleksi Uotila, Jim Holliday Editor, "Programming Wireless Devices with the Java™ 2 Platform, Micro Edition, Second Edition," Addison Wesley, June 2003

[36] "Security and Trust Services API (SATSA), version 1.0," JCP Specification (JSR 177), September 2004

[37] C. Enrique Ortiz, "Elements of a Typical J2ME MIDP Business Application," http://www.microjava.com/articles/techtalk/midp, October 2001, 22/04/2006

[38] "Over The Air User Initiated Provisioning Recommended Practice for the Mobile Information Device Profile, version 1.0," Sun Microsystems, May 2001

[39] "Mobile Service Architecture for CLDC, public review, version 0.1," JCP Specification (JSR 249), May 2006

[40] "Java Technology for the Wireless Industry (JTWI), version 1.0," JCP Specification (JSR 185), June 2003

[41] Arvind Gupta, Martin de Jode, "Extending the reach of MIDlets: how MIDlets can access native services, version 1.1," June 2005

[42] Adobe Website, http://www.adobe.com/products/flashlite, 07/05/2006

[43] Forum Nokia, Developer resources, http://www.forum.nokia.com/main.html, 08/05/2006

[44] Chris Fletcher, "Smart-phones & Vertical Industries," Smart Phones Summit, April 2006

[45] "Payment API (PAPI) for Java 2 Platform, Micro Edition, version 1.1.0," JCP Specification (JSR 229), 2005

[46] "Event Tracking API (ETA) for Java 2 Platform, Micro Edition, version 1.0.0," JCP Specification (JSR 190), 2005

[47] "J2ME Web Services API, version 1.0," JCP Specification (JSR 172), 2003

[48] Microsoft, http://www.microsoft.com, 20/05/2006

[49] Java Verified Program Home, http://javaverified.com/, 16/05/2006

[50] Rodney Aiglstorfer, "Troubleshooting J2ME," mFoundy Inc, JavaOne Conference 2006

[51] Sharewire, http://www.sharewire.net, 16/05/2006

[52] PhoneScoop, http://www.phonescoop.com/, 16/05/2006

[53] Wireless Universal Resource File, http://wurfl.sourceforge.net/ , 16/05/2006

[54] Stephen Cheng, "Last Ounce of Performance from Your MIDlets", Innaworks, JavaOne Conference 2006

[55] Via Vodafone, Global Partner Home, http://via.vodafone.com, 16/05/2006

[56] Mobile Research, http://www.mobileresearch.net/, 18/05/2006

[57] ProGuard, http://proguard.sourceforge.net/, 18/05/2006

[58] "Wireless Application Protocol, Service Indication, WAP-167-ServiceInd-20010731-a, version 31-July-2001," Wireless Application Protocol Forum, July 2001

[59] "3GPP TS 03.41: Technical realization of Cell Broadcast Service (CBS) (version 7.4.0 Release 1998); Digital cellular telecommunications system (Phase 2+)," ETSI Technical Specification, September 2000

[60] Thawte, http://www.thawte.com/, 19/05/2006

[61] Verisign, http://verisign.com/, 19/05/2006

[62] Nokia, http://www.nokia.com, 20/05/2006

[63] Vodafone, http://www.vodafone.com, 20/05/2006

Appendix I

List of Distributors

Distributor	Web Address
Handango	http://www.handango.com
T-Mobile	http://www.t-mobile.net

List of Publishers

Publisher	Web Address
JAMDAT	http://www.jamdat.com/
Tira Wireless	http://www.tirawireless.com/
Mforma	http://www.mforma.com/
Digital Bridges	http://www.iplay.com/
Unplugged	http://www.unplugged-inc.com/
Fifth Axis	http://www.fifthaxis.com/