LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
Department of Information Technology

ALEXANDER DAVYDOV

Provisioning of MIDlets

Master of Science Thesis

The topic of this Master of Science Thesis was confirmed by the Department Council of the Department of Information Technology on 12.03.2003

Supervisor: Dr. Kari Systä
(Nokia Research Center)

Examiners: Prof. Jan Voracek, Dr. Vladimir Botchko
(Lappeenranta University of Technology)

Tampere, 02.05.2003

Visiokatu 1
33720 Tampere
phone: +358504860717

TIIVISTELMÄ

LAPPEENRANNAN TEKNILLINEN KORKEAKOULU
Tietotekniikan osasto

DAVYDOV, ALEXANDER:
PROVISIONING OF MIDLETS
Diplomityö
Toukokuu 2003

97 sivua, 38 kuvaa, 2 taulukkoa

Tarkastajat: Prof. Jan Voracek, Dr. Vladimir Botchko

Hakusanat: provisioning, MIDlet, MIDlet suite, J2ME, JAD file, JAR file, OTA provisioning, local provisioning, OMA DRM, separate delivery, combined delivery, superdistribution, Bluetooth wireless technology

Java™ 2 Platform, Micro Edition on eräs johtava sovellusalusta, joka mahdollistaa kolmannen osapuolen sovellusten luomisen matkapuhelimiin, kommunikaattoreihin ja taskutietokoneisiin. Java-alusta keskeinen etu on sovellusten dynaaminen asentaminen. Käyttäjä ei ole rajoitettu esiasennettuihin sovelluksiin vaan voi asentaa niitä itse tarpeen mukaan. Tämän diplomityö käsittelee erilaisia Java sovellusten (MIDlettien) lataus ja asennusmenetelmiä.

Diplomityö antaa yhteenvedon merkittävimmistä asennus teknologioista. Pääpaino on MIDP-standardin mukaisella langattomalle asennuksella (Over-The-Air provisioning) sillä se on kaikkein laajimmin käytetty menetelmä. Muita käsiteltäviä menetelmiä ovat WAP Push ja paikallinen asennus Bluetoothin ja Infrapunalinkin avulla.

MIDletit, kuten mitkä tahansa ohjelmat, ovat alttiita laittomalle kopioinnille. Tämä diplomityö kuvaa menetelmiä, joilla laiton kopiointi voidaan estää. Yksi esimerkki on OMA™ DRM standardi. Diplomityö kuvaa myös kuinka kopiointisuojaus voidaan yhdistää olemassa oleviin asennusmenetelmiin.

Java sovelluksia, MIDlettejä, käytetään yhä erilaisimpiin tarkoituksiin jolloin tarvitaan myös uusia asennusmenetelmiä. Yksi tällainen menetelmä on asentaminen erillisistä laitteista. Diplomityö kuvaa useita menetelmiä asentamiseen erillisistä laitteista. Käsitellyr menetelmät pohjautuvat Bluetooth teknologiaan ja yhtä lukuun ottamatta perustuvat standardin määrittelemiin Bluetooth profiileihin File Transfer Profile, Personal Area Networking Profile ja Object Push Profile.

Toinen asennustapa on sovellusten edelleen lähettäminen toiseen puhelimeen. Diplomityö kuvaa kuinka OMA DRM standardi voidaan yhdistää tällaisen asennuksen ja ehdottaa kahta vaihtoehtoista menetelmää. Yksi perustuu Bluetoothin Object Push Profiiliin ja toinen Infrapunalinkin käyttöön. Toinen perustuu multimediaviestiin ja sähköpostiin.

ABSTRACT

LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
Department of Information Technology

DAVYDOV, ALEXANDER:
PROVISIONING OF MIDLETS
Thesis for the Degree of Master of Science in Technology
May 2003

97 pages, 38 figures, 2 tables

Examiners: Prof. Jan Voracek, Dr. Vladimir Botchko

Keywords: provisioning, MIDlet, MIDlet suite, J2ME, JAD file, JAR file, OTA provisioning, local provisioning, OMA DRM, separate delivery, combined delivery, superdistribution, Bluetooth wireless technology

The Java™ 2 Platform, Micro Edition is one of the leading software platforms that allow third-party software development for such mobile devices as mobile phones, communicators and PDAs. An essential benefit of the platform is dynamic application delivery. The user of a mobile device is no more limited to a pre-installed set of applications but can add them as need. Methods for dynamic provisioning of Java applications (MIDlets) are in the focus of this thesis.

The thesis gives an overview of existing provisioning technologies. A special emphasis is put on the most widely used provisioning technique, viz. Over-The-Air provisioning as defined by the MIDP standard. The other methods described include WAP Push provisioning and local provisioning via Bluetooth™ and Infrared links.

MIDlets, like any computer programs, can be copied illegally. The thesis describes one of the technologies that can be used to prevent illegal copying, viz. OMA™ DRM. The thesis shows how the technology can be incorporated into existing provisioning methods.

Diversity and application areas of MIDlets are growing, which creates a need for new types of provisioning. One of these types is provisioning from autonomous devices such as electric appliances and MIDlet dispensers. The thesis proposes several methods for such deployment. They all use Bluetooth wireless technology; all but one are based on standard Bluetooth profiles, viz. the File Transfer Profile, the Personal Area Networking Profile and the Object Push Profile.

Another new type of provisioning is forwarding of MIDlets from one mobile device to another. The thesis demonstrates how the OMA DRM facilitates such type of provisioning and proposes two methods. The local one uses the Object Push Profile over the Bluetooth or Infrared link. The other is based on the MMS or e-mail.

Contents

List of Figures

Abbreviations and Acronyms

ACL	Asynchronous Connection-Less (link)
AMS	Application Management Software
API	Application Programming Interface
CDC	Connected Device Configuration
CEK	Content Encryption Key
CLDC	Connected, Limited Device Configuration
CoD	Class of Device
CSD	Circuit Switched Data (call)
DA	Discovery Application
DCF	DRM Content Format
DRM	Digital Rights Management
FP	File Transfer Profile
GAP	Generic Access Profile
GCF	Generic Connection Framework
GN	Group Ad Hoc Network
GOEP	Generic Object Exchange Profile
GPRS	General Packet Radio Service
HTTP	Hypertext Transfer Protocol
HTTPS	Secure HTTP
IP	Internet Protocol
IrDA®	Infrared Data Association®
J2EE™	Java 2 Platform, Enterprise Edition
J2ME™	Java 2 Platform, Micro Edition
J2SE™	Java 2 Platform, Standard Edition
JAD	Application Descriptor
JAR	Java Archive
JCP	Java Community Process
JSR	Java Specification Request
JVM	Java Virtual Machine
KVM	K Virtual Machine
MIDP	Mobile Information Device Profile
MIME	Multipurpose Internet Mail Extensions
MMS	Multimedia Messaging Service
NAP	Network Access Point
OBEX	Object Exchange (protocol)
OMA™	Open Mobile Alliance™
OPP	Object Push Profile
OSI	Open System Interconnection
OTA	Over-The-Air
PAN	Private Area Networking (profile)
PANU	PAN User
PAP	Push Access Protocol
PDA	Personal Digital Assistant
PPG	Push Proxy Gateway
RMI	Remote Method Invocation
RMS	Record Management System
SCO	Synchronous Connection Oriented (link)

SDAP	Service Discovery Application Profile
SDDB	Service Discovery Database
SDP	Service Discovery Protocol
SIG	Special Interest Group
SIR	Session Initiation Request
SMS	Short Messaging Service
TCP	Transmission Control Protocol
UI	User Interface
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
VM	Virtual Machine
WAP	Wireless Application Protocol
WBXML	WAP Binary XML
WSP	Wireless Session Protocol
XML	Extensible Markup Language

Foreword

This thesis was written in winter and spring 2003 in the Software Technology Laboratory of Nokia Research Center, Tampere. This paper would never have appeared or been completed without help, advice and support of many people.

First of all, I want to thank Prof. Jan Voracek, the examiner of the thesis, for the provided opportunity to take part in the IMPIT program, his energy, optimism and constant readiness to help in solving various academic and practical problems.

I express my deep gratitude to Dr. Kari Systä, the supervisor of the thesis, for having invited me to Nokia Research Center. His careful guidance, valuable comments, and granted freedom in selection of research direction played a key role in success of the work. I am also grateful to Nokia Research Center, which has funded this study and provided an excellent research environment.

I am especially grateful to Ilya Baraev, a colleague of mine, for patiently answering countless questions and fruitful discussions. I thank him, and my other colleague, Juha Uola, for lots of important comments and practical help, which have definitely made this thesis better.

I also express my appreciation to Nadezhda, my cousin, for her help with language issues, which made the language of the thesis more understandable.

Finally, special thanks goes to Irina, my girl-friend, and Olga and Anatoly, my parents, for their constant encouragement, support and firm belief that one day this work will finally be accomplished. And this day has come indeed!

Once again, let me thank all of you wholeheartedly!

Tampere, May 2003

Alexander Davydov

1. Introduction

Mobile communicational devices have been evolving quickly. First mobile phones had small text-mode displays and were capable of voice communication only. However, they were already equipped with large color screens and digital cameras after several years of rapid progress, and became powerful enough to run sophisticated software. Furthermore, a whole new class of mobile devices has emerged: communicators, devices that combine functionality of the mobile phone and the PDA. On the other hand, PDAs have got some features inherent to mobile phones, i.e. voice communication. This quick development has resulted in a growing demand for various applications for these mobile devices and generated interest in the third-party software development.

A possibility to facilitate third-party application development for mobile devices is provided by Java™ technology. The Java™ 2 Platform, Micro Edition (J2ME) is specifically tailored for such resource-constrained devices as mobile phones, communicators and PDAs. The platform is implemented on top of the existing operating system and allows creation of portable applications. The J2ME platform has a flexible structure, consisting of configurations, profiles and optional packages. The profile intended for such devices as mobile phones is called the Mobile Information Device Profile (MIDP). Java applications created for this profile are called MIDlets. The number of devices supporting the MIDP is quickly growing.

One of the main benefits of the J2ME platform is dynamic delivery of applications. This process, which is also referred to as provisioning, allows the user of the mobile device to install new applications as needed. The delivery process is crucial for success of the platform, therefore it is very important to make it easy, quick, reliable and standard. The goal of this thesis is to study the existing methods for provisioning of MIDlets and propose several new ones. Most of the proposed methods are for local provisioning via the Bluetooth™ wireless technology.

Another problem discussed in the thesis is protection of MIDlet providers' copyright. MIDlets, like any computer programs, can be copied illegally. A technology that can be used in mobile devices to prevent illegal copying is the Open Mobile Alliance™ Digital Rights Management (OMA™ DRM). The thesis gives an overview of the technology and makes several proposals on how to incorporate it into various provisioning methods.

Finally, the thesis explores forwarding of MIDlets from one mobile device to another. Until recently, there was no standard way to perform this operation. However, the OMA DRM superdistribution delivery method facilitates forwarding of MIDlets without infringement of their providers' copyright. Two methods for superdistribution of MIDlets are proposed.

The thesis is organized as follows. Chapter 2 gives a general overview of Java technology and then concentrates on the J2ME platform. The special emphasis is made on the Mobile Information Device Profile and Java applications for this profile, MIDlets.

Chapter 3 considers existing technologies for provisioning of MIDlets. Over-The-Air provisioning, as the most important provisioning technology is explained in every detail. Other described techniques are WAP Push provisioning, provisioning using PC connectivity software, provisioning via Bluetooth and Infrared links and provisioning via e-mail and the MMS.

Chapter 4 reviews the OMA DRM. All delivery methods are explained, viz. forward-lock, combined delivery, separate delivery and superdistribution. Since the OMA DRM is a general-purpose technology intended for any type of the content, some clarification regarding its usage for provisioning of MIDlets is required. The Chapter suggests how

OMA DRM combined and separate delivery methods can be integrated into Over-The-Air provisioning of MIDlets. Superdistribution of MIDlets is also discussed here.

Chapter 5 provides an overview of the Bluetooth wireless technology in application to local provisioning of MIDlets. Special attention is paid to Bluetooth device and service discovery and those Bluetooth profiles that are intended for object transfer, viz. the Generic Object Exchange Profile, the Object Push Profile and the File Transfer Profile.

Chapter 6 discusses local provisioning from autonomous devices such as automatic MIDlet dispensers, electric appliances, etc. At first, some use cases where such type of local provisioning may be useful are presented. Then, several new methods are proposed, all but one being based on existing Bluetooth profiles, viz. the File Transfer Profile, the Private Area Networking Profile and the Object Push Profile.

Chapter 7 explains how the OMA DRM facilitates forwarding of MIDlets from one mobile device to another. Two methods are proposed. The local one is based on the Bluetooth Object Push Profile, while the other uses e-mail or the MMS.

2. Java™ Technology in Mobile Devices

The Chapter gives an overview of the Java™ 2 Platform, Micro Edition. This Java platform is intended for small embedded and mobile devices such as mobile phones, communicators and PDAs, etc. Small size and modular architecture allows it to cover resource-constrained devices with very different capabilities and functions. The platform allows creation of highly portable Java applications for a wide range of mobile devices by different manufacturers.

2.1 Overview of Java™ Technology

Java™ technology was rolled out by Sun Microsystems in May 1995 in attempt to create a software platform that will allow the same code to be executed on any computer. The innovation was warmly welcomed by the industry, as there was a clear need for some universal platform that will span across different computing devices, especially in the light of quick development of the Internet. Soon Java technology was incorporated in all major Web browsers, and users of the Internet started to enjoy first Java Web applications (applets). However, the application domain of the technology was not limited to simple applets running in a browser - later, Java technology was introduced into most operating systems and it became possible to create complex cross-platform networked applications.

The main idea of the technology is to have a virtual processor, the so-called Java virtual machine (JVM) implemented for all devices that support the Java platform. This virtual machine executes programs written in the Java language. As any implementation of JVM provides the same standard functionality, Java applications can be executed on any device with a JVM. The price paid for making Java applications cross-platform is a computational overhead caused by translation of Java language instructions into native instructions performed by a JVM. This means that the Java code is naturally slower than the native code.

In a few years, the platform has occupied a strong position in the market with increasingly more developers selecting it to create their applications. Moreover, Java technology has spread to new application domains, and it became obvious that a single Java platform cannot cover all variety of smart devices. E.g., a server and a mobile phone serve different purposes and their computational resources vary greatly. Both devices can support the Java language, but JVMs and sets of Java libraries have to be different.

To address this need for segmentation, Sun Microsystems announced (in 1999) and released three editions of the Java 2 platform: Enterprise Edition, Standard Edition and Micro Edition.

The Java™ 2 Platform, Standard Edition (J2SE™) is a direct derivative of the original Java platform, which was released in the early 1996. This platform is intended for workstations, laptops and home computers; it serves as a base for client-side Java applications as well as simple server-side applications. The J2SE is available for most of the widespread operating systems: Microsoft® Windows®, Linux, Solaris™ Operating Environment, Mac OS® X, etc. Further information concerning the J2SE platform can be found in [1].

The Java™ 2 Platform, Enterprise Edition (J2EE™) was created to form a basis for enterprise solutions. J2EE is J2SE plus a whole bundle of technologies and standards intended for development of complex, multi-tier and scalable server-side applications.

These technologies include JavaServer Pages™ (JSPs™), Java Servlets, Enterprise JavaBeans™ (EJBs™), etc. Further information about the J2EE platform can be found in [2].

The Java™ 2 Platform, Micro Edition (J2ME™) addresses devices with limited resources, not capable of supporting the whole J2SE platform. Mobile phones, communicators, PDAs, TV set-top boxes, in-car systems, various embedded devices fall into this category. By supporting the compact J2ME platform, these devices will be able to execute Java applications, but a set of libraries will be more restricted than in the J2SE. Since this platform is the focus of this thesis, it is described in details in Section 2.2.

Figure 1 depicts the three editions along with the types of devices they are intended for.

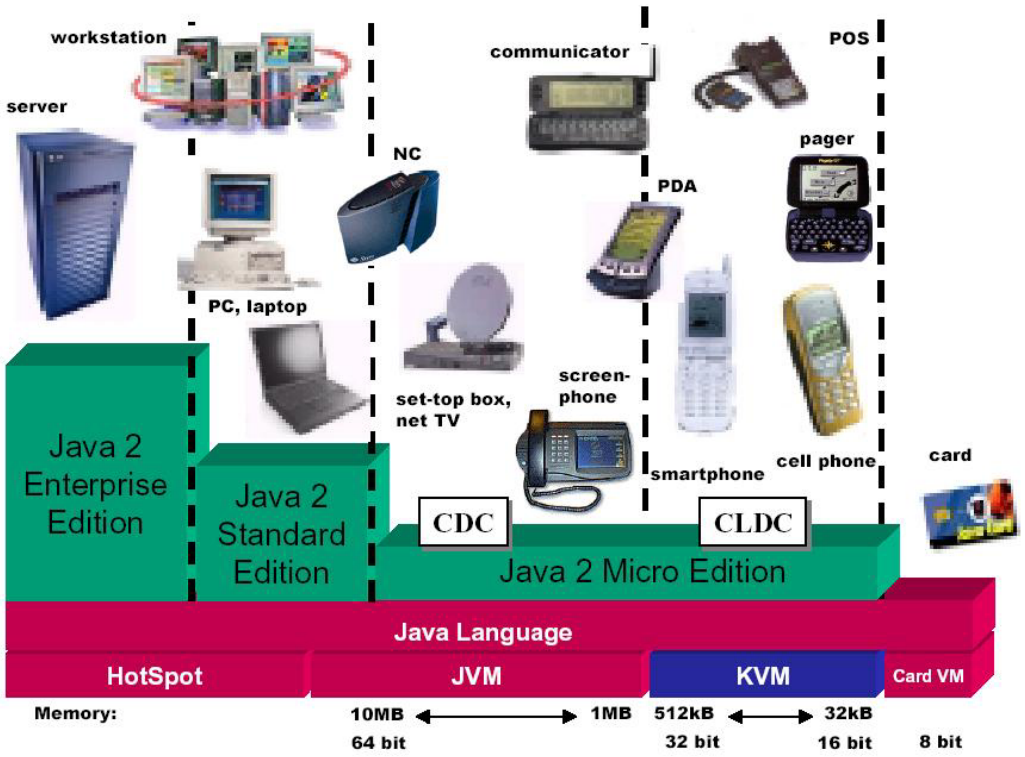


Figure 1. Three editions of the Java™ 2 Platform with their target devices. Source: [3]

In addition to the three platforms, the Java Card™ virtual machine can be seen in Figure 1. Java Card is a technology that enables smart cards (like cellular phone SIM cards, banking cards) and other devices with very limited memory to run small and simple Java applications. The technology uses the smallest virtual machine in the Java world and serves as an addition to the J2ME platform, thus allowing creation of all-Java software solutions. Further information about Java Card™ technology can be found in [4].

2.2 Java™2 Platform, Micro Edition

2.2.1 Java™ Technology in Mobile Phones

Why does the mobile world need Java™ technology? When the Java platform was unveiled in 1995, mobile phones were regarded only as devices for voice communication.

They had text-mode displays and very limited computational resources. However, a few years of rapid progress changed the situation completely – phones received large graphical displays and were capable of running software that is much more sophisticated. Moreover, a new class of mobile phones emerged, the so-called communicators, i.e. devices that combine functionality of the mobile phone and the PDA. Communicators have even larger screens and much more memory than mobile phones. Cell phone manufacturers become interested in third-party software development for their devices, several options being available to facilitate this process.

One way is to enable third-party application development for existing phone operating systems. Such approach has several disadvantages. First, opening a phone operating system for third-party software development makes it vulnerable for destructive applications (viruses). Second, it can damage competitiveness of the company as some know-how can leak to competitors. Third, it can be difficult to attract third-party developers to a usually complex low-level operating system. Finally, such approach does not suit minor manufacturers, as majority of third-party software vendors will most likely ignore their operating systems, preferring to deal only with major manufacturers.

The other way to empower third-party application development is to use some open operating system, like Symbian OS™. But adoption of a new operating system requires significant efforts. What is even more important, Symbian OS and other open systems available still require more computational resources than mass-market phones usually have.

Another option is to implement a Java virtual machine for an existing phone operating system, or, what sometimes happens, port Sun Microsystems reference implementation and use Java platform as a platform for third-party development. Benefits of such approach include lots of experienced Java programmers, reliable “sandbox” security model and simplicity (compared with C++ or C) of the Java language. That is why the Java platform was selected by majority of manufacturers as a platform for third-party applications.

To meet industry requirements, the standards of the new platform were created through the Java Community Process (JCP). Further details about the JCP can be found on the JCP Web site [5]. All major manufacturers of potential target devices took part in making specifications. The standardization work was led by Sun Microsystems, mobile phone manufacturers being key contributors. As a result, the Java 2 Platform, Micro Edition (released in the middle of 2000) was warmly welcomed and immediately became a de-facto standard Java platform for the mobile phone industry.

2.2.2 Platform Architecture

Section 2.1 showed that the Java™ 2 Platform, Micro Edition is a Java platform for devices with constrained resources. However, there are so many various types of such devices and their functions are so different that the platform has to be very flexible to cover them all. This flexibility is achieved through a rather complex internal structure. Figure 2 shows that the J2ME platform is comprised of major building blocks, each having a layered structure. Each of these blocks consists of configuration, profiles and optional packages and defines a complete Java runtime environment for some target category of devices.

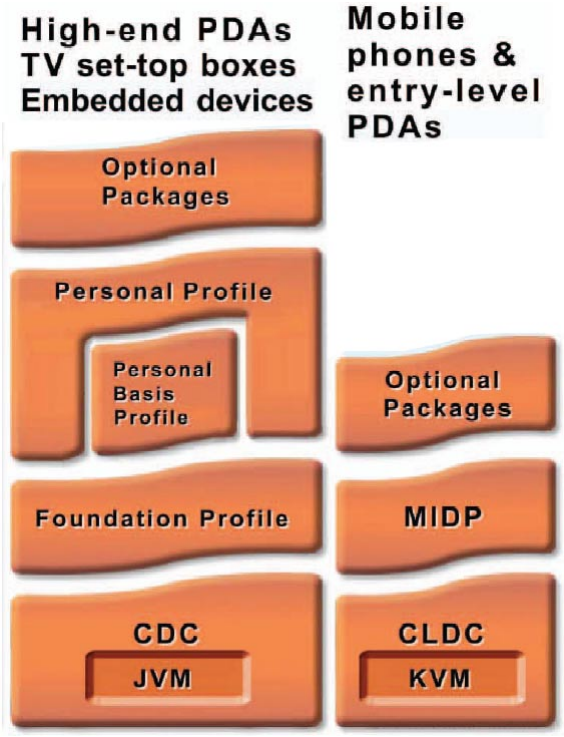


Figure 2. J2ME™ platform architecture. Source: [6]

2.2.3 Configurations

According to [6], a configuration is a Java virtual machine plus a minimal set of class libraries. Each configuration is a basis for a Java platform intended for a group of devices with similar resources. A configuration defines a minimum level of Java technology supported, but does not contain any high level APIs (like UI, Networking, etc.) Two configurations are defined for the J2ME platform: the Connected, Limited Device Configuration (CLDC), and the Connected Device Configuration (CDC). Their names show that both configurations are intended for devices somehow connected to a network. More configurations can be defined through the JCP as needed.

CDC is a less restricting configuration designed for such devices as TV set-top boxes, high-end PDA's and in-car systems. The CDC virtual machine supports the same features as the J2SE virtual machine. Target devices should have 32-bit processors, at least 2Mb of memory, and network connectivity.

CLDC is a more restrictive configuration intended for such devices as mobile phones, PDAs and two-way pagers. These devices have slower processors and less memory as compared to the devices of the CDC target group. Network connection can be inconstant and slow. Requirements for the devices include a 16-bit or 32-bit processor, 128-512 kB of memory, and a network connection. A more detailed description of CLDC can be found in Section 2.3 of the thesis.

2.2.4 Profiles

Configuration alone does not provide a complete Java runtime environment. It has to be supplemented with one or more profiles. Profiles extend configuration by adding a set

of higher-level APIs that provide functionality required for some type of devices, e.g. mobile phones. Thus, computational resources of a device determine the supported configuration, while profiles are selected according to device functions. A device can support several profiles, if they are based on the same configuration. A profile can be based not only on the configuration, but on other profiles as well. E.g., the Personal Information Device Profile (PDAP) uses the Mobile Information Device Profile (MIDP), and both of them are based on CLDC (cf. Figure 3). All the three establish a complete Java runtime environment for a PDA class of devices.

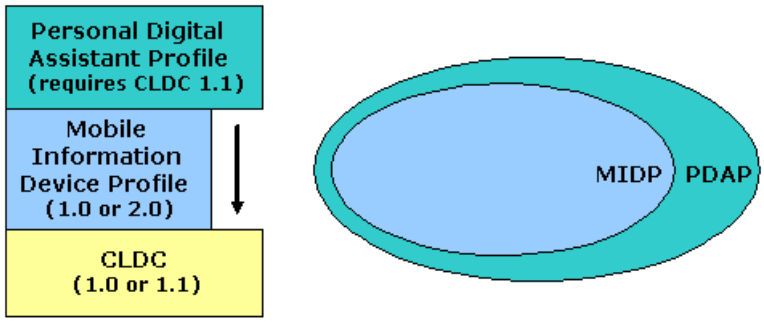


Figure 3. PDA Profile on top of the Mobile Information Device Profile. Source: [7]

Similar to configurations, J2ME profiles are defined through the Java Community Process. Here are some of the CDC-based profiles: the Foundation Profile (FP), the Personal Profile (PP), the Personal Basic Profile (PBP). CLDC-based profiles include the Mobile Information Device Profile (MIDP) version 1.0 and 2.0 and the Personal Digital Assistant Profile (PDAP). MIDP 1.0 and 2.0 are relevant to this thesis; their detailed description is provided in Sections 2.4 and 2.5 respectively.

2.2.5 Optional Packages

An optional package is a profile extension that offers a standard API for some specific technology. Some optional packages can be added to one profile only; others can be used with several profiles. Optional packages are modular, so the device manufacturer can add them as needed to adjust the Java platform to fit the purpose and functionality of the device [6]. Here are some examples of optional packages: Java APIs for Bluetooth, Wireless Messaging API, RMI Optional Package, Location API.

2.3 Connected, Limited Device Configuration

Connected, Limited Device Configuration (CLDC) comprises a Java virtual machine and a set of Java libraries. Together, they define a foundation of the Java platform for a large group of devices with very limited computational power, short battery life and slow-speed network connection. That is why CLDC should be very compact and cannot support all features found in J2SE. There are two versions of CLDC specification: 1.0 (released in 2000 and found in many Java-enabled mobile phones) and 1.1 (released in 2003). CLDC 1.1 adds some features that were omitted from CLDC 1.0 to make it smaller. As a result,

CLDC 1.1 requires more computational resources and addresses less resource-constrained mobile devices, just appearing now.

2.3.1 Connected, Limited Device Configuration version 1.0

CLDC 1.0 was the first J2ME configuration implemented by Sun Microsystems and the first one that appeared in real devices. According to [3], purposes of CLDC are to:

- Define a standard Java platform for small, resource-constrained, connected devices
- Allow dynamic delivery of Java applications and content to those devices
- Enable third-party application developers to easily create applications and content that can be deployed to those devices.

The CLDC 1.0 specification describes the following areas:

- Java language and virtual machine features
- Core Java libraries
- Input/output
- Networking
- Security
- Internationalization.

Support for the Java™ Language

To meet strict memory limits of target devices, some features of the Java language are not supported. Differences from the Java Language Specification [8] are:

- No support for floating point data types (float and double)
- No support for finalization of class instances
- Limitations on error handling. Most subclasses of java.lang.Error are not supported. Errors of these types are handled in a manner appropriate for the device.

In all other respects, support of the Java language in a Java virtual machine for CLDC 1.0 should be the same as in a JVM for J2SE. No support for some language features allows a JVM for CLDC 1.0 to fit the total of 128 kB, CLDC 1.0 libraries included (according to §2.2.1 of the CLDC 1.0 specification [9]).

Java™ Virtual Machine Features

The differences between a CLDC 1.0 and J2SE Java virtual machines are listed below:

- No floating point support. Floating point data types were excluded because their processing will cause significant computational overhead without hardware support for floating point operations (which is the case for most of CLDC 1.0 target devices). Without these datatypes, a CLDC 1.0 JVM is more compact and quick
- No Java Native Interface (JNI) support. JNI was left out partially due to security model of CLDC 1.0 (set of native functions has to be closed) and partially to save memory

- No support for user-defined class loaders. This feature of J2SE JVM was removed because of security restrictions. In CLDC 1.0 JVM, the only class loader is a built-in class loader; it cannot be replaced or modified by the user
- No reflection support. As a result, RMI, object serialization, JVM Debugging Interface, JVM Profiler Interface are not supported either. Further details about reflection features can be found in [10]
- No support for thread groups and daemon threads. All operations (like starting or stopping a thread) should be performed separately for each thread
- Finalization is not supported
- No support for weak references. Weak references are a mechanism in J2SE that allows Java application to store a reference to the object that is already waiting to be garbage-collected
- Error handling capabilities are limited. Most of J2SE error classes are not supported.

Before running a Java application, any JVM must ensure that the application adhere to Java language safety rules by verifying its classfiles. Invalid classfiles must be rejected. A CLDC 1.0 JVM also follows this rule, but as the standard verification mechanism (described in Java Virtual Machine Specification [11]) is too demanding for resource-constrained devices, a new verification method is defined for a CLDC 1.0 JVM. Verification process is split in two phases: pre-verification, performed outside the device, and on-device verification. Most of verification job is done off-device, which allows speeding up the process and saving device memory.

Security Model

An important feature of a CLDC 1.0 based Java platform is the ability to download Java applications dynamically. This requires a carefully designed security model, as the downloaded applications can potentially be unsafe. The J2SE security model is too complex and resource-demanding for mobile devices, which is why new security guidelines were defined. Here are the most important CLDC 1.0 security rules (according to [9]):

- Java applications executed in the virtual machine must not be able to harm the device in which they are running. This is achieved through verification of classfiles. Classfiles that contain references to invalid memory locations (outside Java heap) are rejected
- Java application must run in a closed “sandbox” environment, accessing only the strictly defined set of APIs.
- Classes in system packages cannot be overridden by a Java application
- The set of native functions is closed, and no new native code can be added
- CLDC 1.0-based profiles can provide additional security solutions.

2.3.2 Java™ Virtual Machines for CLDC 1.0

Initially, only one CLDC 1.0 Java virtual machine was available – the K Virtual Machine (KVM) by Sun Microsystems. Since its release in the year 2000, it was ported onto many mobile devices and is currently a de-facto standard Java virtual machine for CLDC 1.0. In the year 2002, Sun Microsystems added one more CLDC 1.0-compliant

virtual machine, the CLDC HotSpot™ Implementation Virtual Machine, which boasts gradually better performance while still having a small memory footprint required by mobile devices. There are also several CLDC 1.0-compliant virtual machines developed by other companies.

