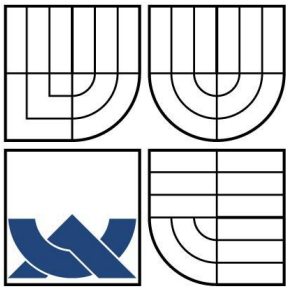
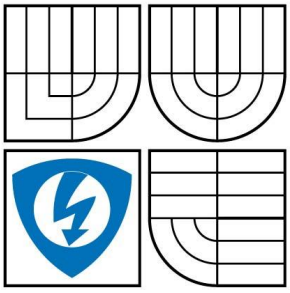


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKACNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

# ROZHRANÍ PRO SKUPINOVÉ ODESÍLÁNÍ SMS V JAVA ME
JAVA ME GROUP SMS MESSAGING INTERFACE

DIPLOMOVÁ PRÁCE
MASTER THESIS

AUTOR PRÁCE — BC. TOMÁŠ KUBINA
AUTHOR

VEDOUCÍ PRÁCE — ING. PETR KOVÁŘ
SUPERVISOR

BRNO 2008

# LICENČNÍ SMLOUVA
# POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

**1. Pan**

Jméno a příjmení: Bc. Tomáš Kubina

Bytem: Bílá 1959, 54701, Náchod

Narozen/a (datum a místo): 10.2.1981, Náchod

(dále jen „autor“)

a

**2. Vysoké učení technické v Brně**

Fakulta elektrotechniky a komunikačních technologií

se sídlem Údolní 244/53, 602 00, Brno

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

prof. Ing. Kamil Vrba, CSc.

(dále jen „nabyvatel“)

## Čl. 1
## Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - □ disertační práce
   - * diplomová práce
   - □ bakalářská práce
   - □ jiná práce, jejíž druh je specifikován jako

(dále jen VŠKP nebo dílo)

Název VŠKP: Rozhraní pro skupinové odesílání SMS v JavaME

Vedoucí/školitel VŠKP: Ing. Petr Kovář

Ústav: Ústav Telekomunikací

Datum obhajoby VŠKP:

VŠKP odevzdal autor nabyvateli v*:

- □ tištěné formě – počet exemplářů 2
- □ elektronické formě – počet exemplářů 2

* hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

**Článek 2**
**Udělení licenčního oprávnění**

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   * ihned po uzavření této smlouvy
   □ 1 rok po uzavření této smlouvy
   □ 3 roky po uzavření této smlouvy
   □ 5 let po uzavření této smlouvy
   □ 10 let po uzavření této smlouvy
   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

**Článek 3**
**Závěrečná ustanovení**

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne:

Nabyvatel: Autor:

# Abstrakt

Diplomová práce se zabývá možnostmi balíčku Wireless Messaging API za účelem vytvoření aplikací využívajících pro svoji funkci příjem a odesílání krátkých textových zpráv. Popisuje programovací jazyk JavaME, základy formátování a posílání SMS, detailní rozebrání balíčku Wireless Messaging API, aplikaci, jež je součástí diplomové práce a konečně bezpečnostní politiku jazyka JavaME.

# Klíčová slova

Java, JavaME, Micro Edition, SMS, Wireless Messaging API, WMA, CLDC, RMS, ochranné domény, důvěryhodný MIDlet, nedůvěryhodný MIDLet

# Abstract

This Diploma Thesis have been describing possibilities of Wireless Messaging API, which are used for applications handling with Short Messages. It have been describing basics of programming language JavaME, basics of formating and sending SMS, detailed analysis of Wireless Messaging API, application, which is part of my diploma thesis and finally Security Politics of JavaME programming language.

# Keywords

Java, JavaME, Micro Edition, SMS, Wireless Messaging API, WMA, CLDC, RMS, protection domains, trusted MIDlet, untrusted MIDLet

# Prohlášení

Prohlašuji, že svoji diplomovou práci na téma rozhraní pro skupinové odesílání SMS v JavaME jsem vypracoval samostatně pod vedením vedoucího projektu a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením tohoto projektu jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobních a jsem si plně vědom následků porušení ustanovení § 11 a následujícího autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení 152 trestního zákona č. 140/1961 Sb.

V Brně dne..... .....
podpis autora

# Poděkování

Děkuji vedoucímu diplomové práce Ing. Petrovi Kovářovi za pomoc při psaní a programování diplomové práce. Především děkuji za schopnost usměrňovat množství dostupných informací o dané problematice, aby mohla vzniknout práce, která se detailně zabývá daným tématem, ale zároveň není díky detailnímu popisu klíčových částí vytržena z kontextu celé problematiky programování aplikací pro malá zařízení.

V Brně dne..... .....
podpis autora

# Obsah


Seznam zkratek 9

Úvod 11

1 Short Message Services (SMS) 13

2 Java ME 16

2.1 CLDC 17

2.2 CDC 19

2.3 Push Registry 20

3 Wireless Messaging 23

3.1 Connector 25

3.2 Message Connection 25

3.3 Message 27

3.4 TextMessage 29

4 Aplikace 31

4.1 Popis funkce 31

4.2 Popis prostředků 35

5 Bezpečnost JavyME 38

5.1 Nízkoúrovňová bezpečnost 38

5.2 Pískoviště a důvěryhodnost MIDletů 38

5.3 End-to-end bezpečnost 42

5.4 Certifikace MIDletu 43

5.5 Přístup WMA k síťovému rozhraní 44

Závěr 45

Literatura 47

# Seznam zkratek

| | |
|---|---|
| *AMS* | Application Management Software |
| *API* | Application Programing Interface |
| *CA* | Certifikate Authorities‘ |
| *CDC* | Connected Device Configuration |
| *CLDC* | Connected Limited Device Configuration |
| *CVM* | Compact Virtual Machine |
| *DA* | Destination Adress |
| *DCS* | Data Coding Scheme |
| *ETSI* | European Telecommunication Standards Institute |
| *FP* | Foundation Profile |
| *GFC* | Generic Connection Framework |
| *GSM* | Global System for Mobile communication |
| *HTTPS* | Hyper Text Transfer Protocol Secure |
| *IANA* | Internet Assigned Numbers Authority |
| *IP* | Internet Protocol |
| *JSR* | Java Specification Request |
| *KVM* | Kilobyte Virtual Machine |
| *MS* | Mobile Station |
| *OA* | Originator Adress |
| *PC* | Personal Computer |
| *PBP* | Personal Basic Profile |
| *PDA* | Personal Digital Assistant |
| *PDAP* | Personal Digital Assistant Profile |
| *PDU* | Protocol Data Unit |
| *PID* | Protocol Identifier |
| *PKI* | Public Key Infrastructure |
| *PP* | Personal Profile |
| *RMS* | Record Management System |
| *RSA* | Rivest, Shamir, Adleman |

| | |
|---|---|
| *SCTS* | Service Center Time Stamp |
| *SIM* | Subscriber Identity Module |
| *SMS* | Short Messaging Service |
| *SMS-IWMSC* | SMS Interworking Message Switching Centrum |
| *SMSC* | Short Message Service Center |
| *TCP* | Transmision Control Protocol |
| *UD* | User Data |
| *UDL* | User Data Length |
| *UDP* | User Datagram Protocol |
| UMTS | Universal Mobile Telecommunication System |
| *URL* | Uniform Resource Locator |
| *VP* | Validity Period |
| *WMA* | Wireles Messaging Application Programing Interface |

# Úvod

Tématem mé Diplomové práce je přiblížit problematiku programování aplikací pracujících s odesíláním a příjmem krátkých textových zpráv v tzv. malých zařízeních, jakými jsou například mobilní telefony. V průměrném součtu má mobilní telefon již každý Čech, proto je s podivem, že zdrojů o programování mobilních aplikací je, v porovnání s internetovými stránkami a knihami o klasickém programovaní, velice málo, i když k mobilnímu telefonu má přístup více lidí, než k osobnímu počítači. Malým zařízením zde však není myšlen jen mobilní telefon, případně malé přenosné počítače (PDA), ale také televizory, audio soupravy, kopírky atd. V podstatě jakékoliv zařízení, které má omezené výkonné a paměťové prostředky, takže by zde aplikace na bázi standardní edice Javy nemohly být spuštěny, protože mají velké nároky jak na výpočetní výkon, tak na paměť. Z tohoto důvodu byl společností Java vyvinut programovací jazyk JavaME.

Diplomová práce „Rozhraní pro skupinová odesílání SMS v Java ME“ je koncipována tak, že jsou v prvních kapitolách probírány základy subjektu a objektu práce nutné pro pochopení textu dalších kapitol. V těch jsou, s využitím vědomostí předchozích kapitol, probírány do nejmenších detailů důležité věci nutné pro úspěšné zvládnutí cíle. V každé kapitole se práce nejvíce věnuje informacím potřebným k úspěšnému dosažení cíle, avšak aby práce nepůsobila dojmem, že je nějaká oblast vytržena z kontextu, je zde vždy uvedeno základní rozdělení každé z probíraných oblastí a nastínění funkce, případně účelu každého ze zmíněných bodů.

V první kapitole jsou zmíněny standardy, podle kterých se musí řídit formát krátkých zpráv a proces posílání SMS. Z nich jsou probrány podrobnosti potřebné dále, jako je délka zprávy, způsoby kódování, problematika posílání zpráv. Pro hlubší zkoumání je v konečném soupisu literatury uveden odkaz, kde se nacházejí.

Dále je popsán nástroj, programovací jazyk JavaME, který je použit k vytvoření aplikace. Je zmíněno jeho začlenění do dalších rodin jazyku Java, jeho hlavní odlišnosti od základní rodiny JavaSE a především je vysvětleno jeho

členění na konfigurace a profily. Pro další vývoj aplikace je to nezbytné, protože podle implementované konfigurace a profilů, které má na starosti výrobce malých zařízení, zjistíme, které prostředky můžeme využít k dosažení cíle. Pokud například víme, že v mobilním telefonu je konfigurace MIDP 2.0, ale není implementována knihovna Wireless Messaging API, pak nemůžeme posílat krátké textové zprávy, i když zmíněná neimplementovaná knihovna využívá přístupu k síti pomocí funkcí konfigurace MIDP 2.0 (ale samotná konfigurace MIDP neobsahuje funkce pro manipulaci s krátkými zprávami). Teoreticky by neměl být problém si podobné knihovny doprogramovat, ale bezpečnostní politika Javy toto neumožňuje.

Třetí kapitola je jednou z nejdůležitějších kapitol diplomové práce, neboť do podrobností rozebírá balíček knihoven Wireless Messaging API. Jsou zde popsána všechna rozhraní, které balíček obsahuje a dopodrobna popsány důležité funkce jednotlivých rozhraní. Dále jsou zde zmíněna rozhraní, která Wireless Messaging API neobsahuje, ale jsou potřebná pro otevření síťového spojení, jeho provoz (odeslání a příjem krátkých zpráv) a konečně i jeho uzavření. Zde v podstatě navazujeme na první kapitolu, jen se na problematiku posílání zpráv díváme z pohledu programátora, který má k dispozici určité programovací prostředky, kdežto v první kapitole je problematika popsána z pohledu zvědavějšího uživatele aplikace.

Následující kapitola je věnována samotné aplikaci. Je zde představena funkce aplikace, ukázány důležité útržky kódu a následně je i představen programovací nástroj NetBeans, kterého bylo užito pro vytvoření kódu.

Důležitá je i poslední kapitola, kde je rozebrána bezpečností stránka aplikací vzniklých v JavaME, protože bezpečnost Javy pro malá zařízení svazuje programátorovi nejvíce ruce. V hodně případech rozhoduje i o úspěchu daného programu. Je zde podrobněji popsána problematika povolování (případně zákazů) přístupu k potřebným síťovým rozhraním a dále proces certifikace.

# 1 Short Message Service (SMS)

Krátké textové zprávy SMS (Short Message Service) v síti GSM (Global System for Mobile communication) jsou specifikovány Evropským ústavem pro telekomunikační normy ETSI (European Telecommunication Standards Institute) v dokumentech GSM 03.40 a GSM 03.38. [1] Podle těchto standardů lze zprávou přenést až 140 oktetů dat – v případě použití sedmibitové abecedy je to všeobecně známých 160 znaků. Přenášet je můžeme buď v textovém režimu, jež je pouhou reprezentací bitového toku druhého režimu zvaného PDU (Protocol Data Unit). [2]

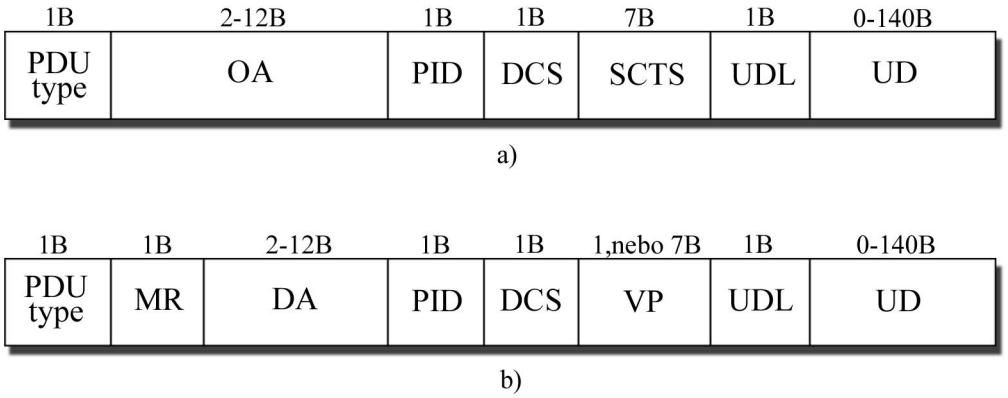
Jelikož můžeme zprávy posílat i do jiných zemí, je třeba vyřešit kompatibilitu různých abeced. Na počítači abecedy vybíráme pomocí AT příkazu (AT+CSCS), na mobilní stanici MS (Mobile Station) se příslušná abeceda pro dekódování vybere automaticky [3].

Poslání samotné SMS zprávy můžeme přirovnat k poslání nespolehlivého datagramu UDP v síti TCP/IP. Dělí se na dvě fáze. V první fázi „originating“ mobilní stanice nastaví adresu příjemce, dobu platnosti zprávy (jedná se o uživatelsky nastavitelné parametry, další parametry datagramu PDU nastaví automaticky) a zprávu přenese do servisního centra své domovské sítě SMSC (Short Message Service center). Adresa centra SMSC je nastavena v paměti mobilní stanice při její aktivaci. Každá síť má ještě speciální centrum (SMS-IWMSC), které slouží pro příjem SMS zpráv účastníků, jež danou síť nemají jako domovskou (roaming). V druhé, složitější fázi „terminating“ se zpráva z tohoto centra přenese k cílové mobilní stanici. Servisní centrum přebírá za zprávu odpovědnost po celou dobu její platnosti (dobu platnosti však může servisní centrum změnit). Složitější je tato fáze proto, že servisní centrum musí zjišťovat, kde se cílový účastník nachází (domovská, zahraniční síť), zda je v dané době přihlášený k síti, zda nemá plnou paměť SMS zpráv apod. [2]

| | |
|---|---|
| SMS deliver | doručí zprávu směrem k MS |
| SMS submit | doručí zprávu směrem od MS |
| SMS deliver report | doručí příčinu selhání směrem od MS |
| SMS submit report | doručí příčinu selhání směrem k MS |
| SMS command | doručí příkaz od MS |
| SMS status report | doručí hlášení o stavu k MS [4] |

Tab. 1.1: Druhy PDU datagramů

Pro transport zpráv slouží celkem šest druhů datagramů PDU (viz tab. 1.1). Uživatelská data se přenášejí datagramem SMS-deliver a SMS-submit, jejichž strukturu můžeme vidět na obr. 1.1. Podrobnější popsání jednotlivých prvků datagramu není cílem této práce (viz. [4]). Podrobněji budou zmíněna pouze ta pole, na která může mít programátor při tvorbě aplikace vliv.