K Virtual Machine (KVM)

The KVM was created to be a Java virtual machine with support for all core features of the Java language, and, at the same time, small enough to run in devices with very limited computational resources (phones, PDAs, home appliances, etc.). According to [3], the main virtues of the KVM are:

- Small (40-80 kB, depending on target platform and compilation options) static memory footprint of the virtual machine core
- Clean and highly portable
- Modular and customizable
- As complete and fast as possible under existing resource restrictions.

As the KVM is implemented in the ANSI C programming language and is just one thread in terms of the operating system, it can relatively easily be ported onto various platforms.

Other CLDC 1.0-Compliant Virtual Machines

In addition to already mentioned Sun's VMs, the list of CLDC 1.0-compliant virtual machines include:

- CLDC VM by Insignia
- J9 VM by IBM® (part of WebSphere® Studio Device Developer)
- MicrochaiVM by Hewlett Packard
- intent® CLDC VM by Tao group.

2.3.3 CLDC 1.0 Class Libraries

CLDC 1.0 class libraries provide minimum functionality required for application development and definition of profiles, with the special emphasis on networking. According to [9], CLDC 1.0 class libraries can be divided into two groups:

- Classes that are subset of standard J2SE libraries
- Classes that are specific to CLDC 1.0.

Most of CLDC 1.0 class libraries fall into the first group, which increases upward compatibility of applications. Some class libraries were designed specifically for CLDC 1.0, due to the fact, inter alia, that I/O and networking happen in a different way in CLDC 1.0 target devices. Figure 4 illustrates relationship between J2SE, CDC and CLDC class libraries.

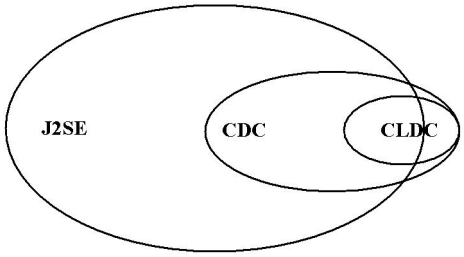


Figure 4. Relationship between J2ME™ and J2SE™ class libraries. Source: [9], Appendix 1

The most important class libraries among the CLDC 1.0-specific ones are those of Generic Connection framework (GCF). According to [9], GCF is a generalization of the J2SE network and I/O classes that allows flexible and extensible support of various protocols. The idea is to use a single set of related abstractions for different forms of communication instead of using several sets of totally different abstractions. Such approach allows addition of communication protocols as needed with minor (if any) changes on the application programming level. Six basic interface types have been defined:

- A basic serial input device
- A basic serial output device
- A datagram-oriented communications device
- A circuit oriented communications device (TCP)
- A notification mechanism for a server to be informed of client-server connections
- A basic Web server connection.

It is necessary to mention that the CLDC specification does not require implementation to support a particular protocol. It is assumed that decisions what protocols to support will be made at the profile level and profiles will add appropriate protocol implementations. However, the Sun Microsystems Reference Implementation of CLDC 1.0 includes implementations of some protocols. They are the datagram protocol (UDP), the socket connection protocol and the file access protocol. These protocols are meant to exemplify how protocols are implemented in the GCF.

2.3.4 Connected Limited Device Configuration Version 1.1

The CLDC 1.0 specification was recognized as successful; therefore, a new version 1.1 (released in 2003) does not contain major changes and is mainly an incremental release. It is fully backward compatible with CLDC 1.0, but includes some enhancements and improvements. Here are the most prominent of them according to [12]:

- The floating point support has been added. Both float and double types are now supported. Various methods were added to numerous library classes to deal with floating point values
- The weak reference support has been added
- Thread objects have names (like in J2SE)
- The classes of Calendar, Date and TimeZone have been redesigned to be more J2SE-compliant

- Error handling requirements have been clarified
- A lot of minor changes were introduced to class libraries.

As a result of support for floating point types, CLDC 1.1 requires 32 kB of memory more than CLDC 1.0. To be more exact, new CLDC specification assumes that:

- At least 160 kB of non-volatile memory is available on a device for CLDC 1.1 libraries and virtual machine (CLDC 1.0 fits in 128 kB)
- At least 32 kB of volatile memory is available on a device for the JVM runtime (the same as for CLDC 1.0).

Formally, requirements for the processor remain the same (16-bit or 32-bit processor). Nevertheless, a more powerful processor or a math co-processor is certainly required to maintain performance at the adequate level while processing numbers with floating point.

In general, CLDC 1.1 provides richer means for profile definition and application programming than CLDC 1.0 (mainly because of floating point support). At the same time, it requires more computation resources. Evolution of CLDC specification reflects growing computational power of target devices (primarily mobile phones).

2.4 Mobile Information Device Profile Version 1.0

The Mobile Information Device Profile 1.0 (MIDP 1.0) was the first profile defined for J2ME™. Currently, it is supported by many Java-enabled mobile phones. The MIDP 1.0 is a CLDC-based profile; it is usually used in conjunction with CLDC 1.0, but can be placed on top of CLDC 1.1 as well (as CLDC 1.1 is backward compatible with CLDC 1.0). Together with CLDC and, possibly, additional packages, the MIDP defines a complete Java runtime environment for small handheld devices like mobile phones. Figure 5 shows a typical Java architecture found on a device with the MIDP support.

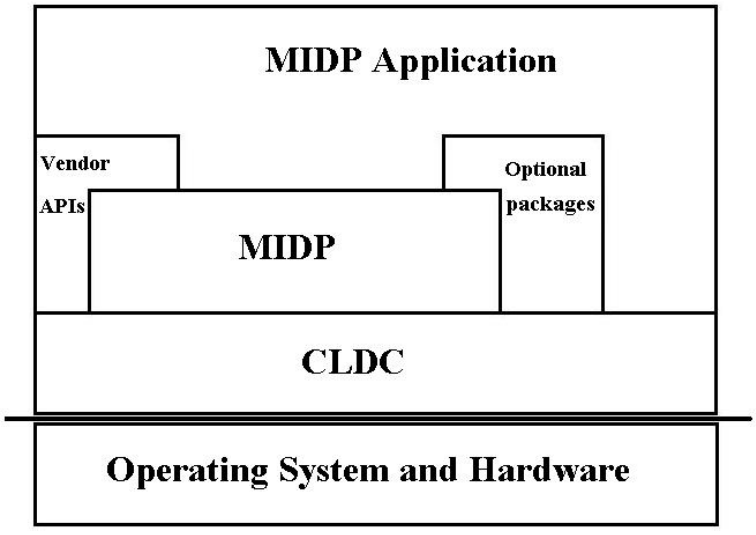


Figure 5. Typical Java runtime environment in a MIDP-compliant device

In addition to requirements set by CLDC, the MIDP 1.0 requires of a device to have the following minimum characteristics:

- A monochrome display of 96 x 54 pixels
- One or more of the following user-input mechanisms: a phone keypad, a usual keyboard or a touch screen
- 128 kB of non-volatile memory for MIDP libraries
- 8 kB of non-volatile memory for application-created persistent data
- 32 kB of volatile memory for the Java runtime
- Two-way network connection, with limited bandwidth, possibly intermittent.

The MIDP 1.0 complements CLDC by providing APIs in the following areas (according to [13]):

- Application, by defining how a MIDP application looks like, how it is controlled, etc.
- User interface (display and input)
- Persistent storage
- Networking
- Timers.

Application

Java applications written for the MIDP are called MIDlets. MIDlets are in a way similar to Java applets. Similar to an applet, a MIDlet has methods that are called by system in order to start, stop and destroy the application. MIDlets are described in more details in Section 2.6 of this thesis.

User Interface

The MIDP UI consists of two APIs: the high level and the low-level ones. The high level API is meant for business applications and provides portability across different devices. To achieve this portability, the API uses high level of abstraction – an application can only define what components (like lists, check-boxes, etc.) it wants on the screen and what kind of user interaction is possible. An application does not have any control either on how these components are drawn on the screen or how the actual user interaction happens (an application cannot access any individual keys). Drawing of components and user input is implementation-dependent and is assumed to be done according to the device UI style (hardware and native).

The low-level API, on the contrary, is intended for applications that need full control over graphics and look-and-feel, as well as access to low-level input events. Example of such application is a game. Applications that use the low-level API are less portable (as they depend on screen size, on presence of certain device-specific keys, etc.). To remain portable, such applications should follow certain guidelines (described in [13], Section 9.2). In other words, some efforts are required at the application programming level.

Persistent Storage

The MIDP 1.0 defines a mechanism that allows MIDlets to store data permanently and to access it later. This mechanism is called the Record Management System (RMS).

Information is stored in record stores; each record in a record store is a byte array. A MIDlet can create one or more record stores, which will remain intact throughout entire stay of the MIDlet on a device (including reboots, battery changes, etc.). When a MIDlet is removed, its record stores are removed as well.

Networking

The MIDP 1.0 adds support for a subset of the HTTP 1.1 protocol to the Generic Connection Framework defined in CLDC (cf. Section 2.3.3 of this thesis). The supported subset includes HEAD, POST and GET requests. The connection is opened using a URL that starts with "http".

The HTTP support can be implemented using either protocols from the IP family (e.g., TCP/IP) or other protocols (e.g., WAP, i-mode). However, implementation details should be hidden from a MIDlet that uses an HTTP connection.

Timers

The MIDP 1.0 adds functionality that allows an application to schedule tasks for future execution in a background thread. Tasks can be scheduled for one-time execution or for repeated execution at specified intervals. Class libraries for support of these features are inherited from J2SE.

While addressing the above-mentioned areas, the MIDP 1.0 does not specify how applications are installed, run and removed. The specification only mentions that it is done in a device-specific manner by the so-called *Application Management Software* (AMS). It is assumed that all MIDP-compliant devices would support dynamic application download, but the actual mechanism is out of the scope of the MIDP 1.0 specification because of variety of possible methods. However, as Over-The-Air provisioning was believed to be the main provisioning method, a recommended practice for the OTA download of MIDP applications was described in the additional document [14].

2.5 Mobile Information Device Profile Version 2.0

The MIDP 2.0, released in autumn 2002, extends the MIDP 1.0 in several areas by adding multiple important features discussed below in this Section. At the same time, the MIDP 2.0 remains fully backward compatible with the MIDP 1.0, i.e. MIDlets created for the MIDP 1.0 will run in the MIDP 2.0 environment without any changes. The new profile can work on top of CLDC 1.0 or CLDC 1.1, although it is assumed that most MIDP 2.0 implementations will be based on CLDC 1.1 [15].

Requirements for devices found in the MIDP 2.0 specification are almost the same as for MIDP 1.0. The differences are:

- MIDP 2.0 requires 256 kB of non-volatile memory for MIDP implementation (128 kB for MIDP 1.0)
- MIDP 2.0 requires 128 kB of volatile memory for the Java runtime (32 kB for MIDP 1.0)
- The target device should be able to play tones (MIDP 1.0 does not have this requirement at all).

The new MIDP, especially in conjunction with CLDC 1.1, makes greater demands of computational resources, which reflects growing capabilities of target devices, first of all, of mobile phones. In exchange for consumed additional resources, the MIDP 2.0 offers an extensive set of new features. Here are the most prominent of them (according to [16]):

- Secure Networking (using HTTPS)
- Multimedia
- Enhanced User Interface
- Special API for game programming
- RGB images (image representation as an array)
- New framework for MIDlet authentication (only authenticated MIDlets can access sensitive APIs)
- Push Registry (a mechanism that allows MIDlets to be launched in response to incoming network connections)
- Changes in persistent storage.

Secure Networking

The MIDP 2.0 includes additional interfaces for secure interaction with WWW network services. These secure interfaces are provided by the HTTPS protocol over any appropriate bearer (e.g., IP, WAP, etc.). A secure connection is opened using a URL that starts with “https”.

Multimedia

The MIDP 2.0 includes a multimedia API, which is a subset of the Mobile Media API (JSR-135). This subset includes audio-related features only; graphics and video are not supported (they require significant computational resources, presently not available on the majority of target devices). The multimedia API of MIDP 2.0 allows generation of single tones or sequences of tones. Playback of sampled audio is also possible; audio files can be located either in the JAR file (JAR files are explained in Section 2.6) or somewhere in the WWW. The support for WAV files is mandatory, other formats are optional.

Enhanced User Interface

MIDP 2.0 UI APIs contain a lot of enhancements and new features compared with the MIDP 1.0. The ultimate goal was to make the user interface richer and more flexible. E.g., now it is possible for a MIDlet to define its own UI Items. Other new features include a more sophisticated layout of components and an extended command handling mechanism.

Special API for Game Programming

The MIDP 2.0 includes the Game API that allows development of rich gaming content. The main concept of the API is that the screen is composed of several layers. Each of them can be manipulated and rendered separately. This API makes creation of animation easier.

RGB Images

Another important feature introduced is representation of images as integer arrays. Each integer in an array represents a pixel, with 8 bits for each opacity, red, green and blue values. This allows MIDlets to manipulate image data directly.

New Framework for MIDlet Authentication

To protect sensitive and restricted APIs, the MIDP 2.0 introduces the concept of trusted MIDlets. A trusted MIDlet is a MIDlet that was able to authenticate itself in one of protection domains available on the device. Each protection domain defines which APIs and how (silently or with user permission) can be used by authenticated MIDlets. If a MIDlet cannot be authenticated, then it runs as untrusted and a set of available APIs is the the most restricting. The mechanism for authentication of MIDlets is based on the X.509 Public Key Infrastructure. In the MIDP 2.0 environment, all MIDP 1.0 MIDlets are run as untrusted.

Push Registry

The Push Registry is a mechanism that allows a MIDlet to register for network connection events, which may happen when the MIDlet is not running. If the MIDlet has registered for some connection, then it is launched when this incoming connection occurs. The registration can happen both during installation (static push registration) and runtime (dynamic push registration). A MIDlet can also register for auto-launch at a specific time.

Changes in Persistent Storage

Records in a record store can now be accesses from the outside of a MIDlet suite (MIDlet suites are described in Section 2.6) that created this record store. In the MIDP 1.0, only MIDlets from the same MIDlet suite can share record stores.

Figure 6 gives a high-level view of the MIDP 2.0. The Over-The-Air provisioning specification is now a part of the MIDP specification (in the MIDP 1.0, it was only a recommended practice). Over-The-Air provisioning of MIDlets is described in details in Section 3.2 of this thesis.

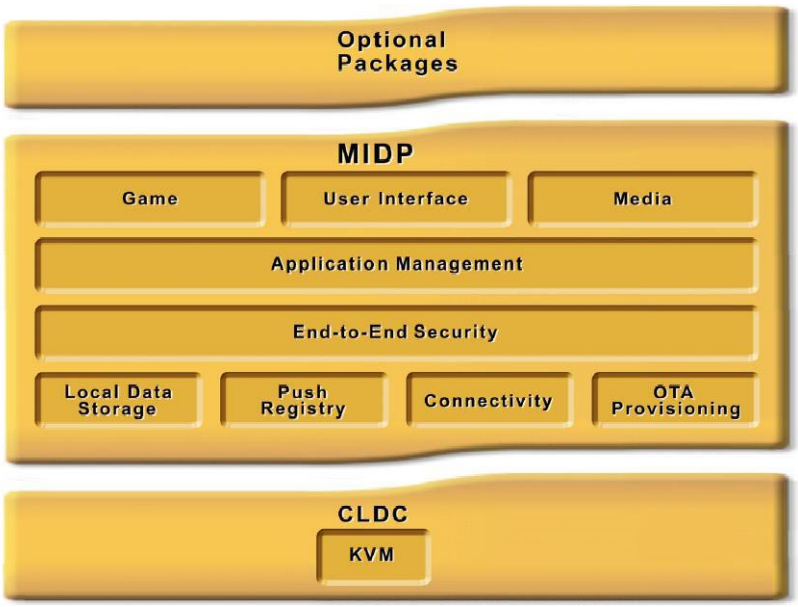


Figure 6. Overview of the MIDP 2.0. Source: [17]

2.6 MIDlet (Java Application for MIDP)

The MIDlet is a Java application written for a device with the MIDP onboard. A strict definition of the MIDlet found in [15] is a bit different: The MIDlet is a Java application that uses only MIDP and CLDC APIs. Nevertheless, this thesis uses the term MIDlet to refer to any Java application that uses the MIDP, CLDC and, possibly, any other Java APIs available on the device (as illustrated in Figure 5).

According to the MIDP specification [15], every MIDP-compliant device contains special software, the so-called *Application Management Software* (AMS). It provides an environment in which the MIDlet is installed, started, stopped and uninstalled. Once the MIDlet is installed, it remains on the device until the user uninstalls it.

A typical MIDlet consists of one or several Java classes and, possibly, resource files (images, data, etc.). All these files are packaged in a single Java Archive (JAR file). Each JAR file also includes a manifest file that contains information about the content of the archive. The AMS uses attributes found in the manifest to identify, install and launch MIDlets.

A JAR file is often referred to as a MIDlet suite, as it may contain several MIDlets. All MIDlets from the same MIDlet suite can share classes and other resource files from their JAR file. Each MIDlet suite is in its own sandbox, so a MIDlet from one MIDlet suite cannot access any classes or resources that belong to another MIDlet suite (the exception is MIDP 2.0 record stores, cf. Section 2.5 “Changes in Persistent Storage”).

A JAR file may be accompanied by an Application Descriptor (JAD file). A JAD file allows the AMS to reject MIDlet suites that, for some reasons, cannot be installed or executed on the device without downloading the JAR file (which sometimes can be long and expensive). A MIDlet suite can be rejected because there is not enough memory on the device to store the JAR file, or because the suite requires a newer MIDP version than is currently available on the device, or for some other reasons. The MIDP specification describes the use of an Application Descriptor as optional, but recommended, as it often allows saving the user’s time and money. Further information about JAD and JAR files can be found in Section 3.1 of this thesis.

An example of MIDlet is given in Figure 7. It represents the author’s Reversi MIDlet running in the emulator. Software emulators of mobile devices are widely used to develop and test MIDlets. Such approach allows transfer of almost the whole development process to the PC.

Figure 7. Reversi MIDlet in the emulator of Nokia 7210 phone (Nokia 7210 MIDP SDK v1.0)

3. Provisioning of MIDlets – State of Art

From the very early stages of development, the dynamic application deployment was considered as the key feature and benefit of the Java™ 2 Platform, Micro Edition. Indeed, the main reason of supporting the J2ME platform on mobile devices is to enable third-party software development. This goal can hardly be achieved without dynamic deployment of applications. In case of no support for this feature, a Java-enabled device (a mobile phone, a pager, etc.) would have a pre-defined set of Java applications, which returns us to the times when mobile terminals came with a hard-coded set of software that could not be modified by the user. In such a scenario, benefits of Java technology are reduced almost to zero.

It is dynamic delivery of Java applications that makes it possible to build extensible and highly customizable mobile devices and encourages third-party software development. The user is free to select which applications and by what vendors to install. E.g., a teenager can fill his/her mobile phone with games, and a business person would benefit from such applications as stock price monitor and mobile safe, where sensitive information can be stored in the encrypted form. The majority of applications come from third-party developers, which significantly increases their diversity. Thus, dynamic application delivery is of vital importance for the J2ME platform.

The process of dynamic application delivery for the J2ME platform is often referred to as provisioning. According to [18], the term provisioning in a strict sense of the word means:

> *The act of supplying telecommunication service to a user, including all associated transmission, wiring, and equipment.*

This thesis uses the term provisioning of a MIDlet to refer to a set of activities related to discovery, downloading and installation of a MIDlet on a mobile device. Provisioning of MIDlets can take very different forms, ranging from Over-The-Air provisioning when a MIDlet is downloaded from the Internet, to local provisioning when, e.g., a MIDlet is sent from a PC via a serial cable. Different provisioning methods that are currently in use are described in Sections 2-6 of this Chapter.

Provisioning of MIDlets is a potentially complex process as it involves different types of mobile devices, MIDlets with different properties (size, required MIDP version, required optional packages, etc.), different types of protocols (HTTP, OBEX, etc.) over different types of data bearers (CSD, GPRS, Infrared, Bluetooth, MMS, e-mail, etc.). In addition, provisioning of MIDlets should happen with respect to digital copyrights of their providers. For such a complex and important process, the need for standardization is clear, and some steps in this direction have already been taken. Thus, e.g., Over-The-Air provisioning as the most important deployment method is described in the MIDP 2.0 specification; the Digital Rights Management specification [19] issue by Open Mobile Alliance™, addresses copyright questions; manufacturers of mobile devices have their internal specifications for local deployment from a PC, etc. At the same time, many important areas have not been covered by specifications yet, e.g. local provisioning from autonomous MIDlet dispensers and home appliances, forwarding of MIDlets from one mobile device to another, etc. The author's proposals regarding new provisioning technologies can be found in Chapters 6 and 7 of this thesis. This Chapter gives an overview of existing methods.

3.1 Role of JAD and JAR Files in Provisioning

Section 2.6 has already stated that a MIDlet suite is a Java Archive (JAR file) that contains one or more MIDlets. It is not possible to transfer or install individual MIDlets, so a unit of provisioning is a MIDlet suite. Thus, when provisioning of a MIDlet is discussed in this thesis, it is implied that an appropriate MIDlet suite (containing the named MIDlet) is provisioned. Along with MIDlets, each JAR file also includes a manifest file that describes the contents of the archive. A JAR manifest consists of attributes, which can be standard and application-specific. Standard ones are used by the Application Management Software (AMS) to identify, retrieve, install, and invoke MIDlets. Table 1 contains such pre-defined attributes along with their description.

Attribute Name	Attribute Description
MIDlet-Name	The name of the MIDlet suite. The AMS uses this name when displaying the suite to the user
MIDlet-Version	The version of the MIDlet suite. The AMS uses this attribute when installing and upgrading the suite and also for communication with the user
MIDlet-Vendor	The organization that provides the suite
MIDlet-Descriptor	The description of the suite
MIDlet-<n>	Each MIDlet in the suite is described by such attribute. It contains the name (shown to the user), the icon and the class of the MIDlet
MIDlet-Jar-URL	The URL from which the JAR file can be loaded
MIDlet-Jar-Size	The size of the JAR file in bytes
MIDlet-Data-Size	The size of persistent storage required by the suite
MicroEdition-Profile	The J2ME profiles required by the suite, e.g. “MIDP-1.0”
MicroEdition-Configuration	The J2ME configuration required by the suite, e.g. “CLDC-1.1”
MIDlet-Permissions	Permissions (to access sensitive API’s) that are critical for functioning of the MIDlet suite. Cf. “New framework for MIDlet authentication” in Section 2.5
MIDlet-Permissions-Opt	Permissions that are not critical for functioning of the MIDlet suite
MIDlet-Push-<n>	This attribute is used to register MIDlets in the suite to handle inbound connections. Cf. “Push Registry” in Section 2.5
MIDlet-Install-Notify	The URL to which an installation status report is sent. Cf. Section 3.2.7

Table 1. Pre-defined attributes of a MIDlet suite. Source: [15]

Attribute Name	Attribute Description
MIDlet-Delete-Notify	The URL to which a deletion status report is sent. Cf. Section 3.2.7
MIDlet-Delete-Confirm	A message that is shown to the user in the confirmation query when the MIDlet suite is removed from the device.

Table 1 (continued). Pre-defined attributes of a MIDlet suite. Source: [15]

Each JAR file may be accompanied by an Application Descriptor (a JAD file). A JAD file allows the AMS to reject MIDlet suites that are unsuitable for the device before downloading a JAR file. It also serves other purposes, e.g. the AMS can inform MIDlets about configuration-specific parameters by placing them to the JAD file instead of modifying the JAR manifest. The format of an Application Descriptor is the same as the JAR manifest format, it can contain any pre-defined attributes from Table 1 along with any application-specific attributes.

Figure 8 provides examples of the Application Descriptor and the JAR manifest. Note that both files contain application-specific attributes: *Manifest-Version* in the JAR manifest and *Nokia-MIDlet-Category* in the JAD file. There is a rule that application-specific attributes cannot start with *MIDlet-* or *MicroEdition-*.

JAR manifest

```
Manifest-Version: 1.0
MIDlet-Name: Reversi
MIDlet-Vendor: Alexander Davydov
MIDlet-Version: 1.11
MicroEdition-Profile: MIDP-1.0
MicroEdition-Configuration: CLDC-1.0
MIDlet-Description: Reversi - a simple logic game.
MIDlet-1: Reversi, /cursor1.png, reversi.Reversi
```

Application Descriptor

```
MIDlet-Name: Reversi
MIDlet-Vendor: Alexander Davydov
MIDlet-Version: 1.11
MIDlet-Jar-Size: 28966
MIDlet-Jar-URL: Reversi.jar
MIDlet-Description: Reversi - a simple logic game.
MIDlet-1: Reversi, /cursor1.png, reversi.Reversi
Nokia-MIDlet-Category: Game
```

Figure 8. JAR manifest and Application Descriptor of the Reversi MIDlet suite

The AMS uses attributes in the JAR manifest and the JAD file to install a MIDlet suite and run MIDlets from it. There are several rules about where certain attributes must appear and what value they must be. Standard attributes that must be present in any JAD file are:

- MIDlet-Name
- MIDlet-Version
- MIDlet-Vendor

- MIDlet-Jar-URL
- MIDlet-Jar-Size.

A JAR manifest also has mandatory attributes:

- MIDlet-Name
- MIDlet-Version
- MIDlet-Vendor.

The three above-mentioned attributes must be the same both in the JAD file and the JAR manifest, otherwise a MIDlet suite will not be installed. Any other attributes present in both files may be of different values, in this case the value of the JAD file will override the value of the JAR manifest. However, this rule is not valid for MIDP 2.0 trusted MIDlets. For such MIDlets, all values of duplicated attributes must be the same in both files.

There is also another rule stating that the following attributes must be present at least in one of the files:

- MIDlet-<n> for each MIDlet in the suite
- Micro-Edition-Profile
- Micro-Edition-Configuration.

If any of these rules is violated, then the AMS must refuse to install the invalid MIDlet suite.

In addition, several other checks are performed during downloading and installation of a MIDlet suite. Some of them are described in the next Section.

3.2 Over-The-Air Provisioning

Deployment of a MIDlet suite from a server to a mobile device over a wireless network is called Over-The-Air (OTA) provisioning of a MIDlet suite. OTA provisioning of MIDlet suites using HTTP is one of the main methods for delivery of MIDlets. The recommended practice for such deployment was initially described in a separate document [14]. It was published after release of the MIDP 1.0 specification; and though it is not a part of the MIDP 1.0 specification, it provides important guidelines to increase interoperability between devices by different manufacturers, networks of different cellular operators, and Web servers of MIDlet distributors. A lot of MIDP 1.0-compliant devices support this recommended practice; but many do not, inter alia because of late issue of the document. With the release of the MIDP 2.0, the recommended practice (with some changes) became a part of the MIDP specification. Now all MIDP 2.0-compliant devices must support OTA provisioning of MIDlet suites. This thesis describes OTA provisioning according to the MIDP 2.0 specification [15].

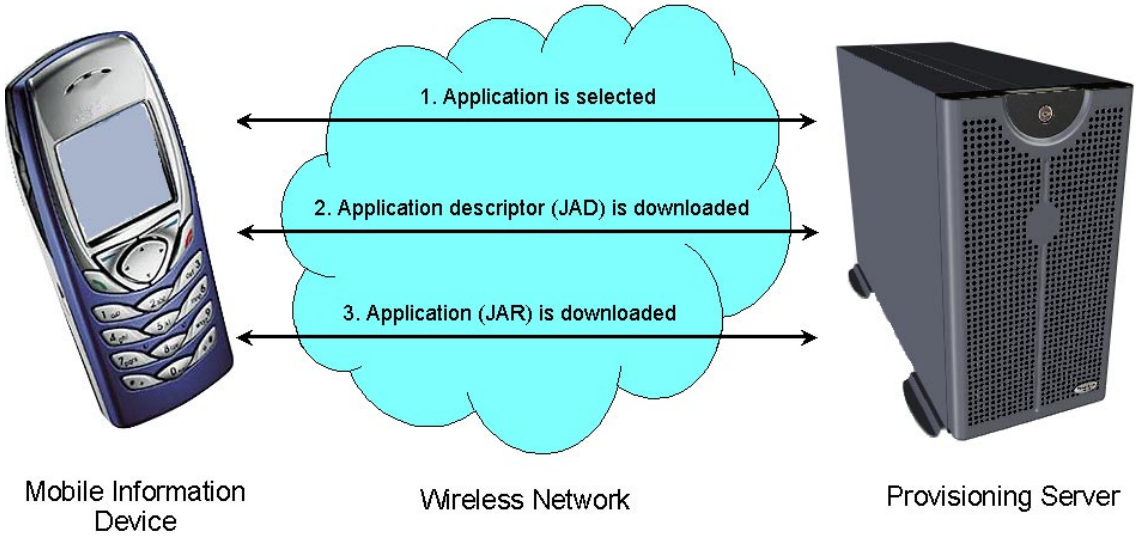


Figure 9. Simplified view of OTA provisioning

The previous Section has already mentioned that the term MIDlet suite or suite is used in this thesis to denote a unit of transfer and installation. A MIDlet suite contains one or more MIDlets. Figure 9 gives a general idea of OTA provisioning. It shows that the main participants of the process are:

Mobile Information Device has possibilities to discover a desired MIDlet suite in the network, download it using HTTP protocol and install. After that, the user can launch MIDlets from the suite, update the suite or remove it from the device. The mobile information device is often referred to as mobile device later on in this thesis.

Provisioning Server is a host in the network that contains MIDlet suites available for downloading. It also maintains a set of menus allowing clients to select a desired suite. A Web server can function as a provisioning server. Menus in this case can be created using HTML, xHTML, WML, etc.

Wireless Network is any wireless network used by the mobile information device. E.g., it can be a GSM radio network with a WAP gateway.

OTA provisioning is only one part of the MIDlet suite's lifecycle, and it is closely linked with the other parts. That is why the OTA Section of the specification does not only describe how a MIDlet suite is discovered, downloaded and installed, but also specifies how MIDlets from a suite are executed and how a suite is removed.

The subsequent Sections will describe the process of OTA provisioning, which is not that trivial in reality, in more details.

3.2.1 Requirements for an MIDP-Compliant Device

To enable OTA provisioning of MIDlet suites, a mobile device must provide a mechanism that will allow users to discover suites that can be downloaded and installed. A WAP, Web or i-mode browser can be used, or a device can contain a special application dedicated to discovery of MIDlet suites. A term *Discovery Application* or DA is used in the specification to refer to an application that is used for discovery.

Mandatory functional requirements related to OTA provisioning are listed above. A device must be able to:

- Discover MIDlet suites in the network using browsing or another mechanism
- Download MIDlet suites (JAR files) and their Application Descriptors (JAD files) from a server using HTTP 1.1 or another protocol that implements the HTTP 1.1 functionality
- Ask for a user name and password and send them to the server if the latter responds with 401 (Unauthorized) or 407 (Proxy Authentication Required). In other words, the Basic Authentication Scheme (described in [20]) must be supported
- Install MIDlet suites
- Launch MIDlets from the installed MIDlet suites
- Let the user remove the installed MIDlet. Note that as a MIDlet suite is the unit of installation, it is not possible to selectively delete some MIDlets from a suite while keeping the others. When a MIDlet suite is removed, all MIDlets in it are removed.

3.2.2 MIDlet Suite Discovery

The specification states that a device must have means to discover MIDlet suites in the network. The suite discovery takes place as follows:

1. Using the Discovery Application, the user finds and selects a link to a desired MIDlet suite (or an Application Descriptor if the suite has any). It is not usually possible to understand by appearance of the URL whether it points to a MIDlet suite (a JAR file) or to an Application Descriptor

2. If the selected link points to an Application Descriptor, then the latter is retrieved from a server by Discovery Application and presented (along with the URL it came from) to the AMS. The AMS handles all further discovery activities.

   If the URL points to a JAR file; then the discovery process ends at this point, the JAR file is retrieved and passed (again along with the URL) to the AMS, which deals with its installation. When requesting JAD and JAR files from a server, the DA should use HTTP request headers *User-Agent* and *Accept-Language* (cf. Section 3.2.8).

Section 2.6 has already mentioned that the idea behind the Application Descriptor (the JAD file) is to reject MIDlet suites that cannot or should not be installed without downloading them. A MIDlet suite can be unsuitable for a device for many reasons: memory limitations, absence of required optional packages, the suite has already been installed, etc. In some cases, presence of an Application Descriptor may help to avoid long and costly downloading. There are also some other cases when it is desirable to have a JAD file; e.g., it is needed in MIDP 2.0 security framework for authorization of trusted MIDlet suites. In general, the specification recommends to use an Application Descriptor when placing a suite for OTA downloading; though it is not a mandatory requirement.

If the selected link refers to an Application Descriptor, the discovery process goes on as follows:

1. The provisioning server indicates that the file transferred is of the *"text/vnd.sun.j2me.app-descriptor"* MIME type. This MIME type corresponds to the Application Descriptor. The received JAD file is passed to the AMS with the URL it came from.
2. The AMS uses the Application Descriptor to check if the MIDlet suite can be installed and executed. Presence of all mandatory JAD attributes is also verified. If something is wrong, the user is notified.
3. The JAD file is converted to the Unicode format to allow MIDlets from the suite to access its content during the runtime (MIDlets use the Unicode). The default encoding for the *"text/vnd.sun.j2me.app-descriptor"* MIME type is *"UTF-8"*, so conversion typically happens between these two formats. If the client supports an encoding different from *"UTF-8"*, it can specify a desired charset when requesting the JAD file, but it is not mandatory for the server to support multiple encodings.
4. Information about the MIDlet suite found in the JAD file is presented to the user. This information typically includes the suite's name, the version, the vendor and the size. Possible installation problems are also highlighted at this stage. E.g., if there is not enough memory to install the suite, the AMS can let the user delete some applications. Finally, the user confirms installation.
   The discovery process ends, and the installation of the MIDlet suite starts.

3.2.3 MIDlet Suite Installation

The installation process consists of two stages: downloading of the JAR file and making its MIDlets available for execution. If the JAD file is provided for the suite, the AMS handles both stages. If there is no JAD file available, the Discovery Application downloads the suite, and the AMS only makes its MIDlets available for launch. During installation, the user is usually informed about the progress and can cancel it, e.g., if the downloading takes too much time.

To install a MIDlet suite that has a JAD file (downloaded at the Discovery stage) the AMS performs the following actions:

1. A JAR file is requested from the URL found in the Application Descriptor (the *MIDlet-Jar-URL* attribute). Note that in this case, HTTP request headers *User-Agent* and *Accept-Language* are not used when requesting the JAR file, only the URL found in the Application Descriptor. A detailed description of possible HTTP request headers can be found in Section 3.2.8.
2. If the provisioning server responds with a request for authentication, the appropriate dialog can be shown to the user. In this case, the user enters the user's name and the password, which are posted to the server. The mobile device must support at least the Basic Authentication Scheme (as described in [20]).

The JAR file is retrieved, and after that the process is the same for MIDlet suites with or without an Application Descriptor.

3. The received JAR file is inspected to ensure that it is valid and can be installed. Possible problems that can prevent from installation include:

   - Connection is lost during installation or there is no JAR file available at the specified URL

- The JAR file is too big to be stored on the device
- The size of the JAR file is different from the one specified in the JAD file
- The JAR file is corrupted, i.e., some files cannot be extracted
- Some mandatory attributes are missing from the manifest file (cf. Section 3.1) or they do not match those found in the JAD file
- The MIDlet suite is signed, but fails to be authenticated or is not authorized to use APIs it needs (cf. "New Framework for MIDlet Authentication" in Section 2.5)
- The MIDlet suite being installed is an unsigned version of the already installed signed suite
- Static push registration fails (cf. "Push Registry" in Section 2.5).

The installation is aborted if any of these problems is encountered. In this case, the user is notified, and the appropriate error code is returned with the Installation Status Report (cf. Section 3.2.7).

4. If the JAR file has successfully passed all the checks, the suite is installed and made available for execution. The Installation Status Report is returned with the Success Status Code (cf. Section 3.2.7). The user can launch MIDlets from the suite, using device-specific selection mechanism. E.g., on some devices, MIDlets can be accessed through some menu; on the others, the AMS should be launched to select a MIDlet. In this case, MIDlets are listed in the AMS internal menus.

3.2.4 MIDlet Suite Update

Installation of the MIDlet suite that is already installed on the device (versions of the suite may differ) is considered as update. The specification states that the device must support update of MIDlet suites, but does not specify the behavior in those cases when the suite is updated to an older or the same version. In this case, behavior is device-specific. Installation of an older version may require removal of the newer one. However, it is explicitly prohibited to update a signed MIDlet with an unsigned one. In all these cases, the user has a chance to confirm or reject the update. If update is not possible (e.g., because a newer version of the suite has already been installed), the user is notified.

An updated MIDlet suite may have some RMS record stores allocated in the device's non-volatile memory (cf. Section 2.4, "Persistent Storage"). The new suite inherits these record stores in the following cases:

- The old and the new suites have the same cryptographic signer
- The JAD file of the new MIDlet suite is downloaded using the same scheme, the host and the path of the URL as the JAD file of the original suite
- The JAR file of the new MIDlet suite is downloaded using the same scheme, the host and the path of the URL as the JAR file of the original suite.

If the update scenario is different from the three listed above, the user is asked whether the new suite can inherit record stores from the original suite.

3.2.5 MIDlet Suite Execution

A device must be able to run MIDlets from installed suites, MIDlet selection mechanism is device-specific. The other device-specific feature is whether execution of several instances of the same MIDlet is allowed: some devices might support several copies of the same MIDlet running simultaneously; however, it is not usually the case.

3.2.6 MIDlet Suite Removal

The user can remove installed MIDlet suites at will. A confirmation query is usually shown before deletion. If the JAD file contains a *MIDlet-Delete-Confirm* attribute, its value is included in the query. Such mechanism allows the MIDlet provider to inform the user about consequences that may arise if the suite is removed. E.g., other installed suites from the same provider may need access to RMS record stores created by the suite being removed.

3.2.7 Status Reports

The information about success or failure of installation, upgrade or deletion of a MIDlet suite can be useful for a provisioning server. It can use it for statistics, billing or other purposes. To obtain such information, a server may include special attributes into an Application Descriptor. If *MIDlet-Install-Notify* and *MIDlet-Delete-Notify* attributes are present in a JAD file, a device tries to send installation and deletion reports using URLs specified in these attributes. However, if for some reason it is not possible to send the report (e.g., network connection is not available), the action to be reported (installation, update or deletion) is still completed. There is no specific status report for update of a MIDlet suite. A normal Installation Status Report is sent in this case.

A status report is sent using an HTTP POST request, the body including the status code and the status message. Valid values of status codes and status messages are summarized in Table 2.

Status Code	Status Message
900	Success
901	Insufficient Memory
902	User Cancelled
903	Loss of Service
904	JAR size mismatch
905	Attribute Mismatch
906	Invalid Descriptor
907	Invalid JAR
908	Incompatible Configuration or Profile
909	Application authentication failure
910	Application authorization failure
911	Push registration
912	Deletion Notification
913	Required package not supported by the device

Table 2. Install Codes and Messages. Source: [15]

In response to the status report, the server sends a *"200 OK"* message. If for some reasons there was no response from the server, or if a client did not manage to send a status report immediately (e.g., because network connection was terminated), a client may try to send it later. A retry can occur when the newly installed MIDlet suite is executed and the device has data network connectivity or, if it was a deletion status report, it can be sent together with an installation status report. However, the specification states that the number of sending attempts should be small as each of them can result in a charge to the user's bill.

3.2.8 Client Identification Using Request Headers

It will be advantageous for a provisioning server to know capability of the client device both during MIDlet suite discovery and installation. E.g., if a provisioning server knows the model of a client device already at the discovery stage, it can display only those MIDlet suites that can be installed and executed on this particular device. When the MIDlet suite was selected, this knowledge allows sending an appropriate Application Descriptor that points to the proper JAR file. In other words, OTA provisioning can be conducted in a more flexible and convenient way if clients are identified to the server.

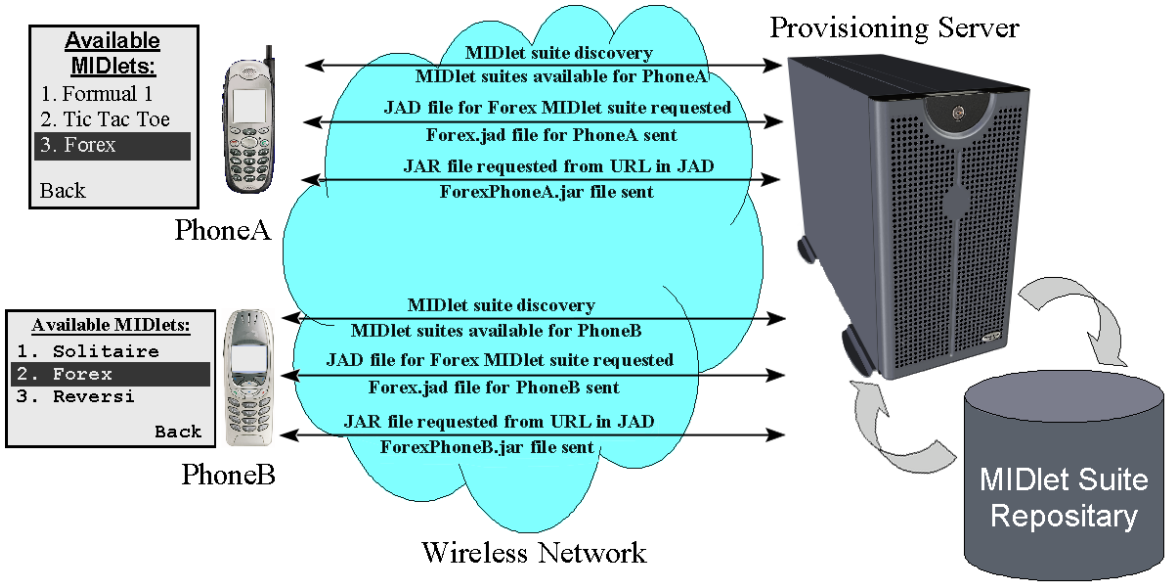


Figure 10. Provisioning server makes use of client identification

Figure 10 shows two different mobile phones browsing the same provisioning server using their Discovery Applications (e.g., a WAP or Web browser). PhoneA is a basic phone with a black-and-white display and limited memory. PhoneB is a business phone with a color screen and much more memory than in PhoneA. The provisioning server makes use of information about phones capabilities to format the menu appropriately. The user of PhoneA sees only those MIDlet suites that can be installed and executed on PhoneA, the same is for the PhoneB user. In this example, a MIDlet suite Forex is available for both phone models. If both users decide to download it, the provisioning server will send different JAD files (possibly generated "on-the-fly") to PhoneA and PhoneB, even though both phones are using the same URL to request the JAD file. Then, the phones will download Forex MIDlet suite from different URLs specified in their Application Descriptors. Each will receive an appropriate version of Forex application. PhoneA will get a lightweight version, which is able to fit into phone's memory, while PhoneB a full-featured, color version.

The specification does not define how the Discovery Application provides the provisioning server with information about the client device; it only states that this information must be provided. E.g., if the DA uses the HTTP, standard HTTP headers can be used for client identification. The same HTTP request-headers should be used when retrieving Application Descriptors and JAR files. The following headers are in use: *User-Agent*, *Accept-Language* and *Accept*.

1. *User-Agent* header. This request header is used to identify the client to the server. The format of the header is specified in the HTTP 1.1 specification as:

   "User-Agent" ":" 1(product / comment)*

   The client device should include device-specific tokens first. After that, tokens that identify the device as supporting CLDC and MIDP, are included. E.g.:

   User-Agent: Nokia7650/1.0 SymbianOS/6.1 Series60/0.9 Profile/MIDP-1.0 Configuration/CLDC-1.0

2. *Accept-Language* header. This header is used to indicate the language (languages) used on the device. E.g.:

   Accept-Language: en, fi

3. *Accept* header. This header identifies the type of content requested. This header should include *text/vnd.sun.j2me.app-descriptor* when requesting an Application Descriptor and *application/java-archive* when requesting a JAR file.

3.2.9 Example of Client-Server Interaction

This Section gives an example of possible client-server interaction during OTA provisioning of a MIDlet suite. Discovery of a MIDlet suite is omitted. Description starts from the point when the user has selected some suite for installation. Figure 11 shows the request for the Application Descriptor (*Reversi.jad*). The client uses *User-Agent* header to identify itself to the server. The server responds by sending the JAD file that is specifically tailored for the model of the client device (Nokia 7210).

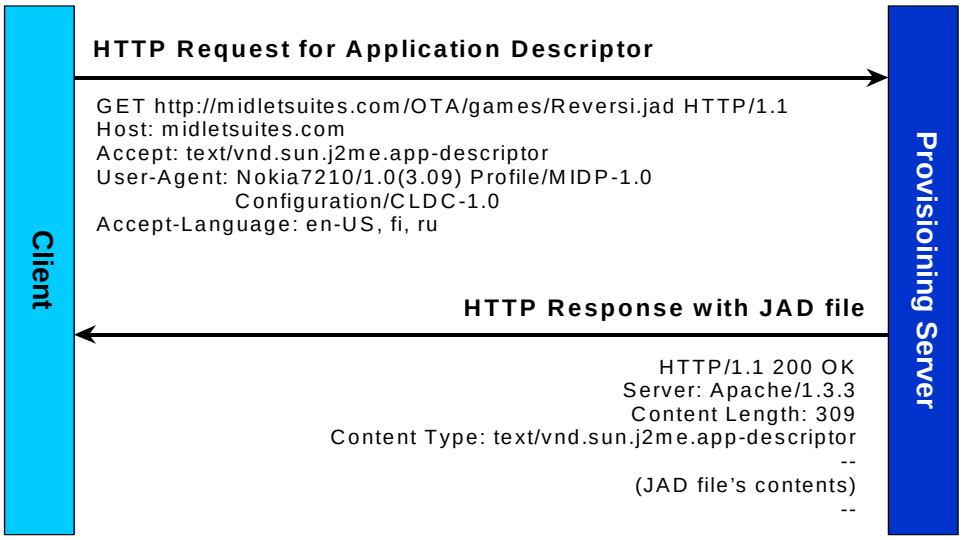


Figure 11. HTTP request for an Application Descriptor

After the AMS has processed the received Application Descriptor and no problems preventing from installation were found, the JAR file is downloaded. The client-server interaction is represented in Figure 12.

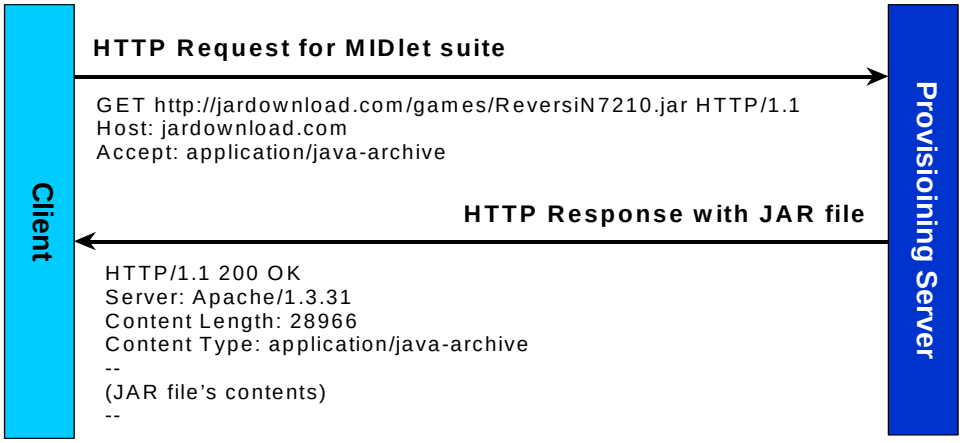


Figure 12. HTTP request for a MIDlet suite

Note that the received JAR file is intended specifically for the model of the client device. This happened because the client has identified itself to the provisioning server while requesting the JAD file. As a result, the server inserted the URL of the appropriate JAR file into the Application Descriptor. That is why (cf. Section 3.2.3) HTTP request headers *User-Agent* and *Accept-Language* are not needed when the JAR file is requested from the URL in the Application Descriptor.

Figures 11 and 12 also show that the MIDlet suite and its Application Descriptor do not have to reside on the same server. In fact, they can reside anywhere in the network, the only link between them being the JAR file's URL in the JAD file (the *MIDlet-Jar-URL* attribute). E.g., all Application Descriptors of a provisioning portal can reside on the same network host, while corresponding JAR files can be stored on multiple hosts. The rationale for such architecture is that JAD files are small and their downloading will not load the network too much. JAR files are larger, and their downloads should be balanced between several servers. A more sophisticated version of this solution may include generation of a JAD file for each client "on-the-fly" and selection of a host for downloading of the JAR file based on information about current loading of available servers.

After the JAR file is received, it is analyzed by the AMS (cf. Section 3.2.3). If all necessary checks are successfully passed, the suite will be installed. The device will try to send the Installation Status Report to the URL found in the *MIDlet-Install-Notify* attribute of the JAD file. The client-server interaction is shown in Figure 13. Even if the installation status report was not sent for some reasons, MIDlets from the Reversi suite are now available for execution.

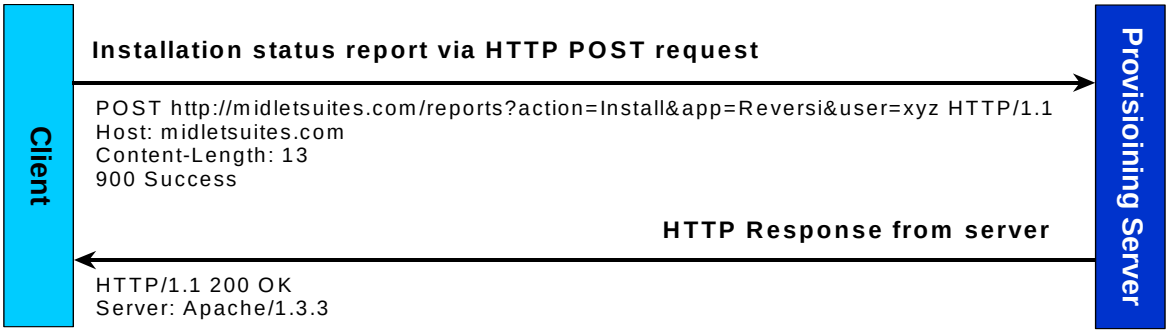


Figure 13. Installation status using the POST HTTP request

Later, when the user chooses to remove the Reversi MIDlet suite, the device will make an attempt to send a Deletion Status Report (cf. Figure 14). Once again, even if the report cannot be sent, the suite will be deleted.

Figure 14. Deletion status using the POST HTTP request

3.3 OTA Provisioning Using WAP Push

The previous Sections have described OTA provisioning of MIDlets as supported on many MIDP 1.0-compliant devices and on all MIDP 2.0-compliant devices. Many MIDlet providers support this provisioning method as well. One of the main advantages of OTA provisioning is that downloading of a MIDlet does not require any other equipment in addition to the mobile device. On the other hand, such scheme greatly restricts the user's ability to discover MIDlets. Indeed, if only an on-device Discovery Application (e.g., a WAP browser) is used for discovery of MIDlets, the discovery process quite often becomes lengthy and too expensive, which is also important. It is lengthy because of low network connection speed and modest screen size of the majority of MIDP-compliant devices. It is expensive since the user usually pays for browsing. As a result, MIDlets cannot be really "discovered" in lots of cases, and the user needs to know their location in advance. This drawback is harmful both to users by restricting their ability to download applications and MIDlet providers especially those who sell MIDlets for money. It is not that easy to seduce the user to buy a MIDlet if its selection can cost more than the MIDlet itself. That is why the classical scheme of OTA provisioning was modified to make it possible to search and select MIDlets using a usual Web browser on a PC.

In this provisioning scenario, the user uses a PC Web browser to discover a MIDlet and then the URL of the suite (or of its Application Descriptor) is pushed to the user's device using the WAP Push mechanism. After the URL is in the device, provisioning happens exactly as in classical OTA provisioning – JAD and JAR files are downloaded OTA using the HTTP. Such approach eliminates problems with suite discovery as in this case, a usual Web site created using HTML and other Web technologies is used to present available MIDlet suites.

3.3.1 WAP Push

In a usual client-server model, the client requests information from the server, then the latter transmits the required information, so the communication process is initiated from the client. Example of such communication is given in Figure 11, where a device requests an Application Descriptor and a server sends it. On the contrary, in the "push" technology, communication is initiated by a server, which transmits information to a client without an explicit request.

Push operation in the Wireless Application Protocol (WAP) is similar. According to [21], it is carried out by allowing a server in the Internet to send some content to the WAP client through a Push Proxy Gateway (PPG). Such scenario is depicted in Figure 15. The server, called Push Initiator (PI), sends information to the client. The Push Access Protocol (PAP) is used for communication between the server and the Push Proxy Gateway. The PPG then uses the Push Over-The-Air (OTA) protocol to deliver the push content to the client.

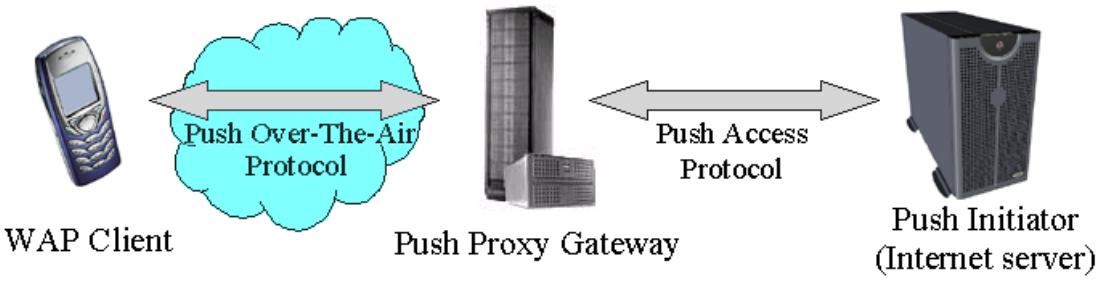


Figure 15. WAP Push architecture

The Push Access Protocol uses the Extensible Markup Language (XML) to express delivery instructions and MIME types to indicate the type of the content being pushed.

The Push Proxy Gateway delivers information to the client, possibly converting the data to adapt it to client possibilities and storing it if the client is currently unavailable. The PPG can also have such features as notification of the Push Initiator about the outcome of push operation, and handling of cancellation, replacement and client capability requests from PI.

The Push Over-The-Air Protocol provides connectionless and connection-oriented services. The connectionless service uses the Wireless Session Protocol (WSP), while the connection-oriented one may use both the WSP and the HTTP. To initiate a communication session, a special message, a Session Initiation Request (SIR), is sent from a PPG to the client. The Session Initiation Application on the client side listens to such messages, activates the required bearer after the user's permission and contacts the PPG. Then the content sent by the PI is delivered to the client. The Session Initiation Request is typically send to the client via the Short Message Service (SMS), as it is available in most mobile networks.

3.3.2 Method Description

To make use of WAP Push provisioning of MIDlets, a device must meet all the requirements listed in Section 3.2.1 and support the WAP Push feature in addition. In fact, the requirement to support the WAP Push means that a device must support WAP version 1.2 or higher.

In WAP Push provisioning (cf. Figure 16), a Web browser installed on a personal computer is used to discover MIDlet suites. A provisioning server in this case is just a Web server that allows users to search and select MIDlets for their devices. After the suite is selected, the user is asked to provide delivery information; typically, it is the user's phone number. If the user has to pay for the suite, some additional information may be needed, e.g., the credit card number. The required information depends on the payment method in use.

Finally, a WAP Push message, containing the URL of the Application Descriptor is sent to the client. The SMS is usually used to notify the user about an incoming push message. Depending on the provisioning server, a JAD file can be the same for all clients who download a particular MIDlet suite, or a personal JAD file can be generated for each client. The same stands for the URL of a JAD file– some servers may use a unique URL for each download. The client uses a received URL to download the Application Descriptor. From this point, the provisioning process goes on as in usual OTA provisioning (cf. Sections 3.2.3 – 3.2.8).

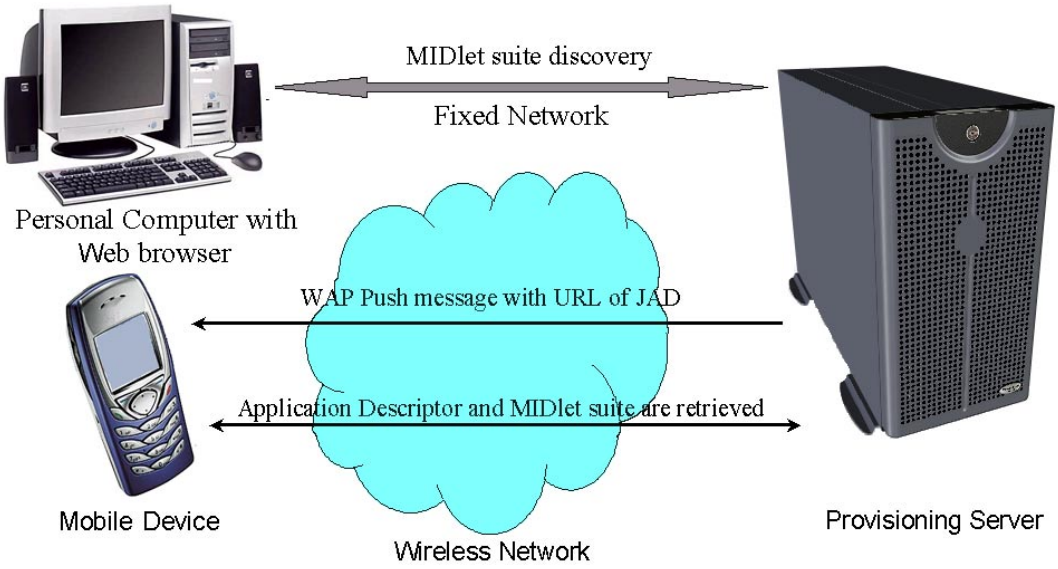


Figure 16. Provisioning using WAP Push

Provisioning of MIDlets using WAP Push has gained ground because it provides convenient means to search and select MIDlet suites. Indeed, when a PC-based Web browser is used to discover MIDlet suites, the user can get much more information than just a brief description of MIDlet's features. It is usually possible to look at screen shots, read a detailed description or even try a MIDlet in the emulator of the device. Another benefit is that distributors that sell MIDlets for money can use standard payment mechanisms adopted in the Internet commerce.

However, the method has its drawbacks. The main disadvantage is that WAP Push (and WAP itself) is not supported in many mobile networks. This fact seriously limits application of this provisioning method. Another problem is increased complexity of the provisioning server introduced by usage of the WAP Push feature; more effort is now required to create and maintain the server.

3.4 Local Provisioning Using Device Connectivity Software

Provisioning technologies described in previous two Sections (3.2 and 3.3) share a distinctive feature: a MIDlet suite resides on a remote server in the network. Therefore, in order to download it, a client needs to connect to the network using an Over-The-Air protocol. It is not always possible or desirable to establish such a connection. First, a mobile device itself may have no support for OTA provisioning (this is not an infrequent case with MIDP 1.0-compliant devices); the same can be true for the wireless network. Second, the user may prefer not to connect to the network just to avoid additional costs on the mobile subscription or because a network connection is not configured. In this case, it is not possible to use either classical OTA provisioning or WAP Push provisioning. But if it is possible to download a desired MIDlet suite to a PC (what is usually the case for suites distributed for free), this suite can be sent to the mobile device using its PC connectivity software. This software comes together with the device or can be downloaded from the manufacturer's Web site for free. It can include support for such functions as synchronization with the PC, setup of various device settings from the PC, uploading of MIDlet suites, uploading and downloading of multimedia objects (pictures, ringing tones, videos), etc. Connectivity software can use different technologies to connect to the device, e.g., an Infrared link, a serial cable link, a Bluetooth link.

If uploading of MIDlet suites is supported by PC connectivity software, the general scheme is always the same: a MIDlet suite stored on the PC is send to the mobile device using some local connectivity technology. Knowledge about the internal structure of the AMS is applied. E.g., suites are sent using the manufacturer's proprietary communication protocol, or the connectivity software may know to what folder(s) in the device file system JAD and JAR files have to be placed, etc. After the suite was sent to the device, it might be necessary to install it using the AMS, but the connectivity software sometimes performs installation as well. Additional features, like review and deletion of already installed suites, may also be supported. However, it should be noted that because of copyright reasons, it is almost never possible to retrieve any MIDlet suites from the device using its PC connectivity software. Indeed, if a commercial suite will be fetched from the device, it will become easily possible to distribute it illegally (as most of MIDlets that are sold for money do not have any protection).

The described provisioning method is widely used for provisioning of free MIDlets, as it is convenient and almost no costs are associated with suite's selection and downloading (it assumed that the Internet connection is free or its cost is fixed). The method also suits MIDlet developers, as it allows to test MIDlets in real devices without wasting time on OTA downloading (which can be long). The main drawback of the method comes from the fact that a deployment protocol is device-specific, even though it may work over a standard bearer (Infrared, Bluetooth, serial cable). This deployment protocol is known only to the device itself and its PC connectivity software, so MIDlet suites can be provisioned only from those PCs where the connectivity software for this particular device model has been

installed. This limitation is overcome by provisioning methods described in the next Section.

3.5 Provisioning via Infrared or Bluetooth Local Connectivity

If the mobile device supports Infrared or Bluetooth local connectivity, then it is sometimes possible to deploy MIDlet suites using standard protocols, viz. IrDA® OBEX™ or OBEX-based Bluetooth profiles, viz. the Object Push Profile (OPP) or the File Transfer Profile (FP). In this provisioning method, a MIDlet suite (both JAD and JAR files or a JAR file only) is sent to the device from any other device (usually from a PC) that supports the same protocol. Software that deals with file uploading on the sending device treats Application Descriptors and Java Archives just as ordinary files that are sent to the remote device. The peer software on the receiving device acts similarly. When the suite is already in the receiver's file system, it can be recognized as a Java application and handed to the AMS for installation. If the AMS supports such type of "local" installation, the suite will be installed. The statements above show that support of this deployment method rests on fulfillment of two requirements. First, the device must accept JAD and JAR files and recognize them as Java applications; second, the AMS must be able to install suites that have already resided in the device file system.