Obr 1.1: Datagram a) SMS deliver b) SMS submit

Jedná se především o pole OA (Originator Adress) obsahující adresu odesílatele dané zprávy (v případě mobilního telefonu se jedná o telefonní číslo). Položka DA (Destination Adress) v sobě ukládá cílovou adresu, na kterou má být zpráva doručena. Oba druhy adres zabírají v datagramu 2-12 bytů. Ukládá se zde délka čísla, formát telefonního čísla (národní, či mezinárodní formát) a zbylých 10 oktetů slouží k uložení samotného čísla v BCD kódu. V případě lichého počtu číslic musí být před poslední číslicí znak „F“. Dalšími položkami jsou formát, či

protokol doručené zprávy PID (Protocol Identifier), kódovací schéma samotných dat DCS (Data Coding Scheme), čas doručení zprávy do servisního centra SCTS (Service Center Time Stamp), čas platnosti samotné zprávy při cestě k cíli VP (Validity Period).

Posledním je pole UD (User Data), které obsahuje zakódovaná data. Maximálně může mít velikost 140 bytů. V poli UDL (User Data Length) se uvádí délka uživatelských dat nikoliv v bytech, ale ve znacích, záleží tedy na použitém typu kódování. Datagram také obsahuje informaci o zřetězení jedné dlouhé zprávy do více (až 255) standardních. Nahrazuje tak nespolehlivost datagramového spojení, které samo o sobě není schopno zaručit doručení zpráv ve správném pořadí. První byte datagramu označený PDU-type obsahuje příznaky týkající se plátce zprávy, povolení potvrzení příjmu zprávy a další příznaky. [4] Balíčky programovacího jazyka Java pro mobilní telefony, které budou popsány níže, umožňují vložení osmi bitového identifikátoru Java aplikace do pole pro uživatelská data UD. [5]

# 2 Java ME

Java 2 Micro Edition je součástí větší rodiny platformy Java společnosti Sun Microsystems. Společnost Sun vznikla roku 1982 s hlavní vizí : „Zajistit, aby síťové služby byly k dispozici pro každého, kdekoli, kdykoli a při použití jakéhokoli zařízení.“ [6] Z uvedeného citátu vychází i filosofie programovacího jazyka Java, který byl představen 23. května 1995.

Jedná se o objektově orientovaný jazyk, jehož hlavními přednostmi jsou přenositelnost programů, jednoduchost, distribuovatelnost, bezpečnost atd. Nevýhodami jsou pomalejší start programů a větší paměťová náročnost. Java se dělí do čtyř rodin, podle systémů, pro které je určena (viz. tab. 2.1).

| | |
|---|---|
| Java EE (Enterprise) | určena pro rozsáhlejší systémy (podnikové sítě) |
| Java SE (Standard) | pracuje na stolních počítačích |
| Java ME (Micro) | určená pro domácí zařízení s malou pamětí |
| JavaCard | určená pro „chytré“ karty do bankomatů apod. |

Tab. 2.1: Rodiny programovacího jazyka Java

Původně byla Micro Edition navržena pro domácí zařízení (např. televizory), ale dlouho zde nenašla uplatnění. V televizorech se začíná používat až v současnosti s nástupem digitální televize a s ní spojených interaktivních aplikací. Nejvíce se však rozšířila mezi malá mobilní zařízení, ať už PDA, nebo mobilní telefony. Vznik této Java rodiny byl nutný z hlediska malé paměti a nízkého výpočetního výkonu mobilních zařízení, protože Standard Edition se stále rostoucím počtem balíčků knihoven není v malých zařízeních použitelná - JavaME je zmenšenou verzí JavaSE.

Micro Edition musí oproti Standard Edition řešit navíc i jeden specifický problém. Zatímco u klasických PC je počet hardwarových zařízení omezený (klávesnice, monitory, procesory), u malých zařízení používáme nepřeberné množství hardwarových prostředků – od displejů, klávesnicí, dotykových zařízení až po dálkové ovladače atd. A v každé z těchto kategorií existuje ještě spousta

pod-kategorií. Například u displejů malých zařízení neexistuje jednotná označení ve velikosti úhlopříčky v palcích, ale v podstatě co telefon, to jiný druh displeje, nepočítaje displeje u různých audio soustav, či obrazovky televizí. Proto nemá JavaME jedinou specifikaci, jak tomu je u Sandard Edition, ale dělí se na různé konfigurace a profily.

Konfigurace určuje základní sadu knihoven a vlastností přístroje (podobný hardware apod.). Není povolené ji nadále rozšiřovat výrobci, jež ji implementují do svých zařízení. V současnosti jsou definovány dvě konfigurace : CLDC (Connected Limited Device Configuration) a CDC (Connected Device Configuration).

## 2.1 CLDC

Konfigurace CLDC je určena pro low-end zařízení s velmi malou pamětí a nízkým výpočetním výkonem. Specifikuje pouze minimální obsah paměti a to 160kB trvalé paměti (ROM) a 32kB paměti s náhodným přístupem (RAM). Byl také zjednodušen proces verifikace aplikací, protože by bylo hardwarově náročné aplikaci překládat a ověřovat správnost „bajtkódu“ po každém spuštění. Verifikace se tedy rozdělila na dvě části – pre-verifikace, kde se kód ověří a v případě správnosti se zapíše atribut StackMap. V druhé části (spuštění programu) se pak kontroluje, zda je tento atribut zapsán. Nad konfigurací CLDC jsou dva profily, které upřesňují nároky a vlastnosti přístrojů definovaných v konfiguraci CLDC : MIDP (Mobile Information Device Profile) a PDAP (Personal Digital Assistant Profile). PDAP je určen pro zařízení s větším displejem jako jsou zařízení PDA. Dále se budu věnovat pouze konfiguraci MIDP použité u mobilních telefonů.

Tento profil se v současnosti těší největší pozornosti vývojářů (již existuje verze MIDP 2.0), protože je určen pro malá mobilní zařízení, která jsou v současnosti velice rozšířena. K hardwarovým požadavkům přidává minimální velikost displeje 96x54, hloubku barev 1-bit, poměr horizontálních a vertikálních

pixelů přibližně 1:1, ovládání zařízení telefonní klávesnicí, QWERTY klávesnicí, nebo dotykovou obrazovkou, 128 kB (256kB) trvalé paměti pro MIDP implementaci, 8 kB trvalé paměti pro trvale uložitelná data MIDP aplikací, 32 kB (128kB) dočasné paměti, obousměrný síťový provoz s omezenou šířkou pásma, schopnost přehrávat tóny (hardwarově, softwarově). [7] Čísla v závorkách u paměťových požadavků jsou pro verzi MIDP 2.0, která se liší od MIDP 1.0 pouze v paměťových nárocích. Ke knihovnám specifikovaných v konfiguraci CLDC (viz. tab. 2.2) přidává MIDP 1.0 další knihovny (viz. tab. 2.3) a profil MIDP 2.0 vše doplňuje (viz. tab. 2.4). [7] [8]

| | |
|---|---|
| `java.lang` (z SE) | VM třídy a rozhraní |
| `java.io` (z SE) | vstup / výstupní operace |
| `java.util` (z SE) | standardní utility |
| `javax.microedition.io` | nové vstup / výstupní operace |

Tab. 2.2: Knihovny konfigurace CLDC

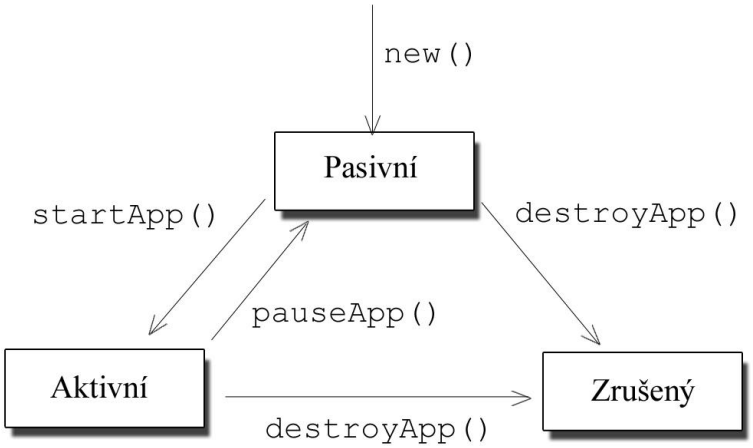
| | |
|---|---|
| `javax.microedition.rms` | správa trvalých dat |
| `javax.microedition.midlet` | obsahuje základní třídu MIDP profilu |
| `javax.microedition.io` | přidaná ke stejné knihovně v CLDC |
| `javax.microedition.lcdui` | pro tvorbu uživatelského rozhraní |