The described provisioning method, being logically the same as the provisioning using PC connectivity software, shares its advantages. At the same time, the method is more flexible as it uses standard protocols, which eliminates the need for device-specific software.

3.6 Provisioning via E-mail or the MMS

Provided that the AMS is able to install MIDlet suites that reside in the local file system, the problem of provisioning is reduced to the problem of sending JAD and JAR (or JAR only) files to the device. It can be accomplished by attaching these files to an e-mail message or a Multimedia Message. The method is somewhat similar to WAP Push provisioning. A message is sent to a client, but now it contains not the URL to the suite, but the suite itself. The method has limited application, as it is quite demanding for a device and a wireless network. Many mobile devices do not support e-mail and the Multimedia Messaging Service, and the same stands for wireless networks.

4. Digital Rights Management

A MIDlet is like any computer program that can be either commercial or free. In this thesis, a commercial MIDlet is the one that has some commercial value and therefore should not be distributed without control. On the contrary, a free MIDlet can circulate freely. A common and well-known problem of commercial software is infringement of copyrights, i.e., illegal copying of programs. Commercial MIDlets also suffer from illegal copying. However, they are less vulnerable because of protection both during deployment and stay on mobile device. During deployment of a commercial suite, it is the provisioning server that ensures that the suite will be delivered to the client that has requested it and not to some other place. And when the suite is in the mobile device, it is usually not possible to take it out. That is why most of commercial MIDlets do not have any internal protection (a serial number, an activation code, etc.). The other reason for not having MIDlet-level protection against illegal copying is limitations of MIDlet's size. Any built-in protection system will inevitably increase the size of a MIDlet, which clearly undesirable because of two reasons. Firstly, the larger is the suite the more money the user spends for download. Secondly, some devices currently have size limitation for MIDlet suites that can be executed or downloaded OTA.

The described protection system is in a way "blind" as it treats commercial and non-commercial suites equally. This means that if someone manages to get a copy of a commercial MIDlet, there will be no technical problem to distribute this MIDlet illegally, e.g., by making it available for downloading from a Web site. Another drawback of the system is that device native software does not only protect commercial MIDlets but also prevents free ones from being freely distributed. In theory, free suites can be sent from one device to another using MMS, e-mail, or local connectivity (Infrared, Bluetooth) or other means. But in practice, it is possible only if there is a reliable mechanism to distinguish between commercial and free MIDlets. That is why it will be very beneficial to have a mechanism that will allow control over distribution of commercial MIDlet suites.

One option is to use the technology proposed in the Open Mobile Alliance™ (OMA™) "Digital Rights Management (DRM)" specification [19]. This specification has a wider scope than just control over distribution of MIDlets. It enables controlled consumption of any digital media objects (images, MIDlets, ring tones, etc.). The essence of the technology is to accompany each media object with usage rights. By formulating these rights, content providers control what happens with their content, e.g., it may be possible only to preview the content (i.e., to execute the MIDlet only once), or it may be allowed to superdistribute the content from one device to another. Some mobile phones with minimal OMA DRM support are already in the market.

Sections 1-6 of this Chapter are devoted to the OMA DRM technology.

Section 7 concentrates on the OMA DRM influence on provisioning of MIDlets. The author proposes several provisioning scenarios that can be used for delivery of OMA DRM-protected MIDlets.

4.1 Overview of Open Mobile Alliance™ Digital Rights Management

Digital Rights Management (DRM) is a technology that allows control over usage of downloadable media objects. The control is exercised by defining and enforcing usage rules for each media object. Content providers define these rules, and different rules can be

associated with the same object. As usage rules are separated from media object, it has become possible to sell not the object itself, but the rights to use it in a certain way. E.g., a MIDlet distributor can allow single execution of a MIDlet for free, and charge only those users who will opt to purchase full usage rights.

Rights are stored in a special object, separately from the media object itself. Objects can be delivered to a device together or separately. The method when rights and media objects are delivered together is called combined delivery (cf. Figure 17). Combined delivery has a special case called forward-lock. In this delivery method, there is no rights object associated with the content, but a set of default usage rules is applied.

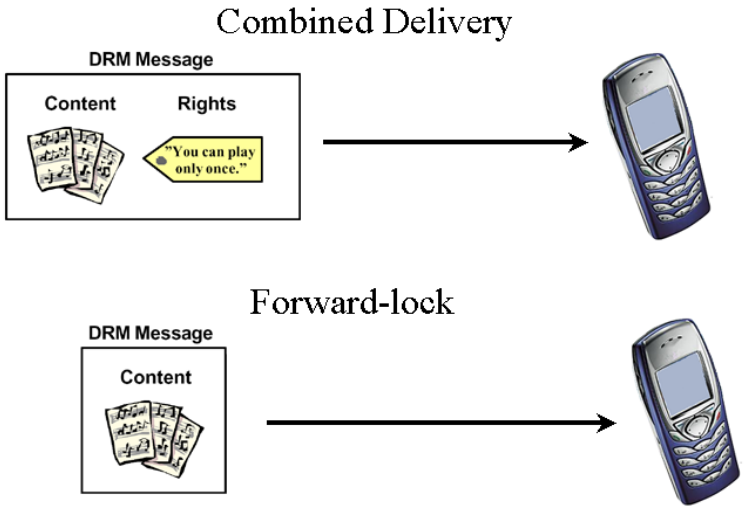


Figure 17. OMA™ DRM combined delivery and forward-lock special case

The method when rights object is sent to a device separately from the media object, typically using another bearer, is called separate delivery. A media object in this case is sent in the encrypted form; and information contained in the rights object is used for decryption. Because of these cryptographic operations and additional transaction needed to download the rights object, separate delivery is a more complex but at the same time more secure delivery solution. Figure 18 illustrates separate delivery.

Figure 18. Separate delivery of a media object and a rights object

The specification [19] assumes that the HTTP is used to download media and rights objects (with the exception of the superdistribution special case, cf. Section 4.4). Though WAP is used as bearer in most examples in the specification; the OMA DRM concept does not stick to WAP and can be used in conjunction with other HHTP bearers as well.

4.2 Combined Delivery

4.2.1 Simplified Case: Forward-Lock

In the forward-lock delivery method, a media object is packed into a DRM message and sent to the device. The format of this message is defined in the specification, the MIME media type is *application/vnd.oma.drm.message*. The syntax of the DRM message is based on the MIME multipart composite type defined in [22]; for the forward-lock delivery method it must contain only one part – the media object itself.

The device retrieves the media object from the DRM message and applies default usage rules. This means that it is allowed to play, display, execute, and print the media object without any restrictions. It is not allowed to forward either the DRM message or the media object to other devices. It is also prohibited to modify the media object.

The forward-lock delivery method must be supported on all devices that conform the OMA DRM.

4.2.2 Normal Combined Delivery

In the combined delivery method, the media object and the rights object are packed into a DRM message and sent to the device. The DRM message must contain exactly two parts; the first part is the rights object. The rights object is of the following MIME type: *application/vnd.oma.drm.rights+xml.* The internal structure of the rights object is defined by the OMA "Rights Expression Language" specification [24] and described in more details in Section 4.5 of this thesis.

After receiving a DRM message, the device retrieves both messages and parses the rights object to determine the rules for usage of the delivered content. Then the received media object (or media objects, as the second part of the DRM message can be a multipart object itself) is consumed according to these rules.

The rights expression language governs only consumption of the content, it does not address its distribution, so the same principle as in forward-lock delivery method is used: neither the DRM message, nor the media object, nor the rights object can be forwarded from the device.

It is not mandatory for an OMA DRM-compliant device to support the combined delivery method described in this Section, but if the device does so, it must support the forward-lock method as well.

4.3 Separate Delivery

The idea behind the separate delivery method is to encrypt the media object and deliver it separately from the rights object that contains the Content Encryption Key (CEK). As the media object is encrypted and useless without CEK, an insecure transport may be used to deliver it, while using a more secure transport for the rights object.

In this method, the symmetrically encrypted media object is in the DRM Content Format (DCF) specified in [23]. The MIME type for the format is *application/vnd.oma.drm.content*. WAP Push is used to send the rights object to the device (WAP Push is described in Section 3.3.1 of this thesis). After receiving the pushed rights

object, the device can decode the content and use it according to rules from the rights object.

The separate delivery method is more complex because it involves two transactions. The media object and the rights object may travel along different paths, and it can happen that the pushed message with the rights object will arrive with some delay as compared with the DCF object. To cope with such situation, the specification proposes to use a special HTTP header *X-Oma-Drm-Separate-Delivery* when sending the DCF object from the server. The value of this header specifies the expected time (in seconds) it will take for the pushed rights object to arrive to the device, e.g.:

X-Oma-Drm-Separate-Delivery : 10

If this header is not present in the server's response with the DCF object, it means that no rights object will be pushed.

Support for a separate delivery method is also not mandatory for an OMA DRM-compliant device, but if it is supported, forward-lock and combined delivery methods must be supported as well.

Similarly to combined delivery, it is prohibited to forward the rights object but at the same time, the DCF file can be freely forwarded, which enables a very important variant of separate delivery, superdistribution.

4.4 Superdistribution

Controlled superdistribution of media objects is possible because in the separate delivery method, the encrypted content and the usage rights are separated. The encrypted media object cannot be used without the appropriate rights object, and therefore, can be freely forwarded. After receiving the encrypted media object (in the DCF format), a device requests usage rights from the rights issuing service. A URL found in the DCF object is used to open a browsing session to this service and the user can request required rights. If the usage rights are granted, then a rights object is pushed to the device.

The main benefit of superdistribution is that it enables forwarding of media objects (including MIDlets) from one mobile device to another without compromising copyrights of providers of these objects. Various technologies, including Bluetooth, Infrared, the MMS, e-mail can be used to send encrypted media objects. Providers retain control over the process by issuing (or not issuing) rights to use the content for each new consumer. Superdistribution is a bright realization of the idea of selling rights to use the content instead of selling the content itself.

4.5 Rights Object

A rights object contains usage rights for some piece of content. These usage rights are expressed using the Rights Expression Language (REL) defined in [24]; its detailed description is out of the scope of this thesis, only a general idea being presented here.

Five permissions to content consumption have been defined:

- Play
- Display

- Execute
- Print
- Modify.

Only Execute permission is applicable to MIDlet suites.

The permissions can be accompanied by constraints. The permission is only granted if all its constraints are met. Three constrains have been defined:

- Count, i.e. the number of time permission is granted
- Datetime<start, end> – permission is only granted if the current date and time is within the specified interval
- Interval, i.e. permission is only granted for a specified period of time. After the specified period elapses, the permission is withdrawn.

Figure 19 gives an example of a rights object.

```
<o-ex:rights
   xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
   xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#/"
>
   <o-ex:context>
      <o-dd:version>1.0</o-dd:version>
   </o-ex:context>
   <o-ex:agreement>
      <o-ex:asset>
         <o-ex:context>
            <o-dd:uid>cid:1234829327@foo.com</o-dd:uid>
         </o-ex:context>
         <ds:KeyInfo>
            <ds:KeyValue>Tei4WxdFh4Gd5D6pLggc=</ds:KeyValue>
         </ds:KeyInfo>
      </o-ex:asset>
      <o-ex:permission>
         <o-dd:execute>
            <o-ex:constraint>
               <o-dd:count>1</o-dd:count>
            </o-ex:constraint>
         </o-dd:execute>
      </o-ex:permission>
   </o-ex:agreement>
</o-ex:rights>
```

Figure 19. XML representation of rights

The rights in this example give a permission to execute a piece of DRM content (e.g., a MIDlet suite) only once. The textual XML notation is used; presence of the <KeyValue> element with the Content Encryption Key makes it clear that this rights object is for the separate delivery method. Since the rights object should be as small as possible (to enable transmission over constrained bearers, e.g., WAP Push over SMS), another notation of rights is also in use. This notation uses the WBXML (the binary XML) defined in [25] and is much more compact. The textual XML is used in combined delivery, while in separate delivery both notations can be exploited. Because of that, rights objects can be of two MIME types:

- *application/vnd.oma.drm.rights+xml* for a textual XML

- *application/vnd.oma.drm.rights+wbxml* for a binary XML.

For a textual rights object, the file extension is ".dr" and for WBXML encoded ".drc".

4.6 Security Considerations

In the model proposed in the OMA DRM specification, the value is associated with rights to use the content and not with the content itself. The consuming device uses rights objects to control how the DRM content is used. This means that both DRM content and rights objects must be protected from being accessed by unauthorized software. Indeed, if the device allows installation of additional non-Java applications, it may be possible for these applications to violate the DRM security model by, e.g., forwarding of content that must not be forwarded, illegal changing of content usage rules, etc. The specification assumes that the compliant device has a mechanism in place to prevent unauthorized access to the content and rights objects, e.g., they can be stored in an encrypted form. Such mechanism is not necessary for the devices that do not support installation of additional non-Java software.

A security model described in previous Sections of this Chapter is not perfect. The specification admits that the security model is a kind of reasonable trade-off between complexity of the protection system and immunity of the content being protected. Higher levels of security require a more sophisticated protection system, but the need for such system is questionable, bearing in mind the value of the content the OMA DRM is intended for.

4.7 OMA DRM and Provisioning of MIDlets

From the point of view of the OMA DRM specification, MIDlets can be viewed as DRM content, and therefore all mechanisms described in this Chapter can be applied to provisioning of MIDlet suites. However, support of the OMA DRM requires introduction of some changes in provisioning methods described in Chapter two of this thesis. The next two Sections are devoted to the author's proposals on how OMA DRM combined and separate delivery methods can be incorporated into classical OTA provisioning. Section 4.7.3 describes the author's view on the OMA DRM superdistribution of MIDlets.

4.7.1 OTA Provisioning Integrated with OMA DRM Combined Delivery

A crucial change that has to be introduced into OTA provisioning to enable support for combined delivery is downloading of a DRM message instead of a JAR file. This results in changes both on the mobile device and on the provisioning server. The mobile device has to be aware of the fact that a MIDlet suite can be packed into a DRM message and act accordingly when receiving the message. The OMA DRM-aware provisioning server must be able to pack JAR files and appropriate rights objects into DRM messages and place URLs of these messages into Application Descriptors.

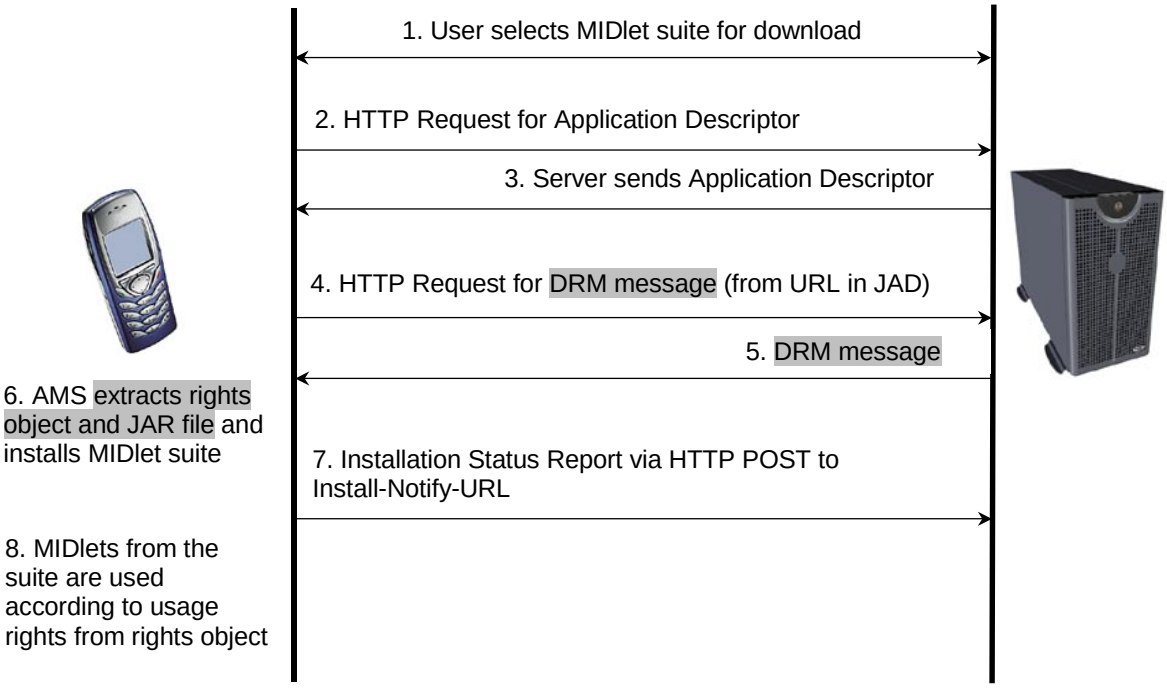


Figure 20. Incorporation of the OMA DRM combined delivery method into OTA provisioning

Figure 20 shows a high-level view of OTA provisioning with support for combined delivery. Changes from normal OTA provisioning are marked with grey. Here is a brief description of each step.

1. The user selects a MIDlet suite for downloading in the same manner as in normal OTA provisioning (cf. Section 3.2.2). A Discovery Application is used, e.g., a WAP or a Web browser. A browser has to be aware of objects of the *application/vnd.oma.drm.message* MIME type. As objects of this type can contain any kind of DRM content (ring tones, MIDlet suites, images, etc.), a mechanism for dispatching of content is needed. This mechanism may be implemented, e.g., as a separate system end employed each time when the object of this type is encountered. Finally, a browser must hand a DRM message with a JAR file to the AMS
2. The Application Descriptor of the selected suite is being requested (if the suite has any). This step is the same as in normal OTA provisioning
3. The server sends the Application Descriptor. Its *MIDlet-Jar-URL* attribute contains a URL that points to the DRM message (not the JAR file as in normal OTA provisioning)
4. The device requests a DRM message from the URL in the Application Descriptor (the *MIDlet-Jar-URL* attribute). The requested MIME type is *application/vnd.oma.drm.message*
5. The provisioning server responds with the DRM message. It may contain one (the forward-lock special case) or two (normal combined delivery) parts. If there are two parts in the message, then the first one is always the rights object and the second one is the MIDlet suite
6. The AMS extracts the rights object (if any) and the JAR file. The JAR file is examined according to Section 3.2.3. If all necessary checks are passed, the suite is installed and usage rights from the rights object are associated with it

7. The Installation Status Report is sent exactly like in normal OTA provisioning (cf. Section 3.2.7)
8. MIDlets from the newly installed MIDlet suite are executed according to usage rules from the rights object or default rules are applied in its absence (in case of the forward-lock delivery method). The rights object is of the *application/vnd.oma.drm.rights+xml* MIME type, i.e., the textual XML is used to express rules. Section 4.4 has already mentioned that the only permission from those defined in [24] applicable to MIDlet suites is Execute. It can be used in conjunction with any of three constrains; rules defined in the rights object apply to all MIDlets in the MIDlet suite. E.g., if the rights object in Figure 19 is for a MIDlet suite, it is allowed to execute each MIDlet from the suite only once.

It should once again be noted that the scenario sketched above is not a standard provisioning method, but rather the authors' proposal of how MIDlets can be deployed using the OMA DRM combined delivery method.

4.7.2 OTA Provisioning Integrated with OMA DRM Separate Delivery

Support for separate delivery requires more changes in the normal OTA provisioning scenario than support for combined delivery. There are two reasons for this. First, the rights object is separated from the MIDlet suite and may travel along the different path, which potentially may cause delay in its arrival. Second, as the MIDlet suite is in the DRM Content Format, it should be decrypted using the Content Encryption Key from the rights object before installation. All that makes the installation process more complex.

Figure 21 gives an idea of how the separate delivery method can be introduced into OTA provisioning. Changes are marked with grey.

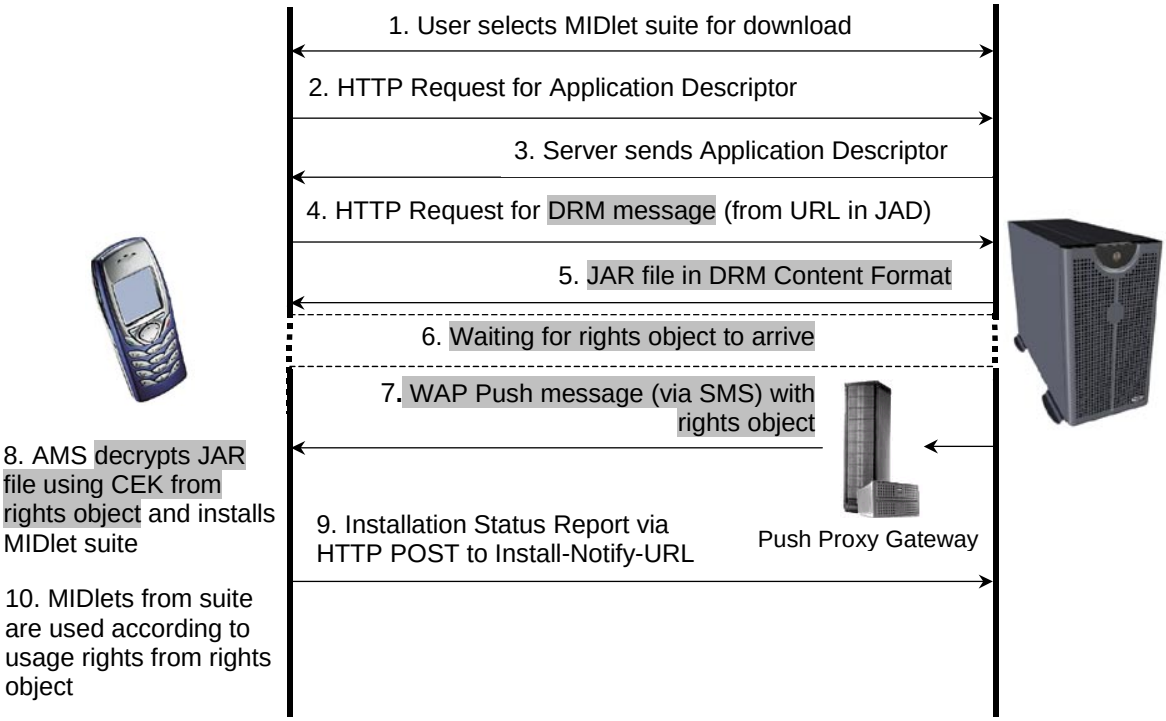


Figure 21. Incorporation of the OMA DRM separate delivery method into OTA provisioning

Phases of the process in details (cf. Figure 21):

1. A discovery of a MIDlet suite happens in the same way as in OTA provisioning, but a Discovery Application must also support the DRM Content Format MIME type: *application/vnd.oma.drm.content*. Similarly to combined delivery, some objects of this type can contain MIDlet suites, so there should be a mechanism for dispatching DCF objects to appropriate consumers. Objects with JAR files must be handed to the AMS
2. The Application Descriptor of the selected suite is being requested (if the suite has any). This step is the same as in OTA provisioning
3. The server sends the Application Descriptor, the URL in its *MIDlet-Jar-URL* attribute points to the message in the DRM Content Format
4. The DCF object is requested from the URL in the Application Descriptor. The requested MIME type is *application/vnd.oma.drm.content*
5. The server sends the DCF message. It contains an encrypted JAR file. When sending the message, server uses *X-Oma-Drm-Separate-Delivery* (cf. Section 4.3) to indicate that a rights object will be pushed to the device. The server can also roughly estimate how long it will take for the push message to arrive
6. The device stores received the DCF message and, if the *X-Oma-Drm-Separate-Delivery* header was present in the server's response, waits for the rights object to arrive. A time-out should be specified for waiting, e.g., it can be the value specified in the *X-Oma-Drm-Separate-Delivery* header (if the value is present) or a device-specific value. If the waiting time elapses and no rights object has been pushed, then two options are available: either to abandon the installation or to explicitly request a rights object from the Rights-Issuer URL (cf. Section 4.4). If the above-mentioned header is not present in the server's response, it means that

the rights object will not be pushed and should be requested explicitly (the superdistribution special case)

7. The rights object is pushed to the device using WAP Push (usually over the SMS). Either textual (*application/vnd.oma.drm.rights+xml* MIME type) or binary (*application/vnd.oma.drm.rights+wbxml*) XML representation of the rights object can be used. When the SMS is used as a bearer for WAP Push, binary representation is more advantageous as a rights object in this format can fit into a single SMS message
8. Now the AMS can use the CEK from the rights object to access the encrypted JAR file inside the DCF message. The JAR file is examined according to Section 3.2.3; here the value of the *MIDlet-Jar-Size* attribute is compared with the size of the decrypted JAR file. If all necessary checks are passed, the suite is installed and usage rights from the rights objects are associated with it.
   The OMA DRM specification does not define how the installed MIDlet suite from the DCF is stored on the client device, it only demands to follow the security rules for the separate delivery method. These rules are: the JAR file can leave the device only encrypted, in the DCF format and the rights object must never be forwarded. In fact, there are two options for storage of the installed JAR file: it can be stored as a plaintext and encrypted only when being forwarded from the device. The other option is to store the JAR file encrypted in the DCF message and decode it "on-the-fly" each time its content is accessed. The first method is suitable for those mobile devices that do not allow installation of additional non-Java applications, thereby ensuring that the JAR stored in plaintext would be accessed only by authorized software. The second method is for mobile devices that have open file systems and allow installation of third-party software that may violate the OMA DRM security model
9. The Installation Status Report is sent like in normal OTA provisioning
10. MIDlets from the newly installed suite are executed according to usage rules from the rights object. The difference from OTA provisioning with combined delivery (described in the previous Section) is that the JAR file (encrypted and placed to the DCF message) can be freely forwarded from the device.

Note that the method above is not a standard provisioning method but the author's idea of how the OMA DRM separate delivery method can be applied for provisioning of MIDlets.

It should be noticed that almost all said in this and the previous Sections goes for WAP Push provisioning as well. Indeed, Sections 3.2 and 3.3 of this thesis show that OTA provisioning and WAP Push provisioning are the same at all points except the MIDlet suite discovery phase. This phase is not affected by the OMA DRM specification, and, therefore, combined and separate delivery methods can be incorporated into the WAP Push provisioning method in much the same manner as described for OTA provisioning.

4.7.3 Superdistribution of MIDlets

The OMA DRM superdistribution method (cf. Section 4.4) allows forwarding of MIDlet suites from a mobile device without violation of copyrights of the suites' developers or distributors. Forwarding of MIDlets from one mobile device to another is the most interesting possibility provided by superdistribution. The mechanism works as shown

in Figure 22: mobile device A has an installed MIDlet suite which was provisioned using the separate delivery method. The user of mobile device A wants to send this suite to the user of mobile device B. The encrypted JAR file packed into the DCF object is forwarded to mobile device B. Different bearers, both OTA (e.g., MMS, e-mail) and local (e.g., Bluetooth, Infrared), can be used. Upon receiving the DCF object, the mobile device B uses a URL from the DCF object to open a browsing session with rights issuing service. During this browsing session the user of mobile device B requests a desired level of permissions for the MIDlet suite. If some permissions are granted, a rights object is pushed to the device using the WAP Push mechanism. Mobile device B can now install the MIDlet suite and use it according to acquired permissions. The user of mobile device B can forward the suite further; the process can be reiterated unlimited number of times, which is why it is called superdistribution.

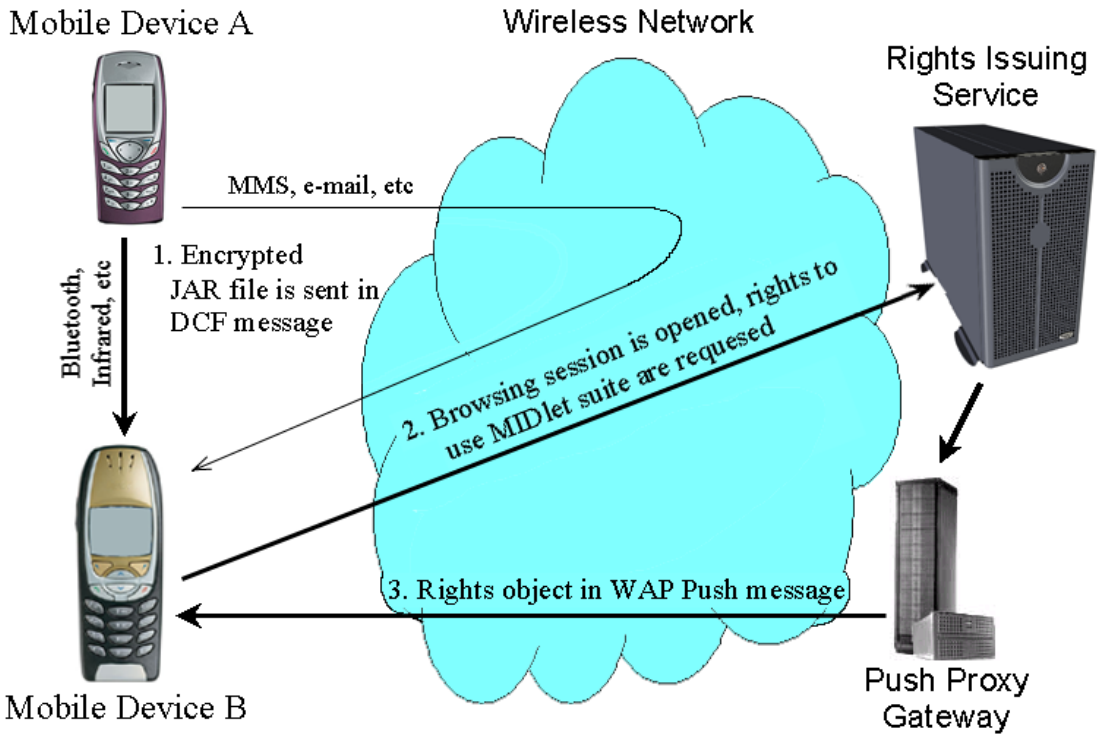


Figure 22. OMA DRM superdistribution of a MIDlet

Superdistribution of MIDlet suites is attractive mainly for developers and distributors of MIDlets to be sold. Indeed, this scenario assumes that every owner of the suite is a potential distributor, which should mean increase in sales volumes. Developers of free MIDlets would benefit from superdistribution to a lesser degree, at least directly. The reason is that free MIDlet suites, in practice, do not need protection provided by OMA DRM, so the only reason to distribute them using the separate delivery method is to enable superdistribution. At the same time, in order to facilitate superdistribution of any MIDlet suite, a rights issuing service should be arranged on a server in the Internet. Many developers of free MIDlets are not able to arrange such service themselves, but some provisioning portals may offer such a service for free. Still, even if superdistribution is enabled for a free MIDlet suite, many end users may prefer not to use this delivery method just because requesting a rights object requires an Over-The-Air network connection,

which is usually chargeable. Free MIDlet superdistributed this way is not free from the user's point of view.

However, in the long-term outlook, ubiquitous adoption of OMA DRM for distribution of commercial MIDlets may ease superdistribution of free MIDlets. Indeed, when all commercial content is distributed using methods described in this Chapter, all non-DRM content can be considered free and distributed without restrictions. There is just one obstacle – universal adoption of OMA DRM methods is required.

5. Bluetooth Wireless Technology

The Bluetooth wireless technology can be used for local provisioning of MIDlets. The full specification of the technology is quite extensive, so this Chapter gives only a brief overview, mostly concentrating on device and service discovery and object transfer capabilities, as relevant to the topic of this thesis. Description of low-level protocols (radio, baseband) along with audio communications is almost totally omitted.

5.1 Overview

The Bluetooth wireless technology has its origin in the research project of the Swedish telecommunication manufacturer Telefonaktiebolaget LM Ericsson that started in 1994. The goal of the project was a research into possibilities of replacement of cables between mobile phones and their accessories with a short-range radio link. Conducted studies had shown usefulness of the new technology, and Ericsson has made a decision to offer it as an industry standard. To produce a common open specification, the Bluetooth Special Interest Group (SIG) was formed. The founding companies were Ericsson, Intel Corporation, IBM Corporation, Nokia Corporation and Toshiba Corporation. The group was publicly announced in 1998 and one and a half years later the version 1.0 of the specification was published.

Here are the highlights of the Bluetooth wireless technology (according to [26]):

- Open specification. All specifications produced by the Bluetooth SIG are publicly available and royalty free
- Short-range wireless connectivity. Communication goes over an air-interface on radio frequency, the range of communication is nominally defined as 10 meters (potentially – up to 100 meters), and communicating devices can be out of "line-of-sight", as radio waves penetrate obstacles
- Intended both for voice and data
- Can be used in any country without a license as a frequency spectrum used for communication does not require licensing throughout the world.

The specification consists of two main volumes: core specification [27] and profiles specification [28]. Other important document [29] contains constants defined for the technology. This Chapter describes the Bluetooth wireless technology according to the specification version 1.1.

5.2 Establishment of Bluetooth Connection

The greatest benefit of the Bluetooth wireless technology is that communicating devices do not need to be connected with wires, they can join and leave the network at any time. However, before any information exchange can start, two devices have to discover each other; in wired networks, this is done by physically connecting the devices with a cable. In the Bluetooth wireless technology, the same is accomplished by two procedures: device discovery (cf. Section 5.2.1) and service discovery (cf. Section 5.4). After two

devices have discovered each other, they can establish a connection. Bluetooth network models and types of connection are described in Sections 5.2.2 and 5.2.3 respectively.

5.2.1 Device Discovery

Device discovery (inquiry, according to the specification) is used by a device to search for other Bluetooth devices in the vicinity. The process goes in the following manner. A device that wants to discover other devices enters an inquiry state and starts to send inquiry messages. Devices that want to be discovered regularly enter an inquiry scan mode, in this mode, they are able to receive and respond to inquiry messages. Upon receiving an inquiry message, a device that wants to be discovered responds with a message that contains the following information: the device's Bluetooth address (which is unique for each device), synchronization information and the class of device (CoD) information. The class of device is a 24-bit field that provides general information about the type of the device (e.g., a mobile phone, a laptop, etc.) and services available on it (e.g. audio, networking, etc.). As a result of inquiry, the device receives a collection of the above-mentioned values for all devices that responded to inquiry messages. This information (viz., the Bluetooth address and synchronization data) is enough to establish a low-level Bluetooth link between the devices (cf. Section 5.2.3). More details about the inquiry procedure can be found in the Bluetooth "Baseband Specification" ([27], Part B).

5.2.2 Communication Topologies

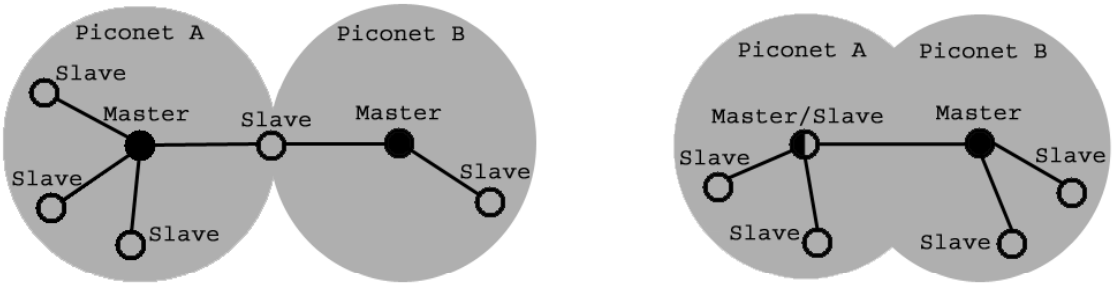
When two devices establish a Bluetooth link, one assumes the role of *master* and the other – the role of *slave*. The role of master does not imply any privileges, it just means that the device controls the synchronization of radio communication between the devices; in general (when communicating using high-level protocols), devices operate as peers. Master and slave roles are temporary and are valid only during current connection; in general, any Bluetooth device may assume either role or even combine both. The device that initiates communication assumes the master role, but it can be possible to switch roles later on.

A simple Bluetooth network is called a *piconet*. It consists of one master device and from one to seven active slave devices. Figure 23 (a) shows a simplest piconet (point-to-point connection). Figure 23 (b) gives a more general piconet configuration. Note that all slaves in the piconet are connected to the same master. It is not possible to connect two slave devices directly.

(a) Simplest piconet (b) General piconet

Figure 23. Piconet examples. Source: [30]

To make the network structure more flexible and to overcome limitation on the number of simultaneously connected devices, another type of network topology is defined – a *scatternet*. A scatternet is formed when two or more piconets overlap. In such scheme, one slave can participate in multiple piconets (cf. Figure 24 (a)) or even be a slave in one piconet and a master in the other (cf. Figure 24 (b)). In general, support for a scatternet is more technically demanding, for this reason many Bluetooth devices do not support scatternet topology.



(a)
The same device is a slave in two piconets

(b)
The same device is a master in one piconet and a slave in the other

Figure 24. Scatternet examples. Source: [30]

A piconet (and scatternet) is established using the so-called page operation. In essence, during page operation, one device explicitly "invites" another to join its piconet. This invitation is called a *page*, and the device that sends it (future master) is the paging device. A device that listens to pages (future slave) is called the paged device. The result of successful page operation is that slave joins master's piconet, i.e. they are synchronized and ready to start communication. Note that at this point, there is no connection that can be used for data transfer between master and slave. All communications serve the purposes of piconet establishment and synchronization between the devices.

5.2.3 Connection Establishment

Once the piconet is created, the devices can establish a connection. In this case, the connection initiator sends a request, which the other device may accept or reject. According to [27] part B, two types of links can be established between master and slave:

- Asynchronous connection-less (ACL) link
- Synchronous connection-oriented (SCO) link.

Initially, when devices have decided to establish a connection, an ACL link is created. There can only be one such link between master and slave. The ACL link provides a packet-switched connection between the devices. The data is transmitted as it became available from higher-level protocols in an irregular manner. Over an ACL link, data can be transmitted at speeds up to 723.2 kB/s ([27] part B).

The SCO link is a point-to-point link between master and slave. These types of links can be considered as circuit-switched connections because slots are reserved regardless of

availability of data to be transmitted. The SCO link is used for real-time information like voice or some general synchronous data. There can be up to three links of that type between the master and one or several slaves. A slave can support up to three links from the same master or two links from different masters. The SCO link can be established only after the ACL link was created. Every link has a data rate of 64 kB/s in both direction ([27] part B).

5.2.4 Name Discovery

Each Bluetooth device can have the so-called user-friendly name associated with it. This name is meaningful for the user (e.g., "Bluetooth Workstation") and is used for UI purposes, such as presenting a list of devices found during inquiry. To perform the name discovery on the remote device, there is no need to establish an ACL link; the information about the device obtained during inquiry is enough to make a name request. The device responds with its user-friendly name.

On some Bluetooth devices, the user-friendly name is hard-coded; on the others, the user can change it.

5.3 Bluetooth Middleware Protocols

The Bluetooth protocol stack is quite extensive, and its low-level protocols are of no interest in the context of this thesis. To simplify the discussion, their descriptions are omitted, along with descriptions of irrelevant middleware protocols.

Figure 25 depicts middleware protocols of the Bluetooth stack. Among protocols in the diagram, Service Discovery Protocol (SDP) and the OBEX protocol on top of the RFCOMM protocol are of particular interest in connection with the subject of this thesis. The Service Discovery Protocol (cf. Section 5.4) is used to describe services available on the device and to locate services on other devices. The RFCOMM protocol provides an interface similar to a typical serial port interface. A whole family of higher-level protocols reside on top of RFCOMM; some of these protocols are not Bluetooth-specific but adopted from other standards. They are OBEX (the IrDa interoperability protocol) and the TCP stack of protocols.

It should be noted that low-level transport protocols in Figure 25 should not be confused with the Transport layer of the OSI model. The Bluetooth low-level protocols, in fact, span over Network, Data link and Physical layers of the OSI model.

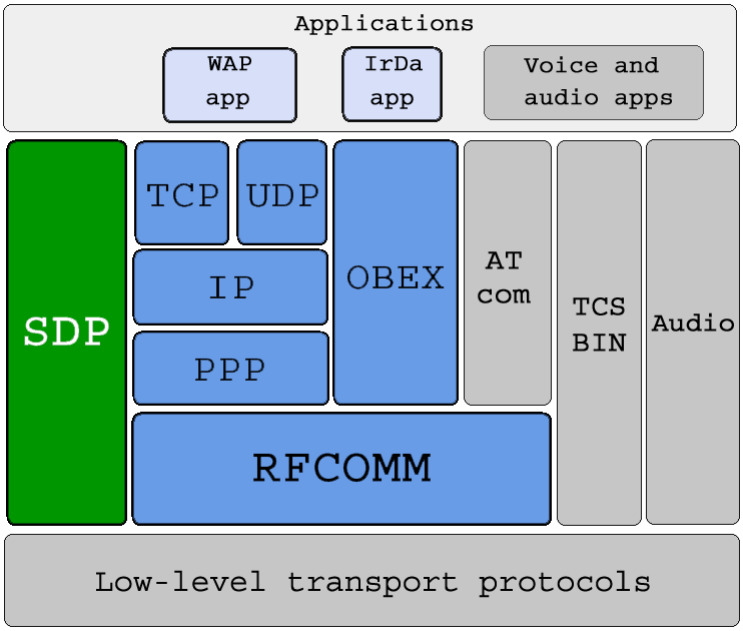


Figure 25. Middleware protocols of Bluetooth stack

Detailed information on Bluetooth protocols can be found in the Bluetooth "Core" specification [27].

5.4 Service Discovery

A service in the network is any kind of resource that can be used by network members. Examples of a service include: a printer, a storage, e-mail, etc. Networks are formed to allow devices to make use of services provided by other devices and to provide services themselves. To use a service, a client needs to locate it in the network and be properly configured. In traditional networks, this can be done statically, e.g., by the system administrator. However, such approach is unsuitable for Bluetooth ad-hoc networks.

The need for dynamic service discovery comes from the dynamic and ad hoc nature of Bluetooth networks. Devices can join and leave the network at any moment, and each of them can provide a different set of services. To make use of each other's services, devices should be able to describe their own services and locate remote ones. In the Bluetooth wireless technology, service discovery is a process by which one device can locate services provided by or available through another Bluetooth device.

Bluetooth Service Discovery Protocol is used for service discovery. According to [26], SDP includes the notion of a client (the entity looking for services) and a server (the entity providing services). Any device can act both as a server and as a client at the same time. The service provider advertises available services through service records. Each service record describes a service in a standard manner defined in the specification [27]. The list of all service records advertised by an SDP server is called registry or service discovery database (SDDB).

A service record consists of service attributes that contain various information about the service. This information usually includes a class of the service, information about the protocol stack layers that are needed to interact with the service, etc. Figure 26 shows the

structure of the SDDB. It can be seen as just a set of records representing all services that a Bluetooth device can offer to other Bluetooth devices. The other fact illustrated by the diagram is that only two attributes (*ServiceRecordHandle* and *ServiceClassIDList*) must be present in each service record, other attributes are optional.

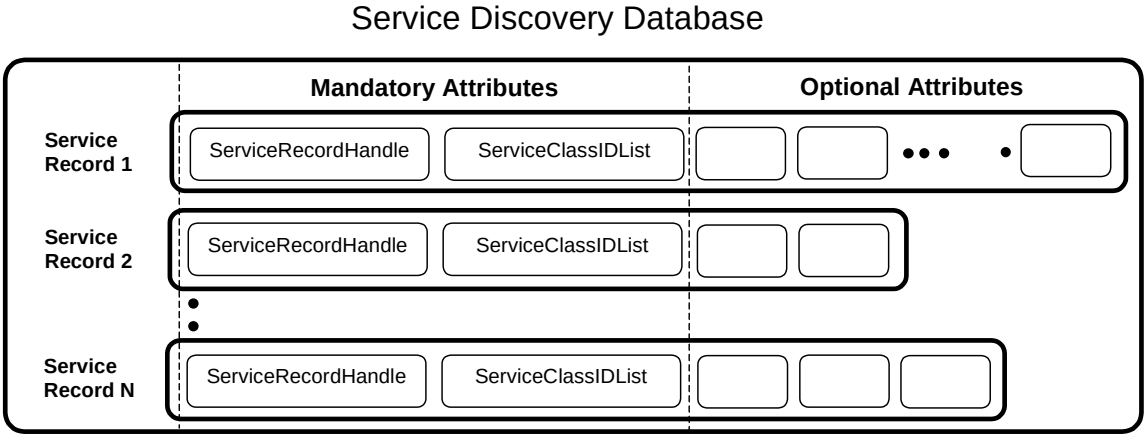


Figure 26. SDDB structure

According to [26], discovery of a service using Bluetooth SDP takes place like this: the client specifies the service of interest and the server responds, indicating any available services that match what the client specified. This means that some standard method is needed for the client to represent the service of interest and for the server to match its available services against the client's specification. For this purpose, SDP uses universally unique identifiers (UUIDs).

Universally unique identifier is a 128-bit value; UUIDs are created using publicly available algorithm defined by ISO in [31]. The algorithm guarantees (with very high probability) that new UUIDs are unique and allows distributed allocation, i.e. there is no need for central registry as UUIDs can be created as needed.

In Bluetooth SDP, every service or service class has a UUID associated with it. UUIDs for services related to profiles (cf. Section 5.6) are set by the Bluetooth SIG and can be found in [29]. When a new service is created, its developer allocates new UUID and associates it with the service.

As the statements indicate, a client looking for a service just specifies a UUID (or UUIDs) associated with this service in its service search request, and the SDP server matches that UUID (UUIDs) against those of the services it has available and generate the response.

More detailed information on Bluetooth SDP can be found in the Bluetooth "Service Discovery Protocol" specification ([27] part E).

5.5 Bluetooth Security

Bluetooth wireless communication as any wireless communication is subject to risk of illegal access to the information transmitted. To cope with this risk, several security algorithms and procedures are defined by the specification.

5.5.1 Authentication

In general, authentication is the process by which one entity proves its identity to another entity. In the Bluetooth technology, the term has similar meaning: the process by which one Bluetooth device proves to another Bluetooth device that it is the one it declares to be. The procedure is based on a 128-bit long shared secret key called a *link key*. When authentication happens for the first time, the devices do not have this key. To create it, they have to undertake a pairing procedure.

According to [26], during the pairing procedure, both devices generate an *initialization key* based on a common PIN entered on both devices and a Bluetooth address of one of the devices. The devices that do not have a user interface have a fixed, non-changeable PIN. In this case, the PIN entered on a device with a user interface have to be the same as the fixed one. If both devices have fixed PINs, they cannot be paired, and, therefore, authentication is not possible. Note that during the pairing procedure, neither PINs that were used to generate secret keys, nor keys themselves are sent over the air.

After the pairing (if it was necessary), the two devices share a secret key (either a link key or an initialization key) and the authentication procedure goes as shown in Figure 27. Challenge-response transaction takes place as follows. The device that initiates authentication procedure (a verifier) challenges the other device (a claimant) by sending a 16-byte random number. The claimant operates on this number using the secret key and its own Bluetooth address. The result is returned to the verifier. In the meantime, the verifier performs the same operation on the same random number and then compares the results. If the results are equal, the verifier consideres the claimant as authenticated device. After that, if the *initialization key* was used during authentication, the devices can agree upon a shared *link key*. This *link key* can be stored and used for future authentications or pairing can be performed each time authentication between the devices is required.

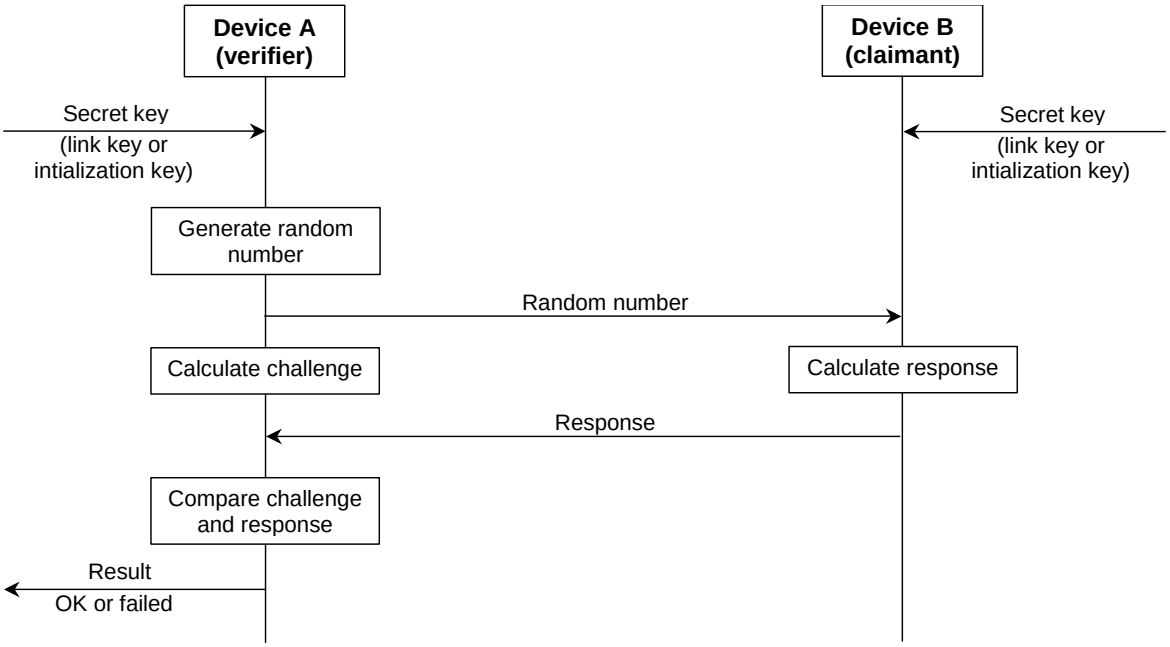


Figure 27. Bluetooth authentication. Source: [27] Part K1

Note that the process described above results in a one-way authentication. Thus, Figure 27 shows that Device A has authenticated Device B. When mutual authentication is required, the devices exchange their roles as verifier and claimant and the same procedure is performed once again.

From the user's point of view, the authentication looks very simple: users just enter the same PIN code on both devices.

More detailed information on Bluetooth security mechanisms can be found in Bluetooth "Baseband" and "Link Manager Protocol" specifications ([27] parts B and C).

5.5.2 Encryption

According to [26], privacy of the data flowing over a Bluetooth link can be protected by link encryption. Encryption in the Bluetooth technology is based on a 1-bit cipher described in the specification ([27] Part B). The encryption key is derived from the link key used to authenticate the devices. The maximum size of the encryption key is 128 bits. Once the encryption is employed over the link, both directions of communication are encrypted. As encryption is a link property, both ACL and SCO packets over the encrypted link are encrypted.

5.6 Bluetooth Profiles

Before creating a specification, the Bluetooth SIG defined several usage models (or use cases) that the new technology should make possible. Here are some of these use cases: wireless connection of a headset to a mobile phone, wireless file transfer, wireless LAN access, wireless modem. Some of the use cases were later formally specified as Bluetooth profiles. A Bluetooth profile specification defines what protocols are needed and how they should be used to support a particular usage model. The need for profiles comes from the fact that the Bluetooth protocol stack is quite extensive, and various Bluetooth devices may support different set of protocols. As a result, interoperability may suffer. However, interoperability was always a matter of great concern for the Bluetooth SIG. In that way, Bluetooth profiles ensure interoperability among many implementations of the Bluetooth stack [26]. However, not all profiles embody some specific use cases; some profiles are defined to serve as a common base for several other application profiles.

Figure 28 depicts Bluetooth profiles of specification version 1.1 in class hierarchy representation. The hierarchy is based upon protocol stack relationships. The higher the profile is placed in the diagram, the lower-level protocol (or protocols) it addresses. E.g., the Generic Access Profile (GAP) mostly describes how low-level transport protocols should be used, while the File Transfer Profile mostly addresses OBEX protocol. The profiles in the diagram are also grouped, basing on the functions they perform.

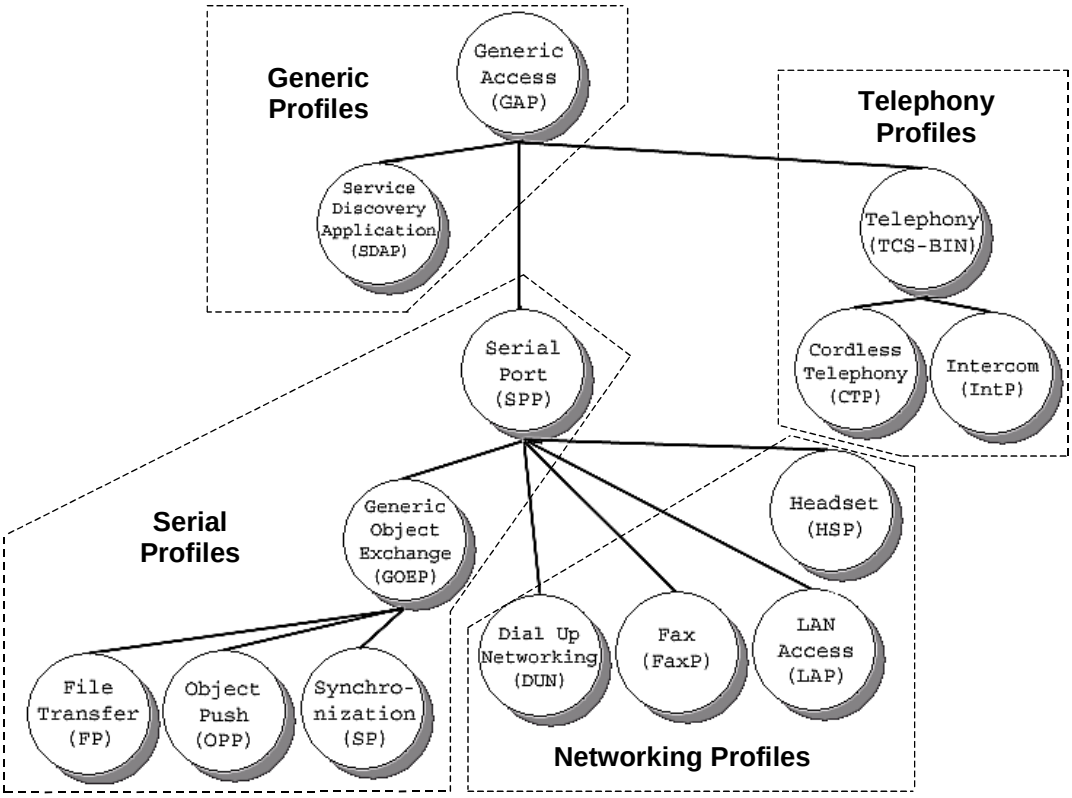


Figure 28. Bluetooth profiles in class hierarchy based upon protocol stack relationships. Grouping is function-based. Source: [26]

Profiles that span off the Generic Object Exchange Profile (GOEP) are in the focus of this thesis; they are elaborated in the next Sections of this Chapter. The other profile of interest is the Service Discovery Application Profile (SDAP). This profile defines a common and standard method for performing service discovery using the Bluetooth protocol stack and establishes a standard discovery application model. The SDAP is based on SDP protocol (cf. Section 5.4) and assumes that a device participating in service discovery has either of the following roles:

Local device – implements the client portion of SDP and initiates SDP transactions (queries for services).
Remote device – implements the server portion of SDP and responds to service inquiries.

Some Bluetooth devices can act only as local or as remote device but usually a Bluetooth device can assume both roles. These roles are meaningful only when an SDP transaction between two devices is underway and have nothing to do with master and slave roles [26]. The specification of the SDAP can be found in [28] part K:2.

A detailed description of other profiles in Figure 28 can be found in [28].

5.6.1 OBEX-Based Profiles

The profiles in Figure 28 that are based upon the Generic Object Exchange Profile (GOEP), viz. the File Transfer Profile (FP), the Object Push Profile (OPP), the Synchronization Profile (SP) and the GOEP itself are known as IrDA® interoperability profiles. As the name of their parent profile implies, they all deal with transfer of objects

between two Bluetooth devices, and, therefore some of them can be used for provisioning of MIDlets.

Generic Object Exchange Profile

According to [26], the GOEP is an abstract profile that is not expected to be used directly by applications. Instead, it serves as a foundation for other OBEX-based profiles, uniting elements that are common for these profiles. First of all, it is a client-server model inherited from IrDA OBEX protocol (OBEX protocol is defined in [32]). The client device in this model is the one that pushes or pulls objects to a server, while the server device provides the object exchange service that allows those objects to be pushed to and pulled from it. It is the client that locates a server and initiates operations. It should be noticed that these client and server roles have nothing to do with master and slave roles (cf. Section 5.2.2), the GOEP client could be either a master or a slave device, and the same stands for the GOEP server.

The GOEP defines the primitives for object exchange; the most important of them being object push and object pull. These two operations are used in all three higher-level profiles. In addition, the profile defines how to establish and terminate OBEX connections and how to use common OBEX functions. The GOEP is described in details in [28] part K:10.

Object Push Profile

The simplest GOEP-based profile is the Object Push Profile. This profile was initially intended for exchange of electronic business cards. Actually, it may be used with any kind of objects, not only those of the vCard format (cf. [38]). The profile's name implies one-way object transfer – pushing them from the client to the server; with the only exception: the client can pull the server's default business card.

According to [26], the OPP assumes compliance with the GOEP and further elaborates scenarios associated with object push (and default business card pull). The OPP specifies a client and a server role more precisely, refining them as a push client and a push server. Similarly to the GOEP, the push server is the device that provides the object exchange service; the push client pushes objects to the server's Inbox (and pulls the default business card). It should be noted that it is not mandatory for the push server to support pulling of the default business card.

As any file transfer operation between two devices is potentially insecure (because of viruses, violation of privacy, etc.), the profile's specification suggests asking user's permission both on the client's and the server's side every time the object is pushed (pull).

A comprehensive description of the OPP can be found in [28] part K:11.

The Object Push Profile is especially interesting in connection with the topic of this thesis as it allows sending objects of arbitrary type from one Bluetooth to another. This means that the OPP can potentially be used to send MIDlet suites from one mobile device to another. The OPP-based provisioning methods are described in Sections 6.8 and 7.2 of this thesis.

File Transfer Profile

The File Transfer Profile can be viewed as a less restrictive form of the OPP as it allows both pushing and pulling of any objects. Though only two objects are supported (the file and the folder), any object can be packaged as a file and sent using the FP. The

client and server roles in the profile are defined here similarly to the GOEP. The client can perform the following operations on the server:

- Pull files and folders
- Push files and folders
- Browse folders
- Delete files and folders
- Create new files and folders.

The operations occur in response to client operations and include, e.g., sending of requested files and folders and resolving requests to create and delete objects. The folder browsing, pushing and pulling of files are mandatory to support for both the client and the server. Other features (creation and deletion of files and folders, transfer of the folder with all its files) are optional.

For the same reasons as the OPP, the FP assumes that the user permits or initiates the file transfer in this or that way. It is also assumed that a user interface is in place for folder browsing [26].

Further information on the File Transfer Profile can be found in [28] part K:12.

The FP also allows sending of files from one Bluetooth device to another and therefore is also a potential candidate for local provisioning of MIDlets.

Synchronization Profile

The last profile from the GOEP family is the Synchronization Profile. As it is clear from its name, the use case that the profile embodies is synchronization of objects on two devices. Synchronization process can be viewed as object transfer according to some rules; the SP relies on the Infra-Red Mobile Communications (IrMC) specification for definition of these rules, mostly concentrating on the procedures needed to initiate and control the synchronization process.

The SP is of little value for provisioning of MIDlets and, therefore, is not discussed here in details. Complete information about the profile can be found in [28] part K:13.

It is important to mention that the set of IrDA interoperability profiles described above was designed in such a way as to promote application level interoperability. In other words, they are to some degree bearer-independent and applications that implement these profiles can use both Infrared and Bluetooth links for their operation without major, if any, changes in the application logic.

5.6.2 New Bluetooth Profiles

The set of Bluetooth profiles in Figure 28 is by no means closed; many profiles did not appear in specification versions 1.0 and 1.1 just because they were not completed and the Bluetooth SIG wanted to accelerate publishing of the specifications. Here are some of the profiles that were published later as separate documents or are still being formally specified and expected to appear in Bluetooth specification version 2.0 (according to [26]): the Personal Area Networking Profile, the Printing Profile, the Still Image Profile, the Car Profile, the Local Positioning Profile.

Among these new profiles, the Personal Area Networking (PAN) Profile is of special interest for the subject of this thesis as it can be potentially used for provisioning of

MIDlets. The PAN Profile was released in the early 2003 and focuses on ad hoc IP networking issues. Ad hoc networks are networks that form spontaneously; Bluetooth piconets and scatternets are examples of such networks. The profile describes how two or more Bluetooth-enabled devices can form an ad hoc network. The PAN Profile-based provisioning scenario can be found in Section 6.7 of this thesis.

6. Local Provisioning from Autonomous Devices via Bluetooth Wireless Technology

Wide acceptance and adoption of the Bluetooth wireless technology creates several new provisioning use cases where MIDlet suites are transferred over Bluetooth link from an autonomous device. In this thesis, the term autonomous device refers to a Bluetooth-enabled device that can provide MIDlets autonomously, without the operator's control. To get a MIDlet suite from an autonomous device, the user needs only a mobile device with Bluetooth connectivity. An autonomous device is somewhat similar to an ATM: a person needs only a banking card to withdraw cash. The term autonomous does not imply that a device has no network connectivity; it rather stands for the fact that it can perform its function automatically (like the ATM in the above example). An example of autonomous device is a MIDlet dispenser, an automatic device that distributes MIDlets in a public place.

In general, depending on the user's role in the process, provisioning of MIDlets can be conventionally divided into two major categories: push provisioning and pull provisioning. In pull provisioning, it is the user of the mobile device who initiates the process, e.g., by discovering a suite using a WAP browser and starting to download. OTA provisioning (cf. Section 3.2) is an example of pull provisioning. In push provisioning, it is another entity (another user, provisioning server, autonomous device, etc.) that initiates deployment, e.g., by sending a MIDlet suite to the user's mobile device via Bluetooth connection. It should be noted that such division is quite relative, and the same provisioning method can sometimes fall into both categories. E.g., WAP Push provisioning (cf. Section 3.3) can be classified as pull provisioning when it is initiated by the user (cf. Section 3.3.3) or as push provisioning when it is used by a service provider to propose a Java application to some users.

This Chapter concentrates on local provisioning of MIDlets from autonomous devices. Section 1 describes some use cases where this type of delivery may be useful. Then several provisioning methods are proposed and examined. All of them are the author's own developments. Though some of the methods are based on existing standards, they are not adopted standards themselves. Sections 2-4 are dedicated to issues common to all provisioning methods discussed in this chapter, viz. role of JAD file in local provisioning, copyright issues and problem of additions to standard profiles. Sections 5 and 6 discuss pull scenarios based on the File Transfer Profile and the Provisioning service. Possible use of the PAN Profile for push and pull provisioning is described in Section 7. Section 8 explains how the Object Push Profile can be used for push delivery.

6.1 Use Cases

Several use cases where a MIDlet is obtained from some type of autonomous device are described in the next four Sections. The list is in no case exhaustive, careful examination will reveal a lot more situations where local provisioning may be useful.

6.1.1 Control over Appliance Using MIDlet

Consumer electronics is getting smarter and smarter. Some home appliances such as refrigerators and TV-sets are already capable of connecting to the Internet, thus opening

possibilities for their remote control. On the other hand, Bluetooth components are getting smaller and cheaper, which facilitates their integration in all kind of electric appliances. Potentially, Bluetooth-enabled appliances could be controlled locally over the Bluetooth link. A mobile device (typically a mobile phone) is a perfect candidate to exercise both types of control as it is always with the user. Several technologies can be used in a mobile device for this purpose.

One option is to use a WAP browser to interact with the appliance. Either normal WAP connection (the appliance is connected to the Interned) or WAP over Bluetooth (the appliance is Bluetooth-enabled) can be used. [33] gives an in-depth analysis of how WAP over Bluetooth link may be used for control over and communication with the appliance.

Another solution is to use a MIDlet for communication with the gadget. In this scenario, the HTTP (supported both in MIDP 1.0 and 2.0) can be used for remote control and Java APIs for Bluetooth (JSR-82, an optional package for MIDP 1.0 and 2.0) for local control. To enable the latter, both devices must support the Bluetooth wireless technology. Typically, every appliance model will require a different control MIDlet.

To start using the mobile device for communication with the appliance, the user needs to install an appropriate control MIDlet. This can be done in the shop where the appliance is bought (using any of the local provisioning schemes described in Sections 3.4-3.5) or the MIDlet can be downloaded using OTA provisioning (cf. Section 3.2) from the manufacturer's site in the Internet. Nevertheless, if both the appliance and the mobile device support the Bluetooth technology, it is very advantageous to enable local provisioning of the control MIDlet directly from the gadget. The main benefit of such approach is that the user can always get a control application directly from the appliance, and that quickly and for free. Figure 29 illustrates the described use case.

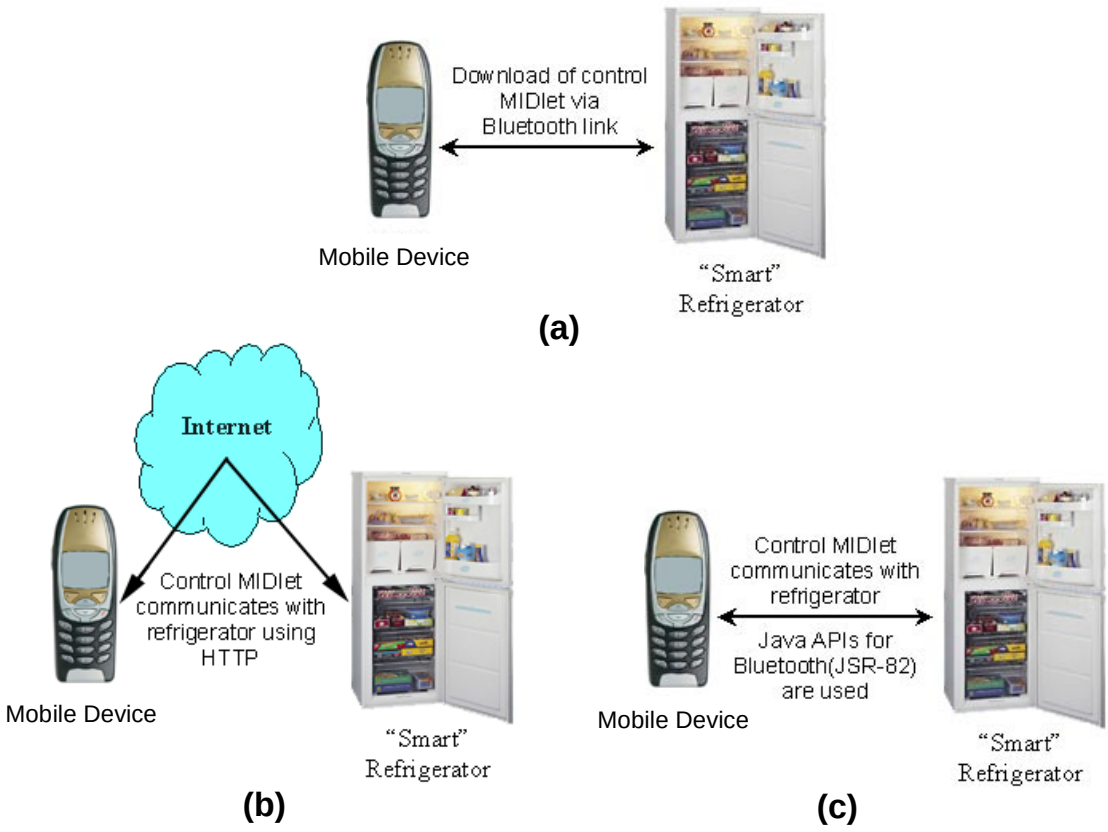


Figure 29. Wireless control over an electric appliance using a MIDlet

Note that control methods displayed in Figure 29 (b) and (c) are not alternatives. The same control MIDlet may use both modes of communications: Bluetooth Java APIs when the user is near the appliance, and HTTP when the devices are out of the Bluetooth range.

6.1.2 MIDlet as Museum Guide

A museum can provide visitors with a MIDlet that serves as an interactive guide. Such MIDlet may contain, e.g., a map of the museum and information about pieces of art. It can also provide an individual voice excursion by connecting to Bluetooth access points in each room of the museum and downloading recordings from them.

The MIDlet in such scenario is downloaded by visitors (or pushed to their mobile devices) from some autonomous MIDlet dispenser near the entrance to the museum.

6.1.3 MIDlet is Used to Query Bus and Railway Timetables

MIDlets can be used to query bus and railway timetables. A MIDlet can either store the whole timetable on the device (if it is small) or just provide a user interface that will allow querying a remote database. The latter has an advantage that the MIDlet does not need to be updated when timetables change. On the other hand, the service that the MIDlet provides is not free from the user's point of view as Over-the-Air connection (to the database) is chargeable.

Such MIDlets may be distributed on bus stops and at railway stations using autonomous MIDlet dispensers.

6.1.4 MIDlet as Advertisement

A shopping centre may install a MIDlet dispenser near the entrance to push MIDlets with advertising information to visitors and passers-by. Such MIDlets may contain information about outlets that can be found in the centre, latest special offers, plan of the centre, etc.

6.2 Role of an Application Descriptor in Local Provisioning

In general, the MIDP specification does not require to provide an Application Descriptor for every MIDlet suite. A JAD file is mostly intended for OTA provisioning, where it allows saving the user's money by rejecting unsuitable Java applications without downloading a JAR file. In local provisioning, a JAD file loses its role as local downloading does not involve any costs and a JAR file is often downloaded to a mobile device before the installation starts.

However, Application Descriptors play an important role in MIDP 2.0 security framework, as they contain digital signatures of signed MIDlet suites. Without a JAD file, a signed suite will be installed as untrusted, which may lead to loss of access to some sensitive APIs available to trusted MIDlets only. Hence, in a more general case, both JAD and JAR files are needed, and all provisioning scenarios discussed in this Chapter support delivery of both files.

6.3 Copyright Issues

Chapter 4 has already stated that all MIDlet suites can essentially be divided into two broad categories: those that can be freely distributed and those for which free distribution is undesirable or even inflicts direct financial loses. For MIDlets that fall into the second category, local provisioning is a much more difficult task and usually requires additional protection mechanisms (like OMA DRM described in Chapter 4). Provisioning methods described in this Chapter are primary intended for MIDlet suites that can be freely distributed. However, most of the methods can be adjusted for distribution of suites from the second category (e.g., by implementing the OMA DRM superdistribution). These more secure counterparts are not described to make the discussion simpler; only short remarks are made on how a particular scenario can be made more secure.

6.4 Problem of Additions to Existing Bluetooth Profiles

Before starting to discuss various local provisioning scenarios, an important remark should be made. All but one methods proposed in this Chapter are based on standard Bluetooth SIG profiles. However, these profiles are not intended for quite a specific task of provisioning of MIDlets. Their scopes are wider. Because of that there is always a dilemma: either to extend the profile to adjust it for provisioning or to use the standard Bluetooth SIG profile. Extension of a profile usually means that additional UUID and attributes are added to its service record. The purpose of these additions is to make it possible to learn (by means of SDP) whether a profile can be used for provisioning of MIDlets. If a profile is used without any additions, then a profile-level Bluetooth connection has to be established to learn if provisioning is supported. Additions to the service record allow getting this information earlier, during Bluetooth Service Discovery, which leads to a better user experience. However, some changes may have other motivation, e.g., to declare the model of the mobile device, etc.

Thus, the advantage of an unmodified profile is that it is totally standard; while the benefit of a profile with modified service record is that it is better adjusted for provisioning. In this thesis, the problem is solved in favor of additions to profile service records. Nevertheless, most of suggested provisioning scenarios can use profiles both with modified and unmodified service records.

6.5 Pull Provisioning Using File Transfer Profile

This Section proposes a provisioning scenario, which is based on the Bluetooth SIG File Transfer Profile (FP) and allows downloading a MIDlet (or MIDlets) from an autonomous device via OBEX protocol over the Bluetooth link. The FP supports pulling files from the OBEX server, which is exactly the feature needed to download JAD and JAR files to a mobile device. As the FP is a general-purpose profile, its presence does not guarantee that the device provides MIDlets. To make it possible for mobile devices to determine (by means of Bluetooth SDP) that MIDlets are available through the FP, a special UUID has to be added to the profile's service record. Now mobile devices will search for the pair of UUIDs – FP UUID and provisioning UUID. Presence of both in the same service record means that MIDlets can be pulled from a given device using the FP.

To support the proposed provisioning method, a mobile device and an autonomous device have to meet the following requirements.

Requirements for Autonomous Device that Provides MIDlets

- An autonomous device must support the role of Server as defined in the File Transfer Profile specification [28] part K:12
- A special UUID associated with provisioning of MIDlets has to be inserted into the profile's service record
- An autonomous device must support the role of Remote Device as defined in the Service Discovery Application Profile (SDAP) [28] part K:2.

Requirements for Mobile Device that Downloads MIDlets

- A mobile device must support the role of Client as defined in the File Transfer Profile specification [28] part K:12
- A mobile device must support the role of Local Device as defined in the Service Discovery Application Profile (SDAP) [28] part K:2.

6.5.1 Description of the Method

In essence, the method works as follows: the user of the mobile device initiates the provisioning process, Bluetooth inquiry and Service Discovery procedures are used to locate the autonomous device, then the FP connection is established. The catalog with information about available MIDlet suites is downloaded using OBEX protocol and the user selects a desired suite. The suite is downloaded (both JAD and JAR files) and installed by the AMS.

Figure 30 illustrates the proposed provisioning scenario.

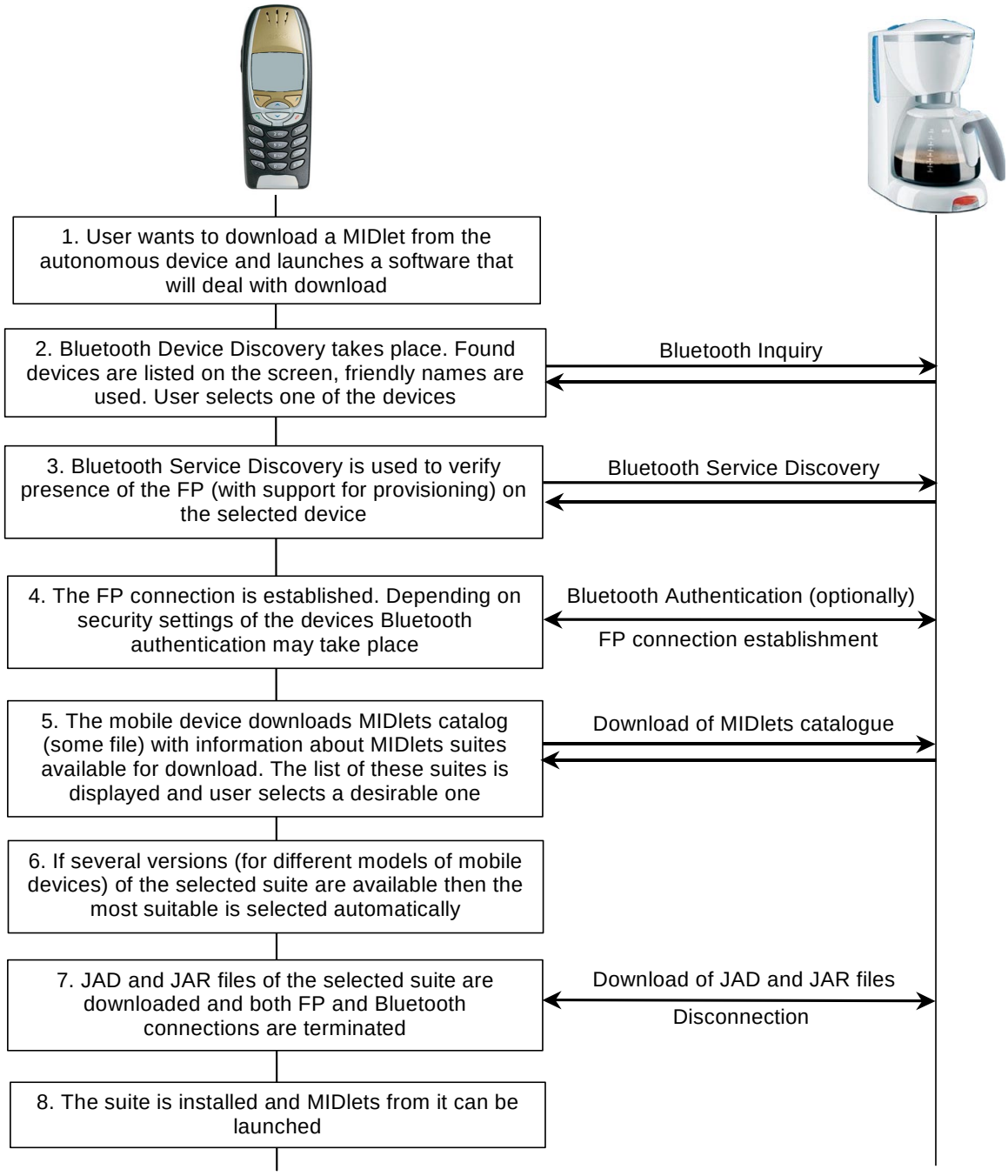


Figure 30. Provisioning from an autonomous device using the File Transfer Profile

Now the scenario will be discussed in more details. The steps below correspond to Figure 30.

1. The user of the mobile device somehow obtains information that a MIDlet (MIDlets) can be downloaded from an autonomous device in the vicinity. E.g., this information may come from the user manual of "smart" appliance or from the poster near a MIDlet dispenser.
   The user decides to get a MIDlet and starts a special application that deals with provisioning via the FP. It can be an autonomous application, or it can be integrated

with the AMS. This application is referred to as provisioning application later on in this Section.

2. The provisioning application executes the Bluetooth inquiry procedure. Found devices are presented to the user, and friendly names are used. Devices are filtered by their class of device field, only those that have Object Transfer bit set to 1 are displayed (further information on the CoD field can be found in [29]). A friendly name usually allows easy identification of the desired autonomous device and the user selects it from the list.
3. The Service Discovery session with the selected device is opened. The File Transfer Profile with support for provisioning of MIDlets is searched using an SDP request with the FP UUID and the provisioning UUID. Presence of both UUIDs in the same service record means that the selected device provides MIDlets, and FP connection parameters are retrieved.
   If the FP was not found on the device, or if its service record does not contain provisioning UUID, the installation stops, and an appropriate message is shown to the user.
4. The provisioning application tries to establish a FP connection with the autonomous device. Depending on security settings of the devices, the Bluetooth authentication may be needed (cf. Section 5.5.1). E.g., an electric appliance will probably request authentication, while a MIDlet dispenser probably will not. As the autonomous device most likely has a fixed PIN, the same PIN must be entered on the mobile device. For electric appliance, it can be found, e.g., on the sticker on the packaging.
   Finally, the File Transfer Profile connection is established either with or without authentication.
5. The provisioning application downloads a special file that contains information about all MIDlet suites available for downloading. The name of the file can be, e.g., “midlets.xml”. This catalog is parsed and available suites are displayed. The user selects a desired one.
6. Several versions of the same suite (intended for different models of mobile devices) can be available on the autonomous device. If so, this fact is reflected in the MIDlet catalog downloaded in the previous step. Based on the information from the catalog, the provisioning application selects the version that gives the best fit for the user’s mobile device.
   Eventually, the provisioning application knows the exact names of JAD and JAR files that have to be downloaded from the autonomous device.
7. JAD and JAR files of the MIDlet suite selected in the previous step are downloaded. Names of the files come from the MIDlet catalog. When downloading is completed, both FP and Bluetooth connections are terminated.
8. The AMS installs the new suite using the JAD and JAR pair. To accomplish this, the AMS must be able to install suites whose JAD and JAR files have resided in the file system of the mobile device (cf. Sections 3.5 and 3.6 of this thesis for situations when the same behavior is required from the AMS).
   If installation was successful, MIDlets from the suite can be immediately launched.

The description above is intended to give just a general idea of the method. Almost all fine details have been omitted from the discussion. Thus, e.g., the user interface and architecture of the provisioning application are not specified as they are hugely device-specific. The internal structure of the MIDlet catalog is also left undefined as there are

several ways to describe available MIDlets, e.g., an XML notation can be used. Copyright issues are not discussed, but in general the scenario is intended for provisioning of MIDlets that can be freely distributed. Its adaptation for provisioning of suites that should not be distributed without control requires additional consideration.

The method needs to be defined more precisely before it can be implemented, but the general sequence of events is as described in this Section.

6.5.2 Advantages and Disadvantages of the Method

Advantages:

- The method can be easily standardized. Whichever approach is chosen for provisioning from autonomous devices, it should be standard and widely accepted. Thereupon, the main advantage of the FP-based method is quite moderate standardization effort required as it is based on the standard File Transfer Profile defined by the Bluetooth SIG
- Simplicity, as the method is based on the FP, which can be considered as a rather simple profile
- The method is bearer-independent. Indeed, only that part of the algorithm that deals with the FP connection establishment is Bluetooth specific; the other part just uses a subset of OBEX protocol and therefore, can work over the TCP/IP, the Infrared link, etc. This may be useful in some cases.

Disadvantages:

- Insufficient security. Openness of the method can be regarded as a disadvantage. For the reason that the File Transfer Profile (as any other Bluetooth SIG profile) was created with interoperability in mind, it may be possible to download MIDlets using not only mobile devices but also any other devices that support the FP. E.g., a Bluetooth-enabled laptop can be used for this purpose. This is a clear disadvantage if free distribution of proposed MIDlet suites is undesirable. To make the method more secure, an additional protection mechanism is required. One option is to use the OMA DRM separate delivery
- Redundancy of supported features. The FP, even in its minimum configuration, contains features that are not needed for provisioning of MIDlets. E.g., ability to push files to the OBEX server is a mandatory requirement, which means that it must be implemented (at least formally) on the autonomous device even if it is going to be used
- The mobile device is responsible for selection of the appropriate version of a MIDlet suite (when versions for different mobile device models are available). As a result, the software that deals with the download has to be more complex. In general, it is better when the autonomous device selects a version like in OTA provisioning, where the provisioning server makes the choice.

6.6 Pull Provisioning Using Provisioning Service

The scenario described in previous Section uses the File Transfer Profile, which is intended for a generic file transfer and contains features redundant for the provisioning use case. Another option is to define a new GOEP-based service that will be specifically tailored for pull provisioning from autonomous devices. Such service will only allow pulling of files from an OBEX server (an autonomous device) and will contain all information about available MIDlets in its service record (instead of a separate file in the FP-based scenario). This new service is referred to as Provisioning service (PS) in this thesis.

The specification of the Provisioning service can be created using the specification of any GOEP-based profile as a pattern. In fact, any of these specifications just defines the use cases covered by the profile and lists operations and headers of OBEX protocol that are in use. For the Provisioning service, the only use case is pulling of files from the server. The list of OBEX operations and headers must contain only those that are needed to enable the pull feature. In addition, the specification must define how MIDlet suites available for downloading are described in the profile service record. Creation of such specification should not involve major difficulties; it is assumed later in this Section that the specification exists and contains description of all features discussed above.

To support the proposed provisioning scenario, a mobile device and an autonomous device must meet the following requirements:

Requirements for an Autonomous Device that Provides MIDlet Suites

- An autonomous device must support the role of Server as defined in the Provisioning service specification (hypothetic specification)
- An autonomous device must support the role of Remote Device as defined in the Service Discovery Application Profile (SDAP) [28] part K:2.

Requirements for a Mobile Device that Receives MIDlets

- A mobile device must support the role of Client as defined in the Provisioning service specification (hypothetic specification)
- A mobile device must support the role of Local Device as defined in the Service Discovery Application Profile (SDAP) [28] part K:2.

6.6.1 Description of the Method

In general, the scenario is similar to the FP-based provisioning. The user of the mobile device starts the provisioning process, Bluetooth Device and Service Discoveries are used to select a device that provides MIDlets via the Provisioning service. The information about available MIDlets is fetched from the PS service record on the autonomous device. This is different from the FP-based method, where this information is taken from a downloaded MIDlet catalog. After that, the user selects a desired MIDlet. The PS connection is established between the devices, and the selected suite is downloaded (both JAD and JAR files). The AMS installs the suite.

The Provisioning service-based scenario is depicted in Figure 31.

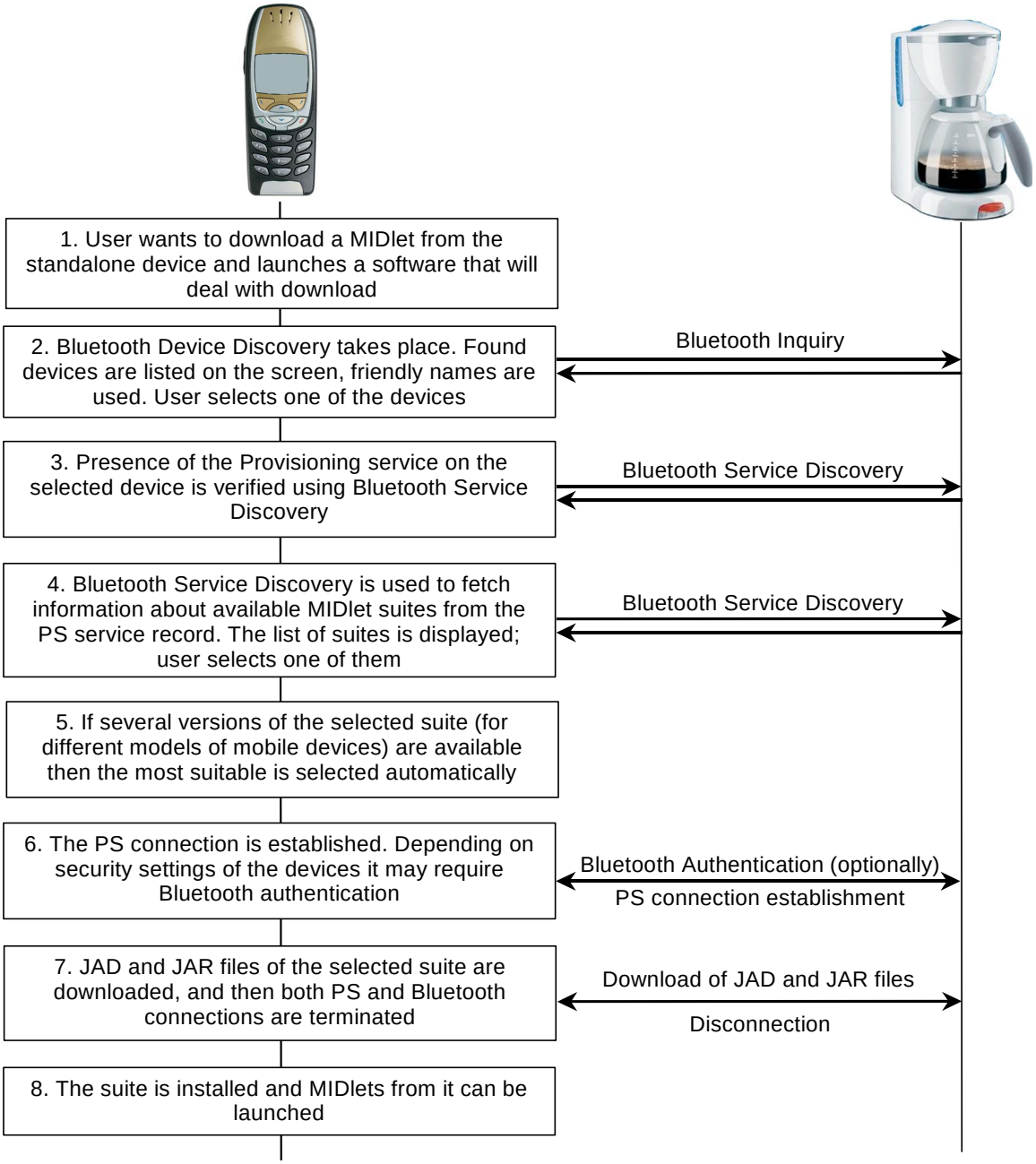


Figure 31. Provisioning from an autonomous device via the Provisioning Service

The main difference between the FP-based scenario and the scenario depicted in Figure 31 that in the former, the information about available MIDlets is in a special file that needs to be downloaded first; while in the latter, this information is retrieved using Bluetooth Service Discovery. Now a brief step-by-step description of the PS-based scenario will be given.

1. – 2. Steps are the same as in the FP-based provisioning
3. The Provisioning service is searched on the selected device using Bluetooth Service Discovery. The SDP request contains only one UUID – the UUID assigned to PS. If the service is found, the device provides MIDlets and connection parameters are retrieved. Otherwise the process stops

4. Bluetooth Service Discovery is used to fetch the information about available MIDlet suites from the PS service record. Suites are displayed, and the user selects one of them
5. If several versions of the suite are available, the one suitable for the user's mobile device is selected. The selection is automatic, based on the information fetched in the previous step
6. The PS connection between the mobile device and the autonomous device is established. This may involve Bluetooth authentication (depends on security settings of the devices)
7. - 8. These steps are the same as in the FP-based provisioning.

It should once again be noticed that the Provisioning service was invented by the author during the work on this thesis. The Bluetooth SIG has nothing to do with this service. Moreover, its specification does not exist although it can be created without essential difficulties.

As in the case with FP-based provisioning, only the general idea of the method is given, and a lot of details have been omitted to make the discussion simpler. E.g., it is not specified how the information about available MIDlets is formatted in the service record or how the most suitable version of the suite is selected. All this has to be clarified, if the method will be standardized.

6.6.2 Advantages and Disadvantages of the Method

Advantages:

- The method is based on the Provisioning service that is highly tailored for downloading of MIDlets from autonomous devices and does not contain any unnecessary features
- The method is more secure compared with the FP-based scenario. Unlike the latter, it is not possible to use, e.g., a laptop that supports the standard File Transfer Profile to fetch MIDlet suites. However, as the specification of the Provisioning service is supposed to be public, it is not difficult to create some special software and get hold of suites.

Disadvantages:

- Requires significant standardization effort. To be successful, the method must be accepted as an industry standard, which involves adoption of now-hypothetic Provisioning service as a standard way of provisioning from autonomous devices. This seems problematic, or at least requires some effort
- Insufficient security. It is not possible to protect MIDlets from being downloaded to devices different from target mobile devices. E.g., a Bluetooth-enabled laptop with software created using the specification of the Provisioning service can be used to get MIDlets from an autonomous device. Later on, the MIDlets can be illegally distributed. Similarly to the FP-based provisioning, the method can provide better security if used in conjunction with OMA DRM separate delivery
- Selection of the appropriate version of a MIDlet suite (when several versions for various mobile device models are available) happens on a mobile device, which makes the mobile device provisioning software more complex. This disadvantage is the same as in the FP-based provisioning.

6.7 Provisioning Using PAN Profile

The Personal Area Networking Profile allows formation of ad hoc IP networks using the Bluetooth wireless technology and therefore can be used for local provisioning of MIDlets. The idea is as follows. A mobile device and an autonomous device form an ad hoc network, after which the process goes like in ordinary OTA provisioning with the difference that a local IP connection over Bluetooth is used instead of an Over-The-Air connection. The provisioning server (like the one used in OTA provisioning) can be located either on the autonomous device itself, or, if the device has network connectivity, somewhere in the network. From the point of view of the Discovery Application (for the DA cf. Section 3.2.1), location of the provisioning server makes no matter since the communication goes over IP. Therefore, changes that should be introduced into mobile device software that deals with OTA provisioning are minimal (if any). Depending on how the PAN connection is established the method can be attributed to either push or pull provisioning.

The PAN Profile specification [34] defines the following roles for devices:

- **Network Access Point (NAP) and NAP service:** A Bluetooth device that provides some features of Ethernet bridge to support network services. The device has an additional connection to some other network
- **Group Ad Hoc Network (GN) and GN service:** A Bluetooth device that is able to forward Ethernet packets to each of the connected Bluetooth devices (PAN users). Group Ad Hoc Network does not provide access to any additional networks
- **PAN User (PANU) and PANU service:** A Bluetooth device that uses either the NAP or the GN service. Direct PANU to PANU communication is also supported.

An SDP service record is defined for each of the services above. To facilitate provisioning of MIDlets, some UUIDs have to be added to each of them.

6.7.1 Additions to Service Records of the PAN Profile

Depending on its purpose and capabilities, an autonomous device can assume any role defined in the PAN Profile. The role of NAP is suitable for an autonomous device that provides access to one or several provisioning servers in the network; an example of such device is a MIDlet dispenser with network connectivity. The role of GN fits an autonomous device with the provisioning server onboard; an example is a MIDlet dispenser without network connectivity. The role of PANU is for an autonomous device that has a built-in provisioning server but is too resource-constrained to support the GN service, e.g., a coffee machine. Whichever of the PAN Profile roles an autonomous device takes up, a UUID associated with provisioning of MIDlets has to be added to an SDP service record of the appropriate service (NAP, GN or PANU). Presence of this UUID will signify that MIDlets are available through the service.

To download MIDlets using any of PAN Profile's services, a mobile device needs to support the role of PAN User. A special UUID has to be added to the PANU service record, which will mean that a mobile device supports downloading of MIDlets via the PAN Profile. Presence of this UUID will facilitate push provisioning, as autonomous devices will be able to discover mobile devices that support PAN Profile-based provisioning.