Tab. 2.3: Knihovny profilu MIDP 1.0

| | |
|---|---|
| `javax.microedition.lcdui.game` | vývoj her |
| `javax.microedition.media` | práce s médii |
| `javax.microedition.media.control` | přehrávání sekvencí |
| `javax.microedition.pki` | práce s certifikáty |

Tab. 2.4: Přidané knihovny profilu MIDP 2.0

Aplikace vytvořené pro tento profil se podle názvu profilu jmenují MIDlety. Některými vlastnostmi se podobá apletu (aplikace pro JavaSE). V telefonu běží na tzv. „sandboxu“, což by se dalo označit jako vlastní pískoviště, které nemůže z důvodu bezpečnosti opustit (viz. kap 5.2). [9]




Obr 2.1: Životní cyklus MIDletu

MIDlet se může nacházet pouze v předem definovaných třech stavech. Těmi jsou stav Pasivní, Aktivní a Zrušený. Běh aplikace a přechody mezi stavy má na starosti aplikační manažer. Na Obr 2.1 jsou zobrazeny zmíněné stavy a metody, které slouží k přechodu mezi stavy. Aplikace je vytvořena zavoláním konstruktoru bez parametrů a po vytvoření se dostane do pasivního stavu, kde nesmí vlastnit a používat žádné zdroje. Zdroje (Thread, Connection apod.) se inicializují až při přechodu do aktivního režimu metodou `startApp()`. Z obou stavů je možné se dostat do stavu Zrušený. V tomto stavu zavolá aplikační manažer metodu `destroyApp(boolean unconditional)` a podle jejího parametru buď aplikaci ukončí (true), nebo vyhodí výjimku (false) `MIDletStateChangeException`, čímž dá najevo, že aplikace ještě nechce být ukončena. [9]

## 2.2 CDC

Konfigurace CDC je určena pro zařízení s lepším hardwarovým vybavením, než konfigurace CLDC, tj. 32-bitový procesor, minimálně 512kB trvalé paměti a 256kB RAM. Tato konfigurace pracuje, stejně jako Stadard Edition a Enterprice Edition, na virtuálním stroji (CVM – Compact Virtual Machine) narozdíl od CLDC konfigurace pracující na kilobajtovém stroji (KVM – Kilobyte Virtual

Machine), což znamená, že na konfiguraci CDC jsou kladeny stejné funkční nároky jako na Standard Edition. Konfigurace CLDC je v podstatě podmnožinou CDC, která přidává navíc další knihovny. CDC také obsahuje více profilů (viz. tab. 2.5).

| | |
|---|---|
| Foundation Profile (FP) | přidává většinu základních tříd, které CDC chybí oproti standardní edici. Tvoří základ pro další rozšiřující profily |
| Personal Basic Profile (PBP) | základní uživatelské rozhraní |
| Personal Profile (PP) | Personal Java, která je rozšířena na spoustě zařízení |
| PDA Profile (PDAP) | Pro zařízení PDA |
| Imformation Module Profile | |

Tab. 2.5: Profily konfigurace CDC

## 2.3 Push Registry

Ve starém profilu MIDP 1.0 bylo možné aplikaci spustit pouze ručně pomocí manažeru aplikací AMS (Application Manager Software), který se stará o správu, spouštění a řízení běhu programů nainstalovaných v zařízení. Profil MIDP 2.0 k ručnímu způsobu přidává ještě další dva způsoby definované ve třídě `javax.microedition.io.PushRegistry` : jednak spuštění aplikace při příchozím spojení, a jednak spuštění ve stanovený čas. Těmto dvěma novým způsobům spuštění Java aplikace se říká Push Registry [21].

Kromě začlenění Push Registrů do manažeru aplikací bylo nutné začlenit je i do GFC (viz. kap.3). Manažer aplikací tak dostává od GFC zprávy, že došlo k přijetí spojení, poté si zkontroluje, jestli parametry příchozího spojení odpovídají pravidlům nastavených v Push Registrech. Pokud ano, aplikační manažer spustí v pravidlech určený MIDlet, jež si příchozí spojení obslouží sám. Java aplikace tak stojí mezi klasickými aplikacemi zařízení (Symbian, Windows Mobile atd.) a obsluhou sítových spojení GFC. Jinými slovy, pokud přijde spojení určené pro Javovskou aplikaci, Java má šanci toto spojení obsloužit dřív, než se

k němu dostanou ostatní programy. V konkrétním případě SMS zprávy určené pro daný JavaME program, přijmeme zprávu tímto programem, ale zpráva se nám již neobjeví v doručených zprávách telefonu. To byl jistě jeden z důvodů, proč je nutné v SMS zprávách specifikovat port Java aplikace (viz. kap.3).

Krátké textové zprávy mají v Push Registrech výjimku ve stylu přijímání spojení. Zatímco socketové a datagramové spojení je přijato jako proud znaků a musí tak být obslouženo ihned po příjmu, aby nedošlo ke zpoždění. U SMS spojení se data ukládají do vstupní paměti (bufferu) dokud není zpráva přijatá kompletně, teprve pak aplikační manažer spustí danou aplikaci.

Podle způsobu zaregistrování nového pravidla do manažeru aplikací AMS, rozdělujeme Push Registry na statické a dynamické. Statickým Push Registrem nazýváme takové pravidlo, které je zapsáno v JAD souboru, nebo v javovském manifestu (JAR souboru), případně v obou. Statickým je nazýván proto, že se Push Registry pravidlo nainstaluje do AMS zároveň s instalací programu do zařízení a manažer aplikací AMS bude vždy stejně reagovat na spojení specifikovaná zapsaným Push Registry pravidlem. Pravidlo zrušíme pouze odinstalováním aplikace ze zařízení. Instalátor aplikace pozná Push Registry pravidlo podle atributu MIDlet-Push v JAD, či JAR souboru – celé pravidlo obsahuje čtyři části :

**`MIDlet-Push-<n>`** : Atribut, pomocí kterého instalátor aplikace identifikuje, že se jedná o nové Push Registry pravidlo. Parametr <n> zastupuje pořadové číslo pravidla.

**`<SpojeníURL>`** : Řetězec určující URL spojení, na které bude manažer aplikací AMS reagovat. Skládá se z kódu spojení určující, zda se jedná o SMS, datagram, nebo socket spojení (pro SMS je to sms://) a z portu aplikace (viz kap.3). Při instalaci může dojít k chybě, pokud se snažíme zaregistrovat stejné URL se shodným portem aplikace pro dva různé MIDlety.

**`<NázevMIDletu>`** : Jméno MIDletu, který chceme spustit se specifikovaným příchozím spojením. Jedná se o plné jméno MIDletu, tzn. že název musí být u veden i s balíčky, ve kterých je uložen.

**<Filtr odesílatelů>** : Zde můžeme specifikovat adresy odesílatelů, od kterých dané spojení čekáme. Na jiné adresy nebude AMS reagovat. V případě socketového spojení se jedná o IP adresy, v případě SMS spojení mluvíme o telefonním čísle. Je zde povoleno používat hvězdičkové a otazníkové konvence, kde hvězdička nahrazuje skupinu znaků, otazník nahrazuje znak jediný. Pokud nechceme filtrovat žádné adresy, použijeme pouze hvězdičku, což znamená, že pravidlo platí pro jakéhokoliv odesílatele.