There may be a need to append additional attributes to service records of PAN Profile's services. Such cases are discussed in the next Section.

6.7.2 Autonomous Device Has a Built-in Provisioning Server

To provide access to the provisioning server that resides on the autonomous device, the latter must support either the Group Ad Hoc Network service or the PANU service. To use such provisioning server, the mobile device must support the role of PAN User. Depending on the entity that initiated the PAN Profile connection, the method can be regarded as push or pull provisioning. Figure 32 illustrates pull provisioning from the autonomous device using the PAN Profile.

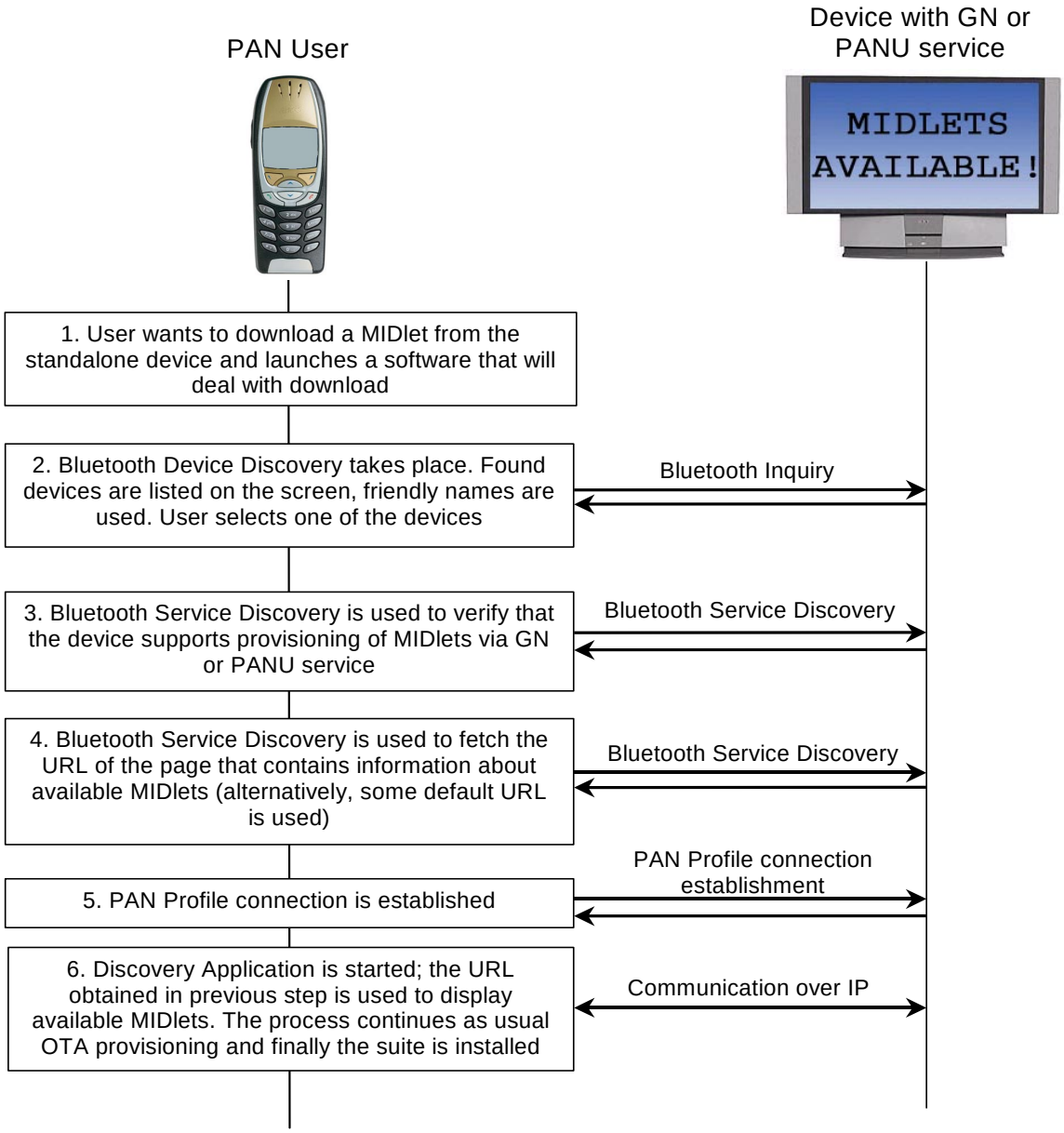


Figure 32. Provisioning from an autonomous device using the PAN Profile. Pull variant, the provisioning server is on the autonomous device

Here is a brief description of the provisioning method depicted in Figure 32.

1. As in all pull provisioning scenarios in this Chapter, the user somehow receives the information that MIDlets can be downloaded from the autonomous device and

launches an application that will deal with PAN connection establishment (possibly the AMS). This application is called provisioning application later on in this Section

2. The provisioning application executes the Bluetooth inquiry procedure. Found devices are presented to the user with their friendly names. The class of device field can be used to filter devices. In this case, only those that have Networking bit set to 1 are displayed. The user selects the desired device from the list
3. The provisioning application opens the Bluetooth Service Discovery session with the selected device. The UUID associated with provisioning of MIDlets via PAN Profile is searched. If the UUID is found, the connection parameters are retrieved, otherwise the process stops
4. To get information about available MIDlets, the mobile device needs to know the URL of the provisioning server. One option is to place this URL into an attribute in the GN or PANU service record on the autonomous device. In this case, Bluetooth Service Discovery is used to retrieve the URL. Another option is to define some default URL for the provisioning server
5. A PAN Profile connection between the mobile device and the autonomous device is established. This procedure is described in [34], and may involve Bluetooth or higher-level authentication and encryption.
   Eventually, the PAN Profile connection is established, and from now on devices communicate over IP
6. The Discovery Application is started on the mobile device. It uses the URL of provisioning server (obtained in step four) to download a page with information about available MIDlet suites (it is assumed that the DA is a browser). The process continues as in usual OTA provisioning (cf. Section 3.2). The fact that IP communication goes over the Bluetooth link and not Over-The-Air is transparent for the DA and the AMS.
   The user selects a suite; it is downloaded and installed according to Section 3.2.

The method presented above also has a push variant. In this case, it is the autonomous device that takes the initiative of establishing a PAN Profile connection with the mobile device. After that, the DA application is used to select the suite, as in the pull variant. Figure 33 demonstrates push provisioning using the PAN Profile.

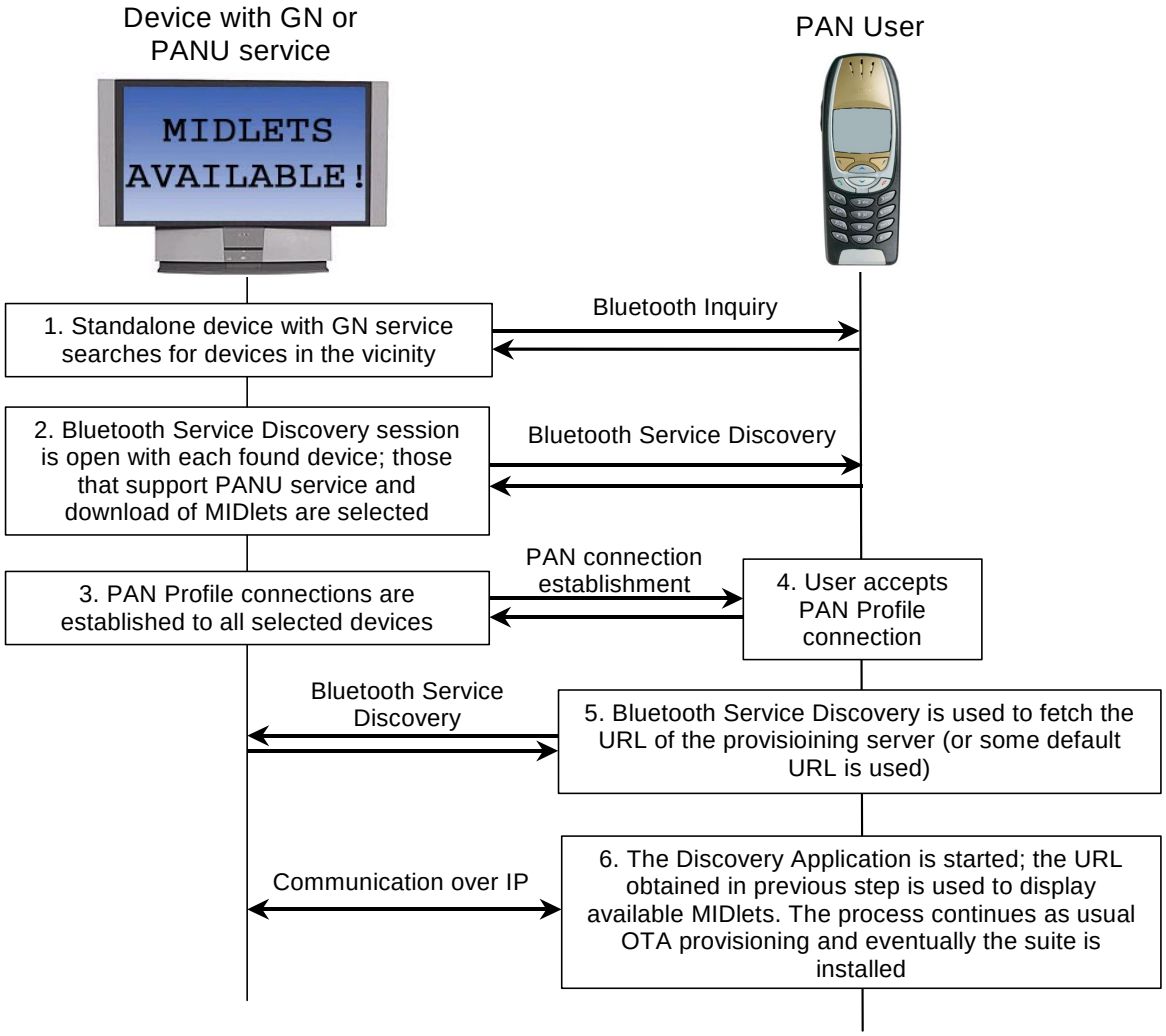


Figure 33. Provisioning from an autonomous device using the PAN Profile. Push variant, the provisioning server is on the autonomous device

Here is a brief description of the scenario in Figure 33.

1. An autonomous device located in some public place periodically executes the Bluetooth inquiry procedure. Found devices are filtered using their Bluetooth addresses, and only those that were not present during previous inquiries are chosen. These devices are filtered again, this time using their class of device field. Those that have Networking bit set to 1 are selected for further processing
2. The Bluetooth Service Discovery session is opened with each device picked out in the previous step. The UUID associated with ability to download MIDlets via the PAN Profile is searched (this UUID has to reside in the PANU service record). Devices that have the needed UUID are selected for further processing
3. The autonomous device tries to establish PAN Profile connections to all devices selected in the previous step. Depending on security settings of devices, this may involve Bluetooth or higher-level authentication and encryption
4. The user of the mobile device gets a notification about a connection attempt. It will be helpful if the notification tells the user that MIDlets can be downloaded as a result of connection acceptance. However, it may be technically impossible to determine it as this stage.

The user accepts the PAN Profile connection

5. The mobile device uses Service Discovery to understand whether the just connected autonomous device supports provisioning of MIDlets. The UUID associated with provisioning is searched, this UUID have to reside in the GN service record (or in the PANU service record). If provisioning is supported, the mobile device needs to learn the URL of the built-in provisioning server. Similarly to the pull variant, the URL can be found in the GN (or PANU service record) or some default URL can be used
6. This step is the same as in the pull variant.

In general, the push variant of the method is more complex, but it can be useful those cases, where "active" distribution of MIDlets is needed.

6.7.3 Provisioning Server in Resides the Network

If the autonomous device has network connectivity, it can provide access to one or more provisioning servers in the network. Mobile devices in the vicinity can download MIDlets suites from these servers. To use the proposed method, a mobile device must support the PANU service. The autonomous device must support the Network Access Point service. The scenario has both pull and push variants, which are similar to those depicted in Figures 32 and 33 respectively. The only distinctions are: NAP service is used instead of GN (or PANU) service and the provisioning server does not reside on the autonomous device. The latter is transparent for the mobile device as it just uses a URL and does not care where the server actually resides.

The benefit of networked autonomous device is in a possibility to have one provisioning server and several autonomous devices that are used as access points to this server. In this case, it becomes easier to change/update available MIDlet suites, as changes have to be implemented only on the provisioning server.

The provisioning scenarios described in this and the previous Sections are intended to give just a rough notion of how the PAN Profile can be used for delivery of MIDlet suites. A lot of details have been left unspecified or omitted for the sake of discussion simplicity. E.g., possible PAN Profile security models are not described (for their detailed description cf. [34]). Other important issue is copyright protection. In the proposed solution, it is assumed that distribute suites are totally free and therefore no protection is needed. However, the PAN Profile can also be used for distribution of MIDlets that should not circulate freely; in this case, an additional protection mechanism method can be integrated into the provisioning scenarios. One option is to use the OMA DRM separate delivery.

6.7.4 Advantages and Disadvantages of PAN Profile-Based Provisioning

Advantages:

- The proposed methods can be easily standardized as they are based on the standard Bluetooth SIG PAN Profile and MIDP 2.0 OTA provisioning
- The solution is quite flexible, allowing both pull and push provisioning; a provisioning server can be local or can reside in the network.

Disadvantages:

- Some resource-constrained devices, such as coffee machines, would not be able to use the proposed scenarios, because the PAN Profile is quite demanding compared with the File Transfer Profile or the Provisioning Profile. In addition, some cases require implementation of a provisioning server as well. So, the solution is not universal, at least in the foreseeable future
- For MIDlets that cannot be freely distributed, an additional protection mechanism is needed, e.g., the OMA DRM. The solution is intended primarily for those MIDlets that can be freely distributed.

6.8 Push Provisioning Using Object Push Profile

The PAN Profile-based provisioning of MIDlets includes push scenarios where the autonomous device initiates the process. One of these scenarios is depicted in Figure 33. Push provisioning can also be carried out using the more simple Object Push Profile (OPP). The purpose of such delivery method will be the same – "active" distribution of MIDlets in some public place (an airport, a museum, a conference, etc.). The essence of the method is as follows: the autonomous device pushes a MIDlet suite to the Inbox of mobile devices in the vicinity, and the user installs the received suite. Similarly to the push provisioning via PAN Profile, the autonomous device uses Bluetooth SDP to figure out whether a given mobile device is capable of receiving and installing MIDlet suites. To make it possible, some additions have to be made to the OPP service record on a mobile device.

6.8.1 Additions to OPP Service Record

The OPP service record includes an attribute called *Supported Formats List*. The value of this attribute is a list of all object formats that can be pushed to the device. The OPP specification ([28] part K:11) defines the following formats:

0x01 = vCard 2.1
0x02 = vCard 3.0
0x03 = vCal 1.0
0x04 =iCal 2.0
0x05 = vNote
0x06 = vMessage
0xFF = any type of object.

Any implementation of the OPP picks out formats it wants to support and places them to its *Supported Formats List* attribute. If the device accepts all types of objects, then it just places 0xFF.

For push provisioning via the OPP, two new formats should be defined:

0x07 = Application Descriptor (JAD file)
0x08 = Java Archive (JAR file).

Presence of these formats in the *Supported Formats List* attribute of the OPP service record will tell the autonomous device that the mobile device is able to receive MIDlets via the OPP and install them. It is important to note that even if the mobile device accepts any

types of objects (*0xFF*), it still needs to add JAD and JAR formats (*0x07* and *0x08*) to the attribute. This is needed because of devices that accept all types of objects but do not support the MIDP. Such devices will be able to receive a suite (JAD and JAR files) but will fail to install it. Therefore, to indicate support for push provisioning via the OPP, a mobile device needs to add *0x07* and *0x08* to its *Supported Formats List* attribute of the OPP service record.

Now requirements for autonomous devices and mobile devices will be summarized.

6.8.2 Requirements for an Autonomous Device that Pushes MIDlets

- An autonomous device must support the role of Push Client as defined in the Object Push Profile specification [28] part K:11
- An autonomous device must support the role of Local Device as defined in the Service Discovery Application Profile (SDAP) [28] part K:2.

6.8.3 Requirements for a Mobile Device that Receives MIDlets

- A mobile device must support the role of Push Sever as defined in the Object Push Profile specification [28] part K:11
- Application Descriptor and Java Archive (*0x07* and *0x08*) formats must be added to the *Supported Formats List* attribute of the OPP service record
- A mobile device must support the role of Remote Device as defined in the Service Discovery Application Profile (SDAP) [28] part K:2.

6.8.4 Description of the Method

Figure 34 gives an idea of the proposed method.

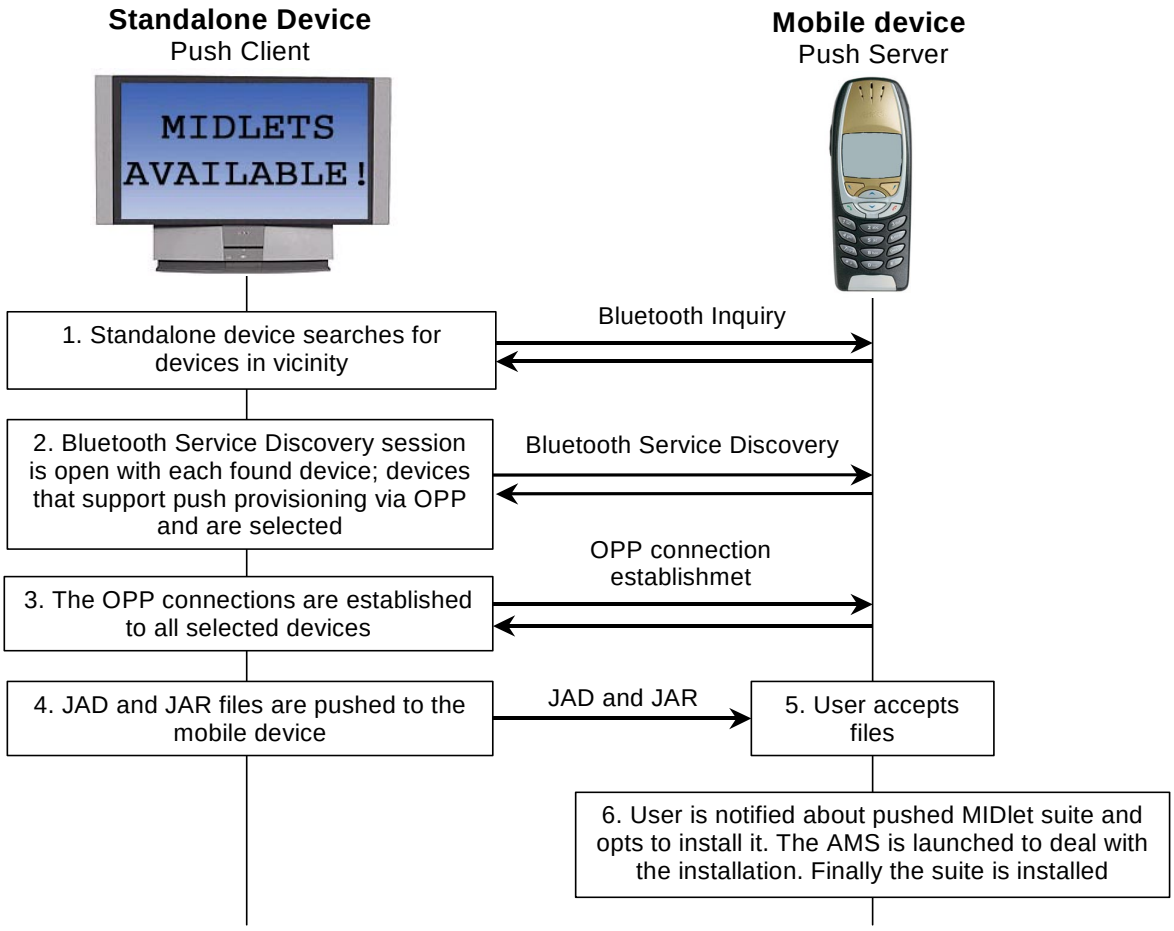


Figure 34. Push provisioning via the Object Push Profile

Now a brief step-by-step description of the method depicted in Figure 34 will be given.

1. An autonomous device located in some public place periodically executes the Bluetooth inquiry procedure. Found devices are filtered using their Bluetooth addresses, and only those that were not present during previous inquiries are chosen. These devices are filtered again, this time using their class of device field. Those that have Object Transfer bit set to 1 are selected for further processing
2. The Bluetooth Service Discovery session is open with each device chosen in the previous step. Presence of the Object Push Profile is checked up. The service record of the profile must indicate support for JAD and JAR file formats (cf. Section 6.8.1). Devices that do not meet these requirements are excluded from the process
3. The autonomous device tries to establish Object Push Profile connections to all devices selected in the previous step. The user of the mobile device may need to accept the OPP connection. Depending on security settings of devices, the process may involve Bluetooth authentication and encryption
   If Bluetooth authentication is needed (which is unlikely), the user is prompted to enter a PIN code. As the autonomous device is most likely to have a fixed PIN, the user needs to obtain it somehow. E.g., it can be in a highly visible announcement near the MIDlet dispenser
4. The autonomous device pushes JAD and JAR files of the MIDlet suite. If there are several MIDlet suites, they all are sent.

It is important to bear in mind that the OPP can be implemented so that the user needs to accept receipt of each object. Therefore, if JAD and JAR files are sent separately, the user may be confused, as pushing of two objects results in one Java application. One way to improve user experience is to send both files using the same PUT OBEX command. However, some OPP implementation may not support this feature. Another option is to pack JAD and JAR files into some object and push them as a single entity. In this case, every accepted object results in a new MIDlet suite. The software of the mobile device must be aware of the composite object's format and present it to the user as a MIDlet suite that needs to be installed. The AMS must also be able to install suites from such objects

5. The user accepts all objects pushed. The previous step showed that it is highly desirable to arrange the sending process so that each accepted object will result in a new MIDlet suite
6. When pushing is complete, the user is notified about a new MIDlet suite that needs to be installed. The new suite is visible in the Inbox, and the fact that it is comprised of JAD and JAR pair has been hidden. The user selects to install the suite, and the AMS is launched to deal with the installation. Eventually, the new MIDlet suite is installed.

6.8.5 Advantages and Disadvantages of the Method

Advantages:

- Can be easily standardized as it is based on the standard Object Push Profile
- Relatively simple, compared with the push provisioning via the PAN Profile
- Can be easily implemented as the OPP is already supported on all Bluetooth-enabled mobile devices; additions to the OPP and other changes (e.g., in the AMS) should not present major difficulties.

Disadvantages:

- Pushing of MIDlet suites according to this Section is somewhat "blind", since the autonomous device cannot recognize the model of the mobile device. The autonomous device only knows that the mobile device supports provisioning of MIDlets using the OPP. This is unimportant when pushing MIDlet suites that do not rely on device-specific functions; but for pushing of device-specific suites, information about the model of the mobile device is needed. One solution is to include this information into the OPP service record on the mobile device. In this case, the autonomous device can access it and select an appropriate version of the suite for a given mobile device
- In this method, additional efforts have to be made to hide the fact that a MIDlet suite is comprised of JAD and JAR files
- A MIDlet suite (or MIDlet suites) that are pushed using the method are pre-defined, the user have no case to select a desired suite
- The method is intended for delivery of free MIDlet suites and does not provide adequate protection for suites that should not be distributed freely. It cannot cope with the situation when some suite will be mistakenly pushed to a Bluetooth-enabled laptop and then illegally distributed. This problem does not exist if the method is used for the OMA DRM superdistribution as described in Section 7.2.

7. Forwarding of MIDlets from Mobile Devices

Until very recently, it was no standard way to forward MIDlets from one mobile device to another. This feature looks quite useful and no support for it seems strange at the first glance. However, a more thorough examination will reveal the reason: there was no reliable mechanism to distinguish between commercial and free MIDlet suites. It has already been mentioned that this thesis uses the term commercial suite to refer to any MIDlet suite that should not be distributed without any control. Therefore, it was not possible to determine whether a given MIDlet can be forwarded from a mobile device without infringement of someone's copyrights. Neither the MIDP specification, nor other related specifications make any provisions on how MIDlets can be forwarded safely. At the same time, the problem is very subtle as it involves software developed by third-party vendors and any violation can cause financial losses. As a result, Java-enabled mobile devices that are currently in the market do not support forwarding of MIDlet suites.

This Chapter describes how adoption of OMA™ DRM could enable forwarding of MIDlet suites. Proposed provisioning scenarios were developed by the author and are not standardized anywhere. They are intended to demonstrate how superdistribution of MIDlets can happen. Section 2 concentrates on local forwarding of MIDlets from one mobile device to another via the Bluetooth or Infrared link. Section 3 describes how MIDlets may be exchanged using e-mail and the MMS. Section 4 shows that it may sometimes be useful to send a URL from which the suite can be downloaded instead of the suite itself.

7.1 Forwarding of MIDlets: Which Ones and How?

The OMA DRM specification, which is described in details in Chapter 4 of this thesis, defines a delivery method called superdistribution. This method can be used for forwarding of MIDlet suites (and any other DRM content) from one mobile device to another. Section 4.7.3 gives a detailed analysis of advantages and disadvantages of the method regarding MIDlet suites and arrives at the following conclusions:

- Superdistribution is advantageous mainly for commercial suites, but some free ones can also be distributed this way
- Universal adoption of OMA DRM for distribution of commercial MIDlets has a positive side effect: in the future, all non-DRM MIDlets can be considered free and forwarded without restrictions.

In the light of the preceding statements, methods for local forwarding of MIDlets take on a special significance. Indeed, forwarding of MIDlets via local connectivity (e.g., the Bluetooth or Infrared link) is easy and free (at least forwarding itself is always free, but in case of OMA DRM superdistribution the new user will need to obtain rights to use the suite, which will involve some costs). A scenario for such local provisioning is sketched in the next Section. It is based on the Object Push Profile. The Section explains how the profile can be used to enable OMA DRM superdistribution.

However, there are plenty of use cases where non-local forwarding of MIDlets is also advantageous. E.g., when the user wants to share some MIDlet suite with the other user who is not in the range of local connectivity methods. The MMS or e-mail can be a

solution in this case. Section 7.3 concentrates on how these bearers can be used for OMA DRM superdistribution.

7.2 Forwarding of MIDlets Using Object Push Profile

A method proposed in this Section allows sending of MIDlet suites in the same way as a business card can now be sent from one mobile phone to another via the Bluetooth or Infrared link. The method is based on the extended Object Push Profile and can be used both for the OMA DRM superdistribution (cf. Section 4.7.3) and for forwarding of free, non-OMA DRM-protected MIDlets. It has already been noticed that the latter is possible only when OMA DRM will protect all commercial suites.

When a business card is sent over the Bluetooth link, the Object Push Profile is used (for the OPP cf. Section 5.6.1). The OPP defines a subset of OBEX protocol; and as this protocol is bearer-independent, the same subset can also be used over the Infrared link. This thesis refers to such a combination as the OPP over Infrared.

In general, the method for forwarding of MIDlets from one mobile device to another proposed in this Section is similar to the push provisioning via the OPP described in Section 6.8. The distinctions are as follows. A MIDlet is sent not from the autonomous device but from another mobile device, and a DCF object is sent instead of a JAR file.

7.2.1 Additions to the Object Push Profile Service Record

Additions to the OPP service record are similar to those described in Section 6.8.1. The additions have the same justification: to let the mobile device that pushes the suite understand that the receiving mobile device is able to handle it properly. Section 6.8.1 adds two new formats to the *Supported Formats List* attribute:

0x07 = Application Descriptor (JAD file)
0x08 = Java Archive (JAR file).

One new format is needed for superdistribution:

0x09 = DRM Content Format (DCF).

A mobile device that wants to indicate that it supports superdistribution of MIDlets via the OPP adds Application Descriptor (*0x07*) and DCF (*0x09*) formats to the *Supported Formats List* attribute of the OPP service record. For the same reasons as described in Section 6.8.1, these formats are added even if the device is able to receive any types of objects.

If the OPP over Infrared link is used, changes are boiled down to the support for these two additional content formats.

Here are the requirements for mobile devices that send and receive MIDlets suites using the OPP.

7.2.2 Requirements for a Mobile Device that Pushes MIDlets

A Bluetooth-enabled mobile device must support:

- The role of the Push Client as defined in the Object Push Profile specification [28] part K:11
- The role of Local Device as defined in the Service Discovery Application Profile specification [28] part K:2.

An Infrared-enabled mobile device must support a subset of OBEX protocol defined for Push Clients in the OPP specification [28] part K:11.

7.2.3 Requirements for a Mobile Device that Receives MIDlets

A Bluetooth-enabled mobile device must support:

- The role of the Push Client as defined in the Object Push Profile specification [28] part K:11. Application Descriptor and DRM Content Format (*0x07* and *0x09*) formats must be added to the *Supported Formats List* attribute of the OPP service record
- The role of Remote Device as defined in the Service Discovery Application Profile specification [28] part K:2.

An Infrared-enabled mobile device must support a subset of OBEX protocol defined for Push Servers in the OPP specification with the extensions listed in Section 7.2.1 of this thesis.

7.2.4 Description of the Method

In the proposed provisioning scenario, both JAD and JAR files (the latter being in the DCF object) are sent from one mobile device to another. Justification for sending of both files is given in Section 6.2. In short, an Application Descriptor is needed for MIDP 2.0 trusted MIDlets because without them, these MIDlets will become untrusted on a new mobile device.

However, sending of both files leads to a problem (cf. Section 6.8.4): the user of the receiving mobile device may receive notification about two new messages and see two new files in the Inbox, both of them corresponding to the same Java application. Users are not technical experts and may be confused with the fact that sending of one MIDlet suite results in receiving of two objects, so a mechanism has to be introduced to hide these details. One solution is to combine JAD and DCF files into one object before sending and then send them as a single entity. In this case, the AMS must be able to install suites using such composite objects. Another option is to change the software that manages the Inbox so that it will present each corresponding pair of JAD and JAR files as one Java application. This approach also requires changes in the notification system. There should be notification about only one new message for each new corresponding pair of JAD and DCF files. There can also be other ways to hide this complexity. Later on in this Section, it is assumed that from the user's point of view, MIDlet suites are sent and received as single objects, a particular mechanism in use is omitted to simplify the discussion.

Figure 35 illustrates local forwarding of MIDlets from one mobile device to another via the OPP. The depicted scenario is bearer-independent; establishment of the Bluetooth or Infrared connection is exported and can be found in Figures 36 and 37 respectively.

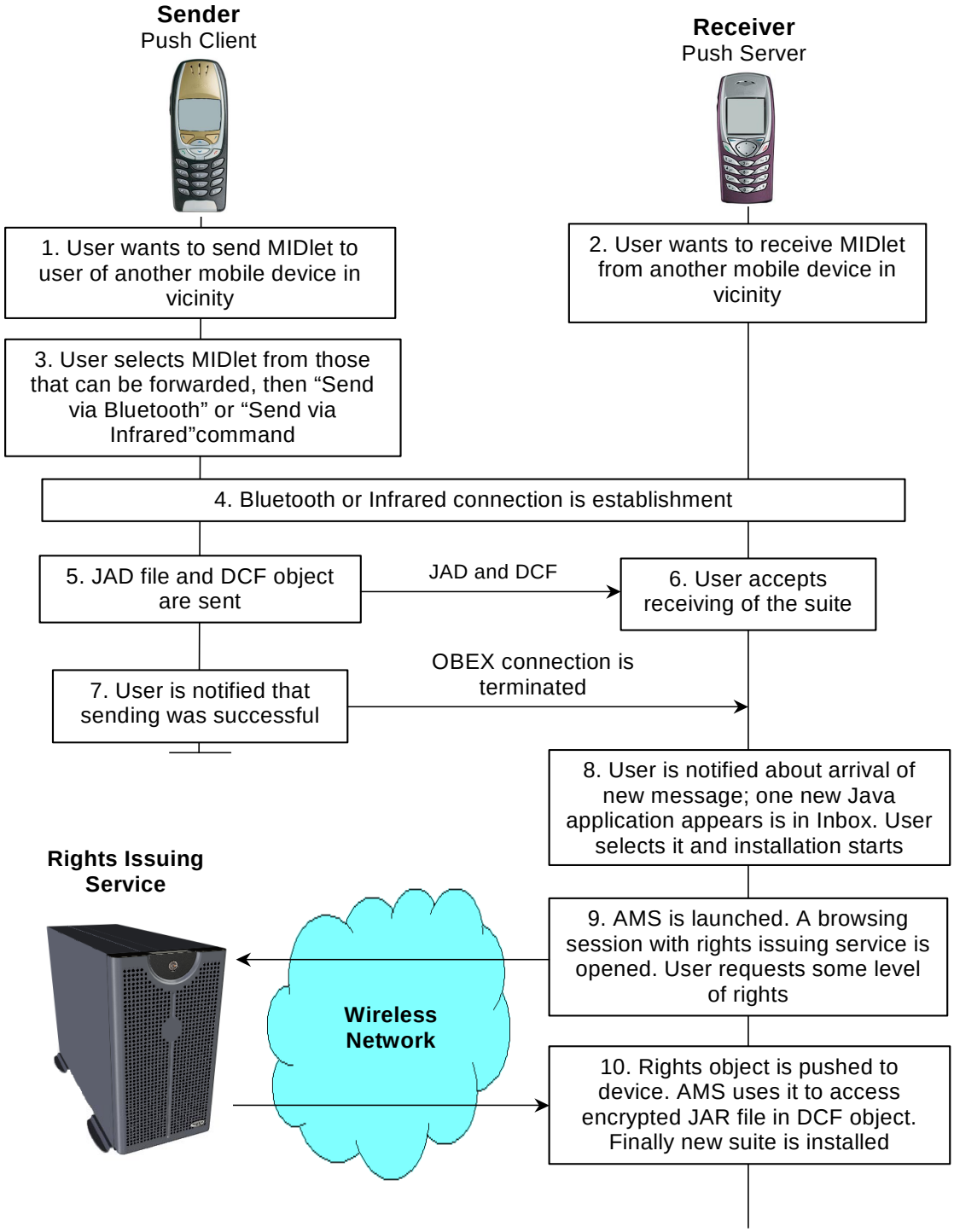


Figure 35. Local forwarding of MIDlets via the Object Push Profile

Here is a step-by-step description of the method.

1. The user of one mobile device wants to send the MIDlet suite to the user of a mobile device in the vicinity
2. The user of this second mobile device is eager to receive this MIDlet suite
3. The user of the sending mobile device selects a desired MIDlet suite (e.g., using the AMS). The MIDlet suite can be forwarded if it is the OMA DRM-protected suite

that was delivered using the OMA DRM separate delivery method. In terms of OMA DRM, forwarding of such suite is called superdistribution.
If the selected suite is eligible for forwarding, it has the Send command associated with it. The user selects this command and then specifies the local connectivity technology to be used for sending, either the Bluetooth or the Infrared one.
A software that will deal with sending is launched (it can be a part of the AMS). Later on in this Section, this software will be referred to as a provisioning application

4. The provisioning application establishes the OPP connection over the Bluetooth or the Infrared link. For Bluetooth connection establishment cf. Section 7.2.5. For Infrared connection establishment cf. Section 7.2.6.
   From now on, the mobile devices are connected, and the MIDlet suite can be pushed using OBEX protocol
5. The provisioning application pushes the JAD file and the DCF object to the receiver's Inbox
6. The user of the receiving mobile device may need to accept objects being pushed. Depending on the bearer in use and security settings of the mobile device, every pushed object may need to be accepted, or one permission may be needed for all objects that are pushed within one OBEX session, or no permission may be needed at all (e.g., when the Infrared link is used). Note that when each object being pushed has to be accepted, the user may be confused if two received objects result in one Java application. Such situations have to be avoided.
   The user accepts all incoming objects
7. The user of the sending mobile device is notified about successful sending of the MIDlet suite. OBEX connection is terminated
8. The user of the receiving mobile device is notified about arrival of a new message. A new, not installed, MIDlet suite is visible in the Inbox, and the user selects to install it
9. The AMS is launched to install the new MIDlet suite using the JAD file and the DCF object. The rights issuing service for this MIDlet suite is contacted and the user requests appropriate rights (cf. Section 4.7.3)
10. If the rights are granted, a rights object is sent to the mobile device using WAP Push. The AMS uses it to install the suite. The user is notified about the outcome of the installation; if it was successful, MIDlets from the new suite can be immediately launched.

7.2.5 Bluetooth Connection Establishment

Figure 36 depicts establishment of the OPP connection over the Bluetooth link.

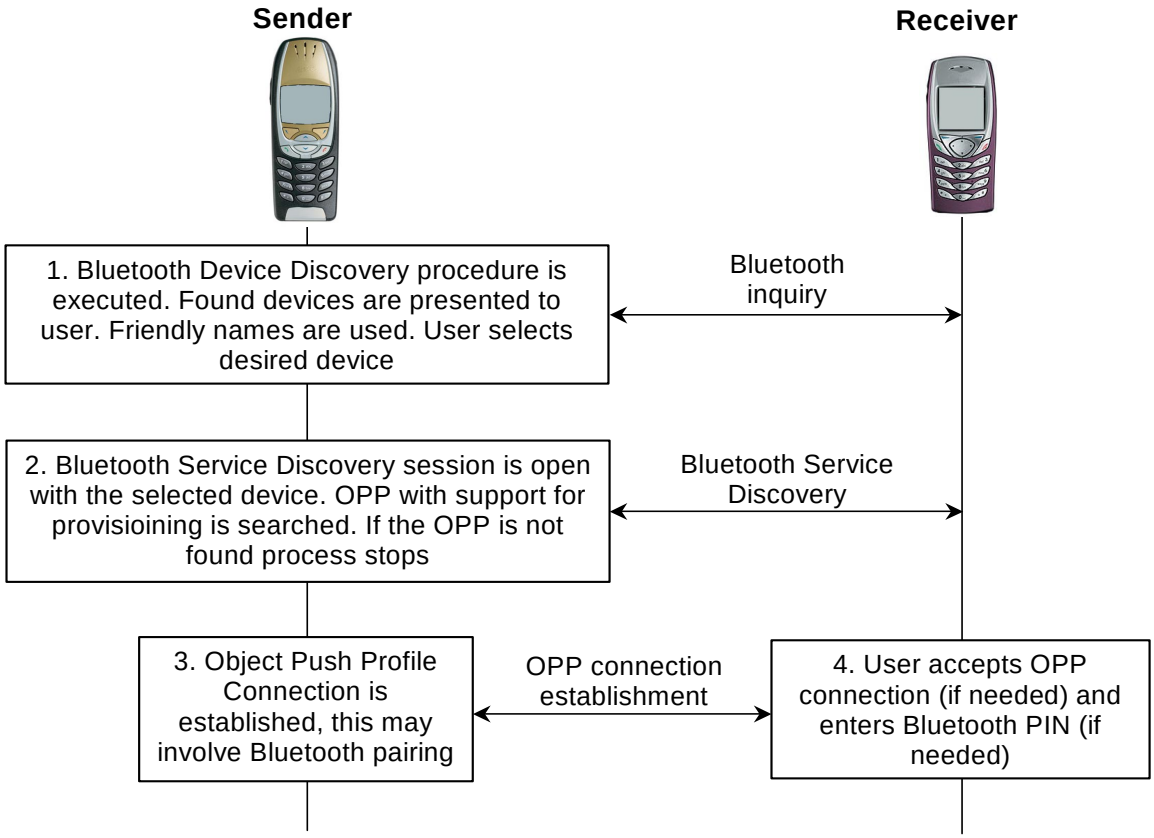


Figure 36. Establishment of OPP connection over the Bluetooth link

Here is a description of Bluetooth connection establishment.

1. After the user has selected the Bluetooth technology as a bearer for the push operation, Bluetooth Device Discovery is started. Found devices are filtered using their class of device values. Only devices with Object Transfer bit set to 1 are presented to the user. Friendly names are used. The user selects one of the devices
2. Bluetooth Service Discovery takes place on the selected device. The Object Push Profile with support for provisioning of superdistribution of MIDlets is searched.
   Sending of the MIDlet suite is also possible when the receiving device supports the normal OPP and accepts any type of objects. However, in this case the receiver may not be Java-enabled and would not be able to install the received MIDlet suite. When forwarding a suite to such a device, a warning should be presented to the user of the sending mobile device.
   Sending is not possible if the OPP is missing or JAD and DCF files are not accepted. In this case, the sending process stops with the appropriate message
3. The OPP connection with the receiving mobile device is established. Depending on settings of the device, this may involve Bluetooth pairing. In this case, the user needs to enter a Bluetooth PIN
4. If Bluetooth pairing is needed, the user of the receiving mobile device enters a Bluetooth PIN (PINs must be the same on both devices). In addition, the user may need to accept the OPP connection.

Now the devices are connected and the sending mobile device may start pushing the MIDlet suite.

7.2.6 Infrared Connection Establishment

Figure 37 depicts establishment of the OPP connection over the Infrared link.

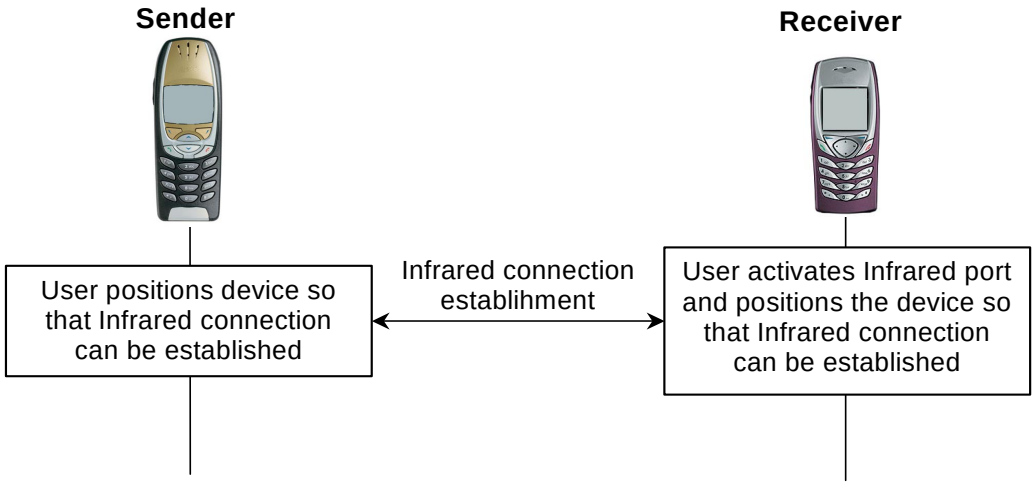


Figure 37. Establishment of OPP connection over the Infrared link

If a MIDlet suite is sent via the Infrared link, the user of the receiving device must activate the Infrared port. On the sending device, the port is activated automatically when the user selects Infrared as a push bearer. After that both devices are positioned so that their Infrared ports are pointing at each other. Now the mobile devices are connected, and the MIDlet suite can be pushed using OBEX protocol.

7.2.7 Advantages and Disadvantages of the Method

Advantages:

- Can be easily standardized as it uses standard Object Push Profile and the OMA DRM superdistribution
- Simple since it is based on the relatively uncomplicated OPP
- Can be used both over Bluetooth and Infrared connection
- Can be easily implemented on mobile devices that already support the OPP and the OMA DRM superdistribution.

Disadvantage:

- The method does not guarantee that the forwarded MIDlet will be able to work properly on the new mobile device. Indeed, the new mobile device may not support some of the required additional packages or may support an older version of the MIDP, or may have unacceptable screen resolution, etc. It is assumed that users of the mobile devices have some idea about compatibility of their devices in terms of the J2ME. MIDlet's portability plays a significant role in this problem, as there are MIDlets that can be run on almost any mobile device, and there are MIDlets that are created for a specific mobile device model and cannot be used on the others.

It should be noted that if OMA DRM is universally adopted, and all commercial MIDlet suites are distributed using OMA DRM methods, all other suites can be treated as free (cf. Section 4.7.3). These free suites can also be forwarded using the method described above. The only difference is that a JAR file will be sent instead of a DCF object.

7.3 Superdistribution of MIDlets Using E-mail and MMS

MIDlet suites can be sent as attachments to an MMS or e-mail message. This can be useful when the receiving mobile device is out of the range of local connectivity technologies and local provisioning (e.g., using the method proposed in Section 7.2) is not possible. Just as with local forwarding, the MIDlet suites that can be forwarded are OMA DRM-protected ones whose JAR files are encrypted and packed into DCF objects (the OMA DRM superdistribution method).

Figure 38 depicts how the MMS or e-mail can be used for forwarding of MIDlets from one mobile device to another.

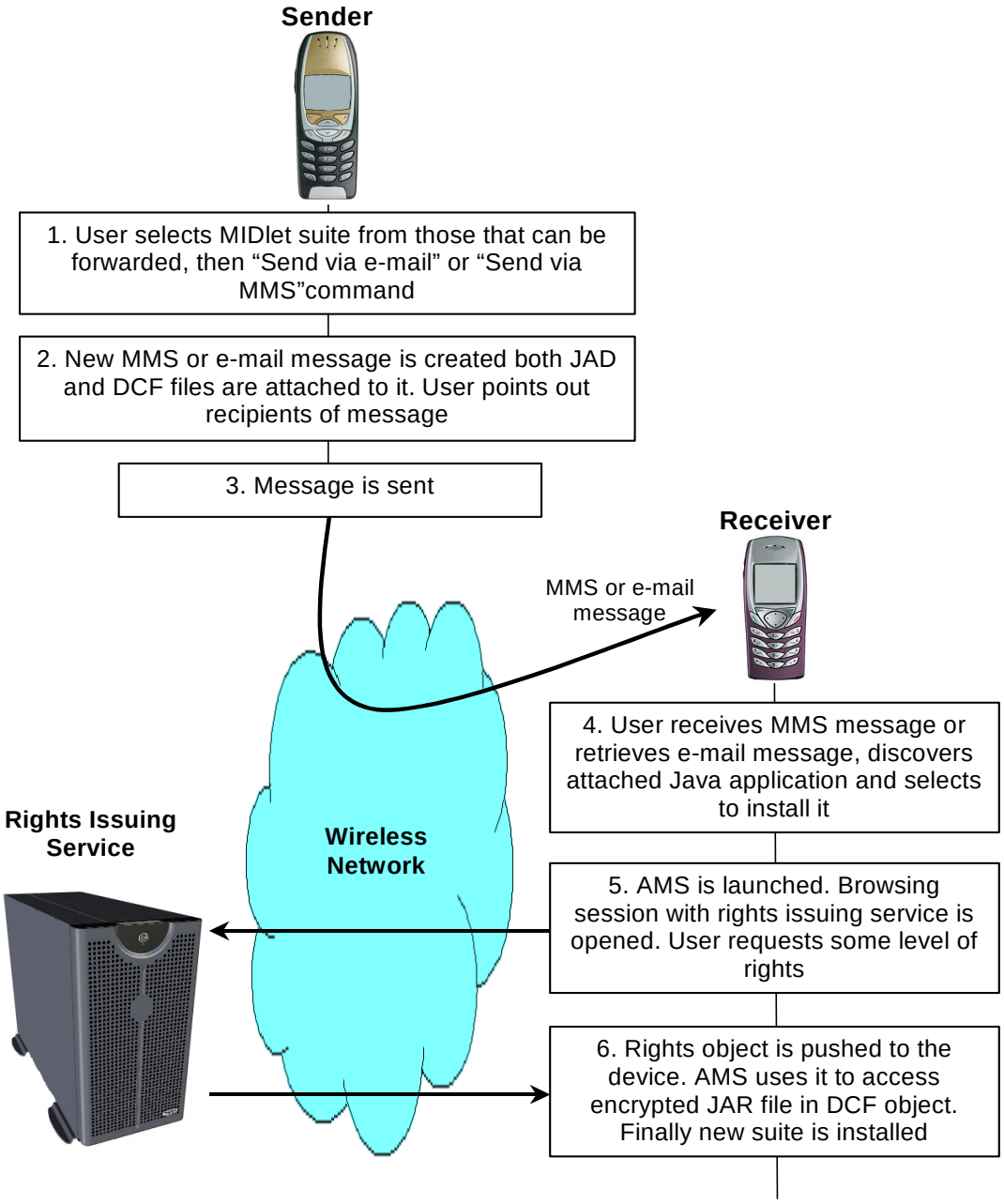


Figure 38. Forwarding of MIDlets using the MMS or E-Mail

Here is a brief description of the provisioning scenario presented in Figure 38.

1. The user of the sending mobile device selects a desired MIDlet suite and then chooses a sending method, either the MMS or e-mail. Alternatively, the user creates a new MMS or e-mail message and then attaches a MIDlet suite to it. In both cases, a MIDlet suite is selected among the installed suites that are eligible for forwarding. There can also be a possibility to forward a not installed suite (such suites may reside, e.g., in the Inbox).
2. A new MMS or e-mail message is created; both JAD and DCF files of the selected suite are automatically attached to it. The assumption is that each installed MIDlet

suite has a JAD file; the AMS may create it for suites that did not have any. Not installed suites may have a JAD file missing.
The user points out one or more recipients of the message, and optionally adds some textual description of the suite being sent

3. The MMS or e-mail message is sent using some network connection
4. The recipient receives the message, discovers a new MIDlet suite in it and selects to install it
5. The AMS is started to install the suite using a pair of JAD and DCF files found in the attachment. A browsing session with a rights issuing service is opened and the user requests appropriate rights (cf. Section 4.7.3)
6. If the rights are granted, a rights object is pushed to the mobile device. The AMS uses it to install the suite. The user is notified about the outcome of the installation. In case of success, MIDlets from the new suite can be immediately launched.

7.4 Forwarding of an Application Descriptor Only

There are situations when it makes sense to forward only a JAD file instead of the whole suite. The use case is as follows.

A free MIDlet suite (e.g., a multiplayer game) is available in the Internet. The user already has the suite and wants to share it with another user. It is not possible to send the whole suite to the new user, because superdistribution is not supported (suite is distributed as a plain text, not in the DCF format). However, if the first user forwards the the JAD file of the suite, the second user will be able to install the suite (the AMS will use the JAD file to download the JAR file from the Internet).

In essence, sending a JAD file is similar to sending a link to the JAR file (the URL of the JAR file is inside the JAD file). Of course, the first user can simply tell the URL to the second user, but it is not always possible, as the user may not be aware of the URL. The method is applicable only if the JAR file is available at some permanent URL, not at the one created only for one downloading session.

Advantage:

- Beneficial for distribution of free MIDlet suites that do not want to use the OMA DRM superdistribution method.

Disadvantages:

- Limited usage because a URL of a JAR file is not always permanent
- Not free from the user's point of view, as the user will be charged for OTA downloading of a JAR file
- An inappropriate version of a MIDlet suite may be downloaded. If the provisioning server uses device identification (cf. Section 3.2.8), then the JAD file forwarded is intended for a certain model of a mobile device. If a mobile device of another model uses the same JAD file for installation, it may receive an incorrect JAR file (intended for another model). This drawback can be fixed by forwarding a URL to a JAD file instead of a JAD file, but this brings additional problems.

8. Conclusions

The thesis has made an overview of existing provisioning technologies and proposed several new methods. The next four Sections summarize the results. Some possible directions for future work can be found in Section 8.5.

8.1 Existing provisioning methods

OTA provisioning of MIDlets remains the most important provisioning method. Its drawbacks, viz. inconvenient MIDlet discovery and lengthy and sometimes expensive downloading, are compensated by two huge benefits. First, OTA provisioning is the only provisioning technology that will invariably be supported by any MIDP 2.0-compliant device. Second, the method can be implemented in almost any wireless network as it relies on the HTTP 1.1 over any bearer.

WAP Push provisioning is currently used mostly for delivery of MIDlets that are sold for money. The method is attractive because it allows transferring the process of MIDlet discovery to a PC, which is usually more convenient and cheap for the user.

Provisioning using PC connectivity software and via the Bluetooth or Infrared link is typically used for installation of free MIDlet suites. These methods are also used by MIDlet developers to test their applications. The main benefits of the methods are their rapidness and the fact that no direct costs are associated with provisioning.

Provisioning via e-mail and the MMS is not widely used now. However, provisioning via the MMS may replace WAP Push provisioning in the future. Indeed, the former is more attractive from the user's point of view as receiving of multimedia messages is free in most wireless networks, while in WAP Push provisioning the user needs to pay for downloading of a MIDlet. In all other respects, the two methods are similar. Currently, there are two obstacles in the way to provisioning via the MMS. First, not all mobile phones and wireless networks currently support the MMS. Second, many mobile phones and wireless networks currently have limitations on the maximum size of the multimedia message they can receive, which is a serious drawback for provisioning.

8.2 Open Mobile Alliance™ DRM

Adoption of the OMA™ DRM for protection of MIDlets will have positive effect on the J2ME platform. First, as the Java technology is based on open standards, the J2ME platform will benefit from the industry-wide standard for protection and consumption of MIDlets. Second, adoption of the OMA DRM will facilitate easy creation of demo versions of MIDlets. This feature will be very much appreciated by providers who sell MIDlets for money. Currently, the user often has no chance to try a MIDlet before buying it, which discourages many potential buyers. Finally, the OMA DRM enables superdistribution of MIDlets.

On the other hand, adoption of the OMA DRM is quite a serious task from the technical point view and will lead to increased complexity of the AMS and provisioning servers. However, most mobile phones that will implement the technology will do so not only because of MIDlets but also to execute control over consumption of ring tones, wallpapers, etc. This means that the OMA DRM engine can be implemented as a separate subsystem, which will be used by the AMS and other systems.

Another issue that can be regarded as a drawback of the OMA DRM is that superdistribution does not enable free forwarding of free MIDlets. In fact, superdistribution involves establishment of an Over-The-Air connection, which makes the method (and the MIDlet) non-free from the user's point of view.

8.3 Local provisioning from autonomous devices

Provisioning from autonomous devices may become topical in the nearest feature. In essence, the more widespread the J2ME platform and the Bluetooth wireless technology will be, the more use cases for such type of delivery will appear. Among methods for local provisioning from standalone devices proposed in the thesis, the most universal is the family of methods based on the Private Area Networking Profile. However, this profile is quite demanding, and many mobile devices will not support it in the foreseeable future. Therefore, it may make sense to use methods that are based on simpler profiles. The File Transfer Profile can be used for pull provisioning, while the Object Push Profile for "active" push provisioning.

Nevertheless, the PAN Profile seems to be the most suitable Bluetooth profile to exercise such type of provisioning in the long-term outlook.

8.4 Forwarding of MIDlets

The Object Push Profile over the Bluetooth or Infrared link seems to give good fit for OMA DRM superdistribution of MIDlets. However, success of the superdistribution concept in application to MIDlets largely depends on portability of MIDlets. Indeed, it does not make sense to forward the MIDlet if it does not work on the new mobile device. Another important issue is arrangement of the rights issuing service. It is necessary to bear in mind that the user is charged for an Over-The-Air connection that is required to request a right object. Therefore, the user's interaction with the service should be as short as possible. E.g., if users are charged sensible sums simply for requesting rights for free MIDlets, they will most likely ignore superdistribution.

8.5 Future work

This study by no means exhausts the subject of provisioning of MIDlets. In essence, provisioning of a MIDlet, in a simplest case, is just delivery of one or two files to a mobile device. It can be done by numerous methods, and the number of these methods will grow as mobile communication devices evolve.

A goal for future work may be experimental implementation of one or several of the provisioning methods proposed in this thesis. Such implementation will provide valuable information on application of methods in practice and may reveal new issues that were not identified in this paper.

Another interesting topic to investigate is obfuscation of MIDlet suites, i.e. making MIDlet suites resistant to disassembling. This issue may become topical when local distribution of MIDlets via MIDlet dispensers will be in use. When MIDlets are distributed using such methods, there is a risk that some of them will end up in other devices than those supporting the MIDP, e.g. in a laptop. Even if a MIDlet is free, its disassembling

may be undesirable as a JAR file may contain pictures, audio clips, etc. Obfuscation may be a solution to this problem.

Another challenging direction is research into forwarding of free MIDlets. OMA DRM superdistribution does not seem to be optimal solution here as it is not free from the user's point of view. It will be very beneficial to find a way to forward free MIDlets without additional costs.

References

[1] Sun Microsystems Java™ 2 Platform, Standard Edition Web Site. http://java.sun.com/j2se/

[2] Sun Microsystems Java™ 2 Platform, Enterprise Edition Web Site. http://java.sun.com/j2ee/

[3] *"Java™ 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices"*, White Paper, Sun Microsystems, Inc., 2000, http://java.sun.com/products/cldc/wp/KVMwp.pdf

[4] Sun Microsystems Java Card™ Technology Web Site. http://java.sun.com/products/javacard/

[5] Java Community Process Web Site. http://www.jcp.org

[6] *"Java 2 Platform, Micro Edition Datasheet"*, Sun Microsystems, Inc., 2002, http://java.sun.com/j2me/j2me-ds.pdf

[7] Ortiz, Enrique C., *"A Survey of J2ME Today"*, November 2002. http://wireless.java.sun.com/getstart/articles/survey/

[8] Golsing, James et al., *"Java language specification, Second Edition"*, 2000 http://java.sun.com/docs/books/jls/

[9] *"Connected, Limited Device Configuration, Specification version 1.0a, Java 2 Platform, Micro Edition"*, Sun Microsystems, Inc., 2000

[10] Sun Microsystems J2SE Reflection Web Page. http://java.sun.com/j2se/1.4.1/docs/guide/reflection/

[11] *"Java Virtual Machine Specification"*, Sun Microsystems, Inc., http://java.sun.com/docs/books/vmspec/

[12] *"Connected Limited Device Configuration, Specification version 1.1. Java 2 Platform, Micro Edition"*, Sun Microsystems, Inc., 2003

[13] *"Mobile Information Device Profile (JSR-37), JCP Specification, Java 2 Platform, Micro Edition 1.0a"*, Sun Microsystems, Inc., 2000.

[14] *"Over The Air User Initiated Provisioning. Recommended Practice for the Mobile Information Device Profile, Version 1.0*", Sun Microsystems, Inc., 2001

[15] *"Mobile Information Device Profile for Java™2 Micro Edition, version 2.0"*, JSR 118 Expert Group, Java Community Process, 2002

[16] Knudsen, Jonathan. *"What's New in MIDP 2.0"*, 2002. http://wireless.java.sun.com/midp/articles/midp20/

[17] *"Mobile Information Device Profile"*, Datasheet, Sun Microsystems, Inc., 2002, http://java.sun.com/products/midp/midp-ds.pdf

[18] *"Telecom Glossary 2000"*, American National Standard for Telecommunications, http://www.atis.org/tg2k

[19] *"Digital Rights Management, version 1.0*", Open Mobile Alliance, 2002, http://openmobilealliance.org/

[20] Franks J., et al., *"HTTP Authentication: Basic and Digest Access Authentication"*, 1999. http://www.ietf.org/rfc/rfc2617.txt

[21] *"WAP Push Architectural Overview"*, Wireless Application Protocol Forum, 2001, http://www.wapforum.org

[22] Fielding R., et al., *"Hypertext Transfer Protocol -- HTTP/1.1"*, 1999. http://www.ietf.org/rfc/rfc2616.txt

[23] *"DRM Content Format"*, version 1.0, Open Mobile Alliance, 2002, http://openmobilealliance.org

[24] *"Rights Expression Language "*, version 1.0, Open Mobile Alliance, 2002, http://openmobilealliance.org/

[25] *"Binary XML Content Format Specification, Version 1.3"*, Wireless Application Protocol Forum, 2001, http://www.wapforum.org

[26] Miller, Brent A., Bisdikian Chatschik, *"Bluetooth Revealed"*, Prentice Hall, 2001

[27] *"Specification of the Bluetooth System"*, Specification Volume 1. Core. Version 1.1

[28] *"Specification of the Bluetooth System"*, Specification Volume 1. Profiles. Version 1.1

[29] *"Bluetooth Assigned Numbers"*, http://www.bluetoothsig.org/assigned-numbers/

[30] Bray, Jennifer; Sturman, Charles F. *"Bluetooth: connect without cables"*, Prentice Hall, 2001

[31] International Organization for Standardization in ISO/IEC 11578:1196, Information technology - Open Systems Interconnection - Remote Procedure Call (RPC), 1996 http://www.iso.ch

[32] *"IrDA Object Exchange Protocol OBEX, version 1.3"*, Infrared Data Association, 2003, http://www.irda.org/standards/specifications.asp

[33] S. Hartwig et al., *"Wireless Microservers"*, IEEE *"Pervasive Computing"*, Volume 1, Number 2, April-June 2002

[34] *"Personal Area Networking Profile, version 1.0"*, Bluetooth SIG, 2003

Bibliography

Riggs, Roger; Taivalsaari, Antero; VandenBrink, Mark, *"Programming Wireless Devices with the Java 2 Platform, Micro Edition: J2ME Connected Limited Device Configuration (CLDC), Mobile Information Device Profile (MIDP)"*, Addison Wesley, 2001

Knudsen, Jonathan, "*Wireless Java:Developing with Java 2, Micro Edition*", Apress, 2001

Piroumian, Vartan, *"Wireless J2ME Platform Programming"*, Sun Microsystems, 2002

Holzner, Steven, *"Inside XML"*, New Riders, 2001

McLaughlin, Brett, *"Java and XML"*, O'Reilly, 2000

Sun Microsystems. *"The CLDC HotSpot™ Implementation Virtual Machine"*, White Paper, 2002, http://java.sun.com/products/cldc/wp/CLDC_HI_WhitePaper.pdf

Java 2 Platform, Micro Edition Web Site http://java.sun.com/j2me/

Forum Nokia Web Site http://www.forum.nokia.com

Motorola Developers Programs Web Site http://developers.motorola.com

Infrared Data Association Web Site http://www.irda.org/

Bluetooth SIG Web Site https://www.bluetooth.org/

Digianswer Bluetooth Software Suite Web Site http://www.btsws.com

Legal Notices