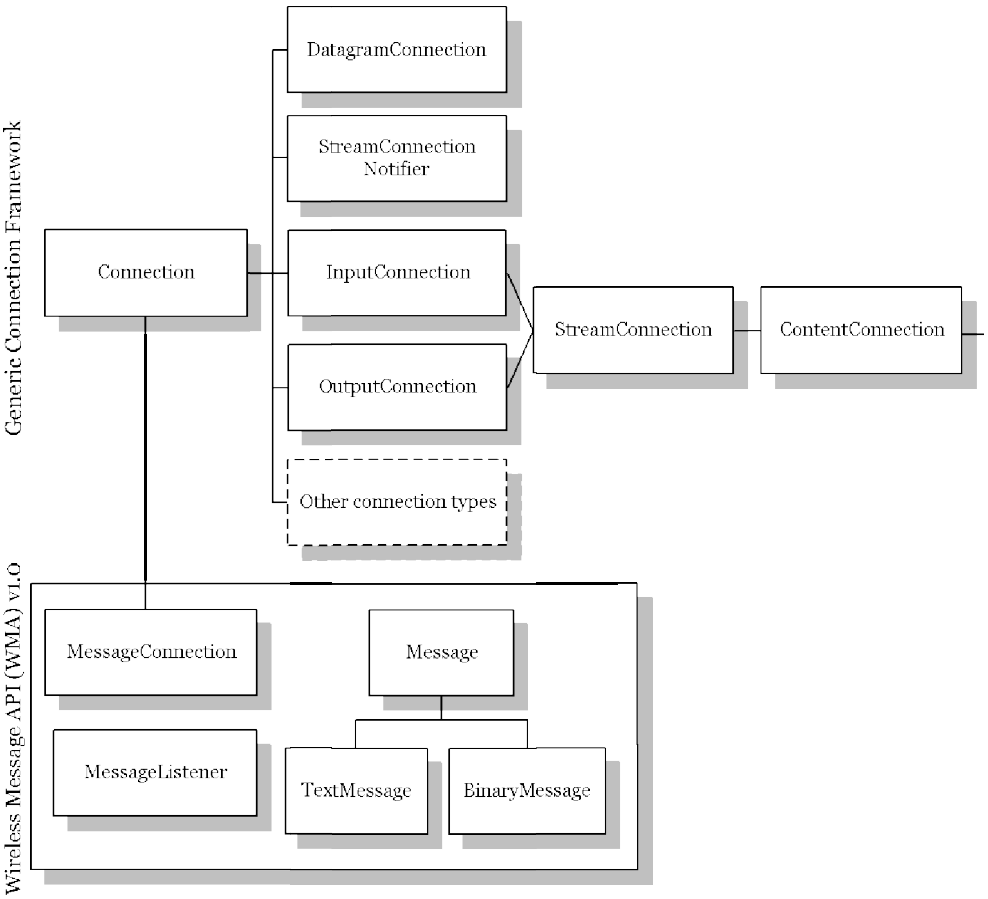
Statické pravidlo pro MIDlet „Zkouska“ v balíčku „Kubina.Tomas“ s portem „4576“ čekajícího na SMS spojení od kohokoliv, pak bude vypadat následovně : `MIDlet-Push-1 : sms://4576, Kubina.Tomas.Zkouska, *`

Dynamické Push Registry se od statických liší místem zapsání pravidla. Místo JAD, či JAR souboru zapíšeme pravidlo přímo do programu pomocí metody `PushRegistry.registryConnection()`. Metoda vyžaduje tři atributy, které jsou shodné s atributy popsanými v předchozím odstavci. Rozdíl je v tom, že se jedná o proměnné typu String, které můžeme v průběhu programu měnit, odsud název „dynamické“. Dále můžeme jednou zavedené pravidlo zrušit kdykoliv při běhu programu metodou `PushRegistry.unregisterConnection()`. Pomocí metody `PushRegistry.listConnections()` můžeme zjistit zaregistrovaná Push Registry pravidla, případně, které pravidlo vedlo ke spuštění daného MIDletu.

Rozdělení na statická a dynamická pravidla je možné pouze pro příchozí spojení. U časových spouštění MIDletů je povoleno registrovat pouze jedno pravidlo. Při použití statického pravidla bychom tak byli omezeni pouze jedním spuštěním. Časové pravidlo má tak smysl pouze tehdy, pokud je dynamické. Při spuštění MIDletu v daný čas musíme změnit parametry tohoto pravidla (například zjistit aktuální datum a čas a k tomuto přičíst určitou hodnotu). Nastavení časového pravidla děláme metodou `PushRegistry.registerAlarm()`, kde jako atributy uvádíme název kompletní MIDletu v proměnné typu `String` a dále datum a čas dalšího spuštění v proměnné typu `Date`. [22]

# 3 Wireless Messaging

Jelikož v profilu MIDP 1.0 nebyla zahrnuta Application Programing Interface (API) pro posílání SMS, museli výrobci do svých zařízení importovat speciální API (např.: Siemens API, Nokia UI), které dané funkce obsahovaly. Následující verze profilu MIDP 2.0 problém opět nevyřešila, proto vzniklo Wireless Messaging API (WMA), které ošetřuje posílání textových, binárních a smíšených zpráv. V současnosti existuje druhá verze WMA, jež původní verzi rozšiřuje.




Obr 3.1: Struktura WMA a jeho propojení s GCF

Jelikož JavaME nedědí žádné třídy z oblíbeného balíčku `java.net` rodiny J2SE, musela pro tuto rodinu Javy vzniknout speciální třída Generic Connection Framework (GFC) obsluhující připojení k síti. Balíček `javax.microedition.io` pak definuje strukturu, podporu definovaných vstupů/výstupů a jejich využití v J2ME profilech. Práci se síťovým připojením třídy GFC využívá ke své práci Wireless Messaging API jak je patrné na Obr 3.1.

Balíček `javax.wireless.messaging` je navržen tak, aby uměl pracovat s rozhraním typu `Message` nezávisle na typu zprávy (SMS, MMS). `Message` rozhraní obsahuje pouze základní funkce stejné pro všechny druhy zpráv obsluhovaných pomocí WMA.

Zprávu posíláme a přijímáme přes rozhraní `MessageConnection`, které aplikace otevře pomocí GCF. Otevřeno může být buď v klient, nebo server módu. Klientský mód lze použít pouze k posílání krátkých zpráv, zatímco v serverovém módu můžeme zprávy jak posílat, tak i přijímat. Otevřené spojení je definováno tzv. identifikátorem aplikace (např. číslo portu aplikace). Zpráva s identifikátorem dojde pouze aplikaci s příslušným portem. Číslo portu se ukládá do části s uživatelskými daty. To znamená, že namísto 140 bytů dat máme k dispozici o jeden byte méně.

| **Rozhraní :** | |
|---|---|
| `BinaryMessage` | Reprezentuje binární zprávy |
| `Message` | Základní rozhraní, z něhož se další odvozují |
| `MessageConnection` | Základní funkce pro posílání zpráv |
| `MessageListener` | Upozorňuje aplikace o došlých zprávách |
| `MultipartMessage` | Reprezentuje zprávy s více druhy dat |
| `TextMessage` | Reprezentuje textové zprávy |
| **Třídy :** | |
| `MessagePart` | Instance můžou být přidány do „multipart-message“ |
| **Výjimky :** | |
| `SizeExceededException` | Operace není proveditelná |

Tab. 3.1: Prvky WMA

Základní rozhraní, která Wireless Messaging obsahuje jsou `Message` a `MessageConnection`. Ostatní rozhraní vycházejí z rozhraní `Message`. Všechny třídy a rozhraní balíčku Wireless Messaging jsou v Tab. 3.1. V následujících podkapitolách budou podrobněji rozepsány ty rozhraní a třídy, které budou přímo využity pro praktický výstup diplomové práce, nebo jsou důležité z hlediska cílů tohoto projektu. [5]

## 3.1 Connector

Třída `Connector` slouží k vytvoření objektů typu `Connection`. Wireless Messaging ji využívá pro otevření, uzavření spojení a k posílání dat přes toto spojení.

**`open()`** : (`public static javax.microedition.io.Connection` **`open`**`(java.lang.String name)`) vytvoří a otevře spojení `Connection`. Parametr `name` je typu `String` a uchovává adresu URL (Uniform Resource Locator). Wireless Messaging API bude používat tento konstruktor. Dále je možné volat konstruktory s dalšími parametry jako jsou `mode`, `timeouts`.

## 3.2 MessageConnection

Rozhraní `MessageConnection` definuje základy pro posílání a příjem krátkých textových zpráv přes rozhraní popsané v předchozí podkapitole `javax.microedition.io.Connection`. Obsahuje metody pro poslání a příjem zprávy, dále pak metodu pro vytvoření nového objektu `Message`.

Na začátku volá třída metodu `Connector.open()`. Když spojení skončí, aplikace by měla zavolat i metodu `Connector.close()`, není to však povinností. Pouze tím přicházíme o možnost kontrolovat jiné metody pomocí výjimky `IOException`, která vznikne při volání `MessageConnection`

Spojení může být otevřeno buď v klient, nebo server módu. Jak jsme již uvedli, v klient módu můžeme zprávy pouze vysílat. Takové spojení otevřeme zavoláním metody `Connector.open()` a jako parametr jí pošleme pouze řetězec cílové adresy. Tato metoda vrací objekt `MessageConnection`. Server mód, ve kterém můžeme zprávy i přijímat, otevřeme zavoláním `Connector.open()` s atributem řetězce, jež definuje konečný bod aplikace na místním zařízení. Pokud je daný koncový bod již obsazen jinou aplikací, metoda otvírající spojení vyhodí výjimku `IOException`. Může nastat i případ, kdy koncový bod není obsazen, ale díky bezpečnostním opatřením bude znemožněno přes tento koncový bod posílat zprávy (více v kapitole o bezpečnosti). Je také možné, aby měla aplikace více instancí objektu `MessageConnection` ať už v klient, nebo server módu. Aplikace může dále implementovat rozhraní `MessageListener` a přidělit instanci objektu této třídy `MessageConnection`, aby mohla být aplikace upozorněna při příchodu nové zprávy. V tomto případě nebude vlákno blokováno.

**`newMessage()`** : (`public javax.wirelless.messaging.Message` **`newMessage`**`(java.lang.String type)`) Konstruktor vytvoří instanci objektu `Message` daného typu zprávy. Typ zprávy se uvádí v parametru a může nabývat hodnot `TEXT_MESSAGE`, `BINARY_MESSAGE` a `MULTIPART_MESSAGE`. Co který parametr znamená je patrné z jejich výstižného názvu. Podle typu parametru se také aktivuje příslušné rozhraní.

Pokud je tato metoda zavolaná v klient módu, automaticky se vyplní parametr cílové adresy (destination address). V server módu musí toto pole vyplnit aplikace před tím, než zprávu odešle.

V případě uzavření spojení tato metoda stále vrací instanci objektu `Message`. Můžeme taktéž použít konstruktor se dvěma parametry třídy String : `newMessage(java.lang.String type, java.lang.String address)`, které cílovou adresu nastaví v obou výše zmíněných případech.

**send()** : (`public void` **send**`(javax.wireless.messaging.Message msg)`) posílá zprávu. Při použití metody `send()` může vzniknout pět výjimek. Výjimka `IOException` vznikne v případě, kdy zpráva nejde odeslat z důvodů zavřeného spojení, nebo chybě v síti. `IllegalArgumentException` pokud není zpráva kompletní, obsahuje nesprávný formát dat, nebo délka zprávy přesahuje limit. `InterruptedIOException` je volána v případě vypršení časového limitu, nebo uzavření spojení v průběhu odesílání zprávy. Výjimka `SecurityException` nastává v případě, kdy nemá aplikace povolení posílat zprávy. A konečně `NullPointerException` pokud je parametr roven hodnotě `null`.

Dalšími metodami rozhraní `MessageConnection` jsou **`setMessageListener`**`(MessageListener)`, který přiřazuje instanci objektu MessageListener danému spojení MessageConnection, **`numberOfSegments`**`(Message)` počítající, do kolika zpráv bychom měli danou zprávu rozdělit (pokud je delší, než určuje limit) a metoda **`receive`**`()`. [5]

## 3.3 Message

Rozhraní `Message` je základním rozhraním pro odvozená rozhraní (`TextMessage` apod.). Navrženo je tak, aby pracovalo s objekty typu `Message`, které mohou v závislosti na odvozených rozhraních obsahovat rozdílné součásti. V uvedené analogii SMS zpráv s datagramy UDP protokolu je zde patrná odlišnost – zatímco datagram je pouze blokem binárních dat, SMS zpráva může obsahovat různé typy dat v závislosti na použitém odvozeném rozhraní. Další odlišností je způsob doručení zprávy. Zatímco se v případě UDP datagramů nestaráme, zda je cíl k dispozici (pokud není, datagram je zahozen), SMS zpráva čeká v servisním centru, dokud neuplyne doba její platnosti, nebo se účastník nepřihlásí do sítě. Instance objektů implementující toto rozhraní jsou pouze

kontejnery pro uživatelsky zadaná data. Rozhraní `Message` obsahuje metody `getAddress()`, `setAddress()`, `getTimestamp()`.

**`getAddres()`** : (`public java.lang.String` **`getAddress`**`()`) vrací adresu dané zprávy. Pokud se jedná o zprávu určenou k poslání, vrátí adresu příjemce. V případě přijaté zprávy vrátí adresu odesílatele, což je užitečné, když chceme na přišlou zprávu odpovědět. Zavoláme pouze metodu `getAddress()`, aniž bychom museli adresu vybírat přímo ze jmenného seznamu. Užitečné je to také z důvodu znovupoužití objektu. Pro odpověď nemusíme vytvářet novou instanci `Message` objektu. Když zpráva nemá přiřazenu žádnou adresu, metoda vrací `null`. Adresa má stejnou syntaxi jakou používá `Connector.open()` k získání `MessageConnection`. V případě poslání SMS zprávy uvádíme adresu ve tvaru „`sms://`“ plus číslo začínající znakem „+“. Pokud chceme s číslem uvést i identifikátor aplikace, napíšeme ho za dvojtečku (`sms://+420124456789:3333`).

Pro přidělování portů aplikacím existují pravidla. Která aplikace si číslo portu zabere jako první, ta ho má. V případě, že se aplikace snaží obsadit již obsazené číslo, aplikace vyhodí vyjímku `IOException`. To samé se stane v případě, pokud se aplikace snaží obsadit port používaný systémovým zařízením. Ve standardu GSM 03.40 jsou čísla portů rozděleny do rozsahů. Jeden rozsah má zabraný organizace IANA (Internet Assigned Numbers Authority), která čísla portů žadatelům přiděluje. Tvůrce aplikace si tak může zajistit, že pouze jeho aplikace bude působit na daném portu. Programátor má samozřejmě i na výběr z rozsahu volných adres portů, zde však musí doufat, že jím naprogramované číslo neobsadí před ním jiná aplikace. Dále ještě existují zakázaná čísla portů, která jsou přidělena už trvale určitým službám. Jejich seznam je uveden v [5] v příloze o GSM adaptéru. Nakonec se tu pouze zmíním o adrese servisního centra. V určitých případech ji bude aplikace potřebovat zjistit. Podrobnější informace jsou ve zdroji [5] opět v příloze o GSM adaptéru.

**`getTimestamp()`** : (`public java.util.Date` **`getTimestamp`**`()`) vrací čas odeslání zprávy. Formát časového údaje je shodný s formátem objektu `Date` třídy `java.util.Date`. [10]

**`setAddress()`** : (`public void` **`setAddress`**`(java.lang.String addr)`) Nastavuje adresu aktuální zprávy a může ji nastavit i na hodnotu `null`. Adresa musí mít stejnou syntaxi jako adresa zadávaná `Connector.open()` k získání `MessageConnection`.[5]

## 3.4 TextMessage

Rozhraní `TextMessage` je odvozeno z rozhraní `Message` a má za úkol reprezentovat textovou zprávu. V podstatě se stará pouze o získání textového řetězce z aktuální zprávy, nebo se textový řetězec snaží do zprávy vložit. Instance objektů implementujících toto rozhraní jsou tak pouze kontejnery pro uživatelská data. Data tohoto rozhraní mají formát javovského `Stringu`. Ten se musí převést do formátu srozumitelného textové zprávě tak, že podle použitých znaků „překladač“ pozná, který z protokolů má použít. Aplikace by však měla používat znaky, které jsou překladatelskými protokoly známy, jinak by znaky nemusely být přeloženy správně.

Aplikace je dále zodpovědná za délku uživatelských dat. Aby došlo k řádnému odeslání zprávy, nesmí délka dat přesahovat kapacitu zprávy. V opačném případě vyhodí metoda `MessageConnection.send()` výjimku `IllegalArgumentException`. K rozdělení velké zprávy do více menších zpráv pomáhá metoda `MessageConnection.numberOfSegments(Message)` počítající, do kolika zpráv bude třeba uživatelská data uložit. O samotné rozdělení se musí starat sama aplikace.

**`getPayloadText()`** : (`public java.lang.String` **`getPayloadText`**`()`) vrací data uložená v aktuální zprávě, nebo hodnotu `null`, pokud zpráva žádný text neobsahuje.

**`setPayloadText()`** : (`public void` **`setPayloadText`**`(java.lang.String data)`) ukládá do aktuální zprávy text. V případě, že chceme obsah uživatelských dat vymazat, uložíme pomocí této funkce do zprávy hodnotu `null`. [5]

# 4 JavaME aplikace

## 4.1 Popis funkce

Jádro aplikace spočívá v použití balíčku Wireless Messaging API (viz. kap.3) k příjmu a posílání krátkých textových zpráv. Aplikace funguje stejně jako klasický program na správu SMS zpráv, který je v každém telefonu. Umožňuje posílat zprávy, přijímat zprávy, rozlišit přečtené a nepřečtené přijaté zprávy, dále umožňuje listovat přijatými i odeslanými zprávami, každou zprávu si prohlédnout a buď ji přeposlat s vyplněným telefonním číslem a textem, nebo na ni odpovědět. Odeslání zprávy je začleněno do hlavní třídy MIDletu. Spojení pro odeslání zprávy se otevře pouze při uživatelském požadavku se specifickými parametry spojení (různá telefonní čísla příjemců). Spojení se zavře hned po úspěšném odeslání zprávy, aby nezabíralo omezené prostředky mobilního zařízení. Na obr 4.1 vidíme nejdůležitější části kódu pro otevření spojení, poslání zprávy a uzavření spojení. Nejprve je nutno zformátovat adresu příjemce do tvaru `sms://cislo:port` řetězce `String` (01). Poté použijeme tuto adresu pro otevření spojení (02) definovaného adresou a vracející ukazatel na otevřené spojení (zde přetypované na `MessageConnection`). Funkcí `newMessage()` otevřeného spojení vytvoříme kontejner (03) pro novou zprávu daného typu (zde textová zpráva). Do tohoto kontejneru nahrajeme text zprávy typu `String` (04). Poté můžeme již zprávu odeslat (05) a zavřít spojení (06).

```
(01) adr  = "sms://" + cislo + ":" + port;
(02) spoj = (MessageConnection) Connector.open(adr);
(03) zpr  = (TextMessage) spoj.newMessage(spoj.TEXT_MESSAGE);

(04) zpr.setPayloadText(text);

(05) spoj.send(zpr);

(06) spoj.close();
```

Obr. 4.1: otevření a zavření spojení pro odeslání

Příjem zprávy je řešen spuštěním programového vlákna při startu aplikace, protože je nutné, aby nedošlo k zastavení programu při čekání na zprávu a zároveň je nutné, aby bylo příchozí spojení neustále hlídáno. Použití vláken je elegantní řešení, jak splnit obě podmínky. Ke spuštění vlákna dojde pokaždé při příchozím spojení, bylo tedy nutné ošetřit přístup více vláken k jednomu globálnímu zdroji pomocí synchronizovaných sekcí, které zajišťují, že k danému bloku kódu (proměnné) přistupuje pouze jedno vlákno (viz. obr.4.2). Další vlákna čekají, než první vlákno opustí synchronizovaný blok. Je to důležité hlavně při ukládání zprávy do permanentní paměti, kde by mohlo dojít ke kolizi, nebo uložení dat na špatné místo. Na obr. 4.2 vidíme překrytí abstraktní metody `notifyIncomingMessage()` rozhraní `MessageListener` (01-03), která se spustí pokaždé, když na otevřené spojení přijde nová zpráva. Po příjmu nové zprávy spustíme vlákno (04-06), které má za úkol přišlou zprávu ošetřit (uložit ji atd...). Může však dojít k příjmu více zpráv na jednou a tedy ke spuštění více vláken. Synchronizovaného přístupu ke zdrojům je dosaženo klíčovým slovem `synchronized` (07) u příslušné metody, která má za úkol uložení příchozí zprávy do databáze, tedy přistupuje ke globálním zdrojům, ke kterým mají přístup i případná ostatní vlákna a mohlo by tak dojít ke kolizi.

```
(01) public void notifyIncomingMessage(MessageConnection sp){
(02)      new Thread(this).start();
(03) }

(04) public void run() {
(05)      prijmiZpravu();
(06) }

(07) synchronized public void prijmiZpravu() {
(08) .....
(09) }
```

Obr. 4.2: Synchronizace zdrojů u příjmu zprávy

K uložení dat, která mají být uchována i při příštím spuštění aplikace (nastavení programu, přijaté a odeslané zprávy, výsledky hlasování) bylo použito Systému správy záznamů RMS (Record Management System). Jedná se o

jednoduchou formu databázového systému (třída `javax.microedition.rms`), který umožňuje využít permanentní paměť přidělenou každému MIDletu. Po vytvoření/otevření databáze pak můžeme vkládat nové záznamy, přepisovat staré, číst záznamy a mazat je.

V aplikaci je použit neobvyklý způsob pro ukládání dat (viz. obr. 4.3). Místo toho, aby se mazané záznamy skutečně mazaly a nové znovu vytvářely, je v databázi vymezeno místo pro daný počet dat a nová data jsou na těchto místech pouze přepisována. Děje se tak proto, že RMS při vymazání záznamu už navždy obsadí ID tohoto záznamu a nově vytvořený záznam má ID o jednu větší, než posledně obsazený. Aby bylo možné použít ID databáze tak, jak jdou skutečně za sebou a použít tak cyklus `for` pro přístup k databázi, je nejdřív testováno, zda je v databázi další potřebné ID již alokováno. Pokud ano, hodnoty na tomto místě se pouze přepíšou, pokud ne, alokuje se nové místo, které se poté obsadí uživatelskými daty (viz. obr 4.3). Aplikace si to hlídá pomocí dvou globálních proměnných (01). `Posledni` v sobě uchovává číslo posledního alokovaného záznamu v databázi, `aktualni` zase pořadové číslo právě ukládané zprávy do databáze. Číslo `aktualni` může být buď menší, nebo rovné číslu `posledni`. V druhém z těchto případů dochází právě k alokování dalších míst v databázi a zvýšení čísla `posledni` o jednotku. `Aktualni` se zvýší o jednotku při každém uložení zprávy (06) a sníží o jednotku při každém vymazání zprávy z databáze. `Posledni` se nikdy nesnižuje. Obě tyto globální hodnoty jsou v aplikaci uloženy do databáze nastavení sloužící k ukládání počátečních hodnot při spuštění aplikace. Elegantnějším způsobem by bylo alokování všech míst na začátku při prvním spuštění aplikace v telefonu. Při testech se toto ovšem neosvědčilo, protože telefon na delší dobu „zamrznul“, než stihl najednou alokovat potřebný paměťový prostor. Každá zpráva zabírá 4 paměťová místa tzn., že při maximálním počtu 20ti uložitelných zpráv se bude alokovat 160 paměťových míst (80 pro přijaté zprávy, 80 pro odeslané). Zde se právě projevily ony omezené výkonové prostředky malých zařízení, se kterými se musí při návrhu aplikace počítat. I když tedy šlo o elegantnější řešení, pro návrh aplikace muselo být zvoleno řešení

z obr. 4.3, protože delší znehybnění aplikace není z uživatelského pohledu vhodné. Mohlo by působit dojmem, že je aplikace vadná

```
(01) if(posledni == aktualni) {
(02)    alokovaniPotrebnychMistVDatabazi;
(03)    posledni++;
(04) }

(05) ulozeniDatDoDatabaze;
(06) aktualni++;
```

Obr. 4.3: ukládání do databáze s případným otevíráním nových záznamu

Poté, co se vykonají všechny příkazy v MIDletovské funkci `startApp()`, se program zastaví a čeká. Probudit ho může pouze aktivování některého z příkazů pomocí softwarového tlačítka zařízení, nebo příchozí SMS. Oboje akce jsou ošetřeny spuštěním abstraktních metod rozhraní, které musíme v programu překrýt vlastním kódem. Pro příchozí spojení se jedná o zavolání metody `notifyIncommingMessage()` rozhraní `MessageListener`, ve které voláme překrytou abstraktní metodu `run()` rozhraní `Runnable`, starající se o vlákna. Pro ošetření příkazů softwarových tlačítek je překryta abstraktní metoda `commandAction()` rozhraní `CommandListener`, ve které jsou ošetřeny všechny příkazy MIDletu (pro představu se pro tuto aplikaci jedná o ošetření padesáti pěti příkazů). Zde nastává základní větvení programu, zde se děje veškerá akce a volají se ostatní „pomocné“ metody.

Druhou funkcí programu je SMS hlasování. Jak bude popsáno v následující kapitole (viz. kap.5) z důvodu bezpečnostních opatření JavaME nebylo možné využít možností WMA k hromadnému odesílání zpráv, protože je pokaždé vyžadováno ruční uživatelské potvrzení pro přístup k rozhranní, které je otevřeno v serverovém módu. Naštěstí pro klientský mód toto omezeno není, takže program nevyžaduje potvrzení pro poslech příchozích spojení na síťovém rozhranní telefonu. Je tak možné přijmout jakoukoliv SMS určenou dané aplikaci, bez uživatelova potvrzování. Výborně se to hodí pro SMS hlasování, které známe z velkých televizních soutěží. V aplikaci je možné začít až deset hlasování, která

uživatel specifikuje třímístným kódem hlasování a otázkou, na kterou je možné odpovědět „ano“, nebo „ne“. Hlasující pak pošle na telefonní číslo terminálu, na kterém hlasování běží, textovou zprávu, jež bude ve formátu `<třímístný kód><mezera><ano/ne>`. Je samozřejmě nutné, aby zprávu poslal z aplikace, která mu vyplní port aplikace shodný s portem aplikace sloužící jako terminál hlasování. V opačném případě by daná zpráva nebyla odchycena javovskou aplikací, ale standardním programem obsluhujícím textové zprávy v telefonu. Program pak počítá počet ano/ne v daném hlasování, umožňuje si prohlédnout výsledky a umožňuje hlasování zastavit tak, že program sice SMS zpracuje (nedojde k její převzetí klasickou aplikací telefonu), ale nepřičte již hlas. Také je důležité říci, že zprávy, které obsahují hlas, nejsou ukládány do Přijatých zpráv této aplikace.

V případě, že by v telefonu již byla aplikace, která by používala port používaný touto aplikací , je možné v nastavení změnit jak port pro výstupní zprávy, tak port pro vstupní zprávy (po změně je ukončeno naslouchání na starém spojení, a je otevřeno nové spojení na nové portu). V příloze je uveden vývojový diagram aplikace.

## 4.2 Popis prostředků

K vývoji aplikace bylo využito programovacího prostředí NetBeans v.6.0.1 [14]. Pro testování pak prostředí Wireless Toolkit 2.5.2 pro CLDC.

Pro návrh mobilních aplikací nabízí NetBeans grafický nástroj, kde si pouze vytvoříme jednotlivé obrazovky zobrazované na mobilním telefonu, přidáme k nim příkazy a tyto příkazy spojíme s obrazovkami, na které mají aplikaci přepnout při stisknutí softwarového tlačítka. Jedná se o rychlý návrh aplikace, bohužel je nutné říci, že při složitější aplikaci, jakou je tato v diplomové práci, není tento prostředek příliš vhodný. Jednak se díky automatickému generování kódu postupně ztrácí kontrola nad napsaným kódem, a také automaticky

generovaný kód obsahuje spoustu vedlejších komentářů, které jsou matoucí. Nejhorší vlastností grafického návrhu však je, že vytváří příliš mnoho vnořených metod, u kterých je pak těžké sledovat funkci, případně navazovat programátorův kód. Vede to také k velké nadbytečnosti kódu. Možnosti tohoto nástroje byly nakonec pro použití v diplomové práci zavrženy a každá obrazovka MIDletovské aplikace byla naprogramována ručně, což dovolovalo plnou kontrolu nad celou aplikací. Samotné psaní v NetBeans se ničím neliší od ostatních programovacích prostředí, není ho tedy nutné podrobněji popisovat.

Nová verze NetBeans v sobě již obsahuje integrovaný mobilní balíček Mobility Pack, jehož funkce byla popsána v předchozím odstavci (grafický návrh aplikace). U dřívějších verzí NetBeans bylo nutné balíček mobilních utilit do NetBeans ručně nainstalovat. Na webových stránkách výrobců mobilních zařízení bývají i moduly pro balíček Mobility Pack podporující konkrétní telefony.[16] Respektive tyto moduly říkají vývojovému prostředí, kterou konfiguraci a přídavné balíčky má konkrétní telefon k dispozici. Usnadnění spočívá v otestování dané aplikace pro konkrétní telefon, přímo v počítači, bez nahrávání programu do telefonu (nemluvě o možnosti testování dané aplikace na velice širokém spektru zařízení).

Jelikož je JavaME poměrně mladá technologie, nese s sebou i několik nedostatků. Java byla udělána s cílem, aby aplikace šla na co největším počtu zařízení. I když má telefon implementovány požadované balíčky Javy, nemusí aplikace na takovém zařízení běžet, jelikož se všechny implementace stoprocentně neshodují. Dále je problémem využívat v aplikaci širokého spektra implementovaných balíčků. Čím více balíčků totiž budeme v aplikaci využívat, tím více se vystavujeme riziku, že potřebná kombinace nebude implementována v mobilním telefonu. [17]

Proč tedy každý telefon obsahuje jiné balíčky? Proč nemůže každý telefon obsahovat všechno? Výrobce telefon dělá s určitým cílem. Některý je určen pouze pro volání a psaní zpráv, jiný i pro poslech hudby, další může být speciálně určen pro mladší generaci na hraní her a v neposlední řadě existují telefony určené pro multimediální používání. Jelikož jsou hardwarové prostředky omezené, nevyplatí

se investovat do větších pamětí pro implementaci všech JavaME balíčků, protože si telefon stejně koupí určená cílová skupina, pouze pro daný účel [18].

Například telefon určený pro široké multimediální využití - Nokia N95 – obsahuje velký počet implementovaných balíčků [19]. Vedle konfiguračního balíčku CLDC 1.1 a balíčku obsahujícího profil MIDP 2.0 (oba jsou obsaženy v JTWI API – JSR 185). Dále obsahuje Mobile Media API (JSR 135), které slouží jednoduchému přístupu k multimediálním zdrojům. Web Services Specification (JSR 172) určující optimální spojení k webovým službám. Security and Trust Services API (JSR 177) definující bezpečnostní služby. Location API (JSR 179) umožňující programátorům psát aplikace využívající informace o poloze (navigace apod.). SIP API (JSR 180) umožňující práci s protokoly SIP a dovolující tak pomocí Java aplikace volat z telefonu skrz internet. Mobile 3D Graphics (JSR 184) podporující 3D grafiku. Wireless Messaging API 2.0 (JSR 205), o kterém je tato práce. Scalable 2D Vector Graphics API (JSR 226) pro renderování 2D grafiky, práci s obrázky a využití 2D grafiky v aplikacích. Advanced Multimedia Supplements (JSR 234) pracující s pokročilými multimediálními službami. Staví na taktéž implementovaném JSR 135. FileConnection and PIM API (JSR 75) umožňující přístup k souborům. Bluetooth API (JSR 82) pracuje s Bluetooth rozhraním. [17] Nokia UI API obsluhující zvuk. Acces Permissions API opět pracující s bezpečnými přístupy ke zdrojům. [20]

# 5 Bezpečnost JavaME

Obecně je v Javě kladen na bezpečnost velký důraz. Z důvodů omezených zdrojů malých zařízení, je koncept ochrany zjednodušen. Bezpečnost se zde řeší na třech úrovních. Jsou jimi nízkoúrovňová bezpečnost (bezpečnost na virtuálním stroji), bezpečnost na úrovni aplikace („pískoviště“, důvěryhodnost MIDletů) a end-to-end bezpečnost.

## 5.1 Nízkoúrovňová bezpečnost

Nízkoúrovňová bezpečnost zaručuje, že budou spuštěny pouze ty aplikace, jež sémantikou odpovídají programovacímu jazyku JavaME, a které nemohou poškodit zařízení. Jak již bylo řečeno výše, javovské aplikace jsou při prvním spuštění preverifikovány. Při každém následujícím spuštění se pouze ověří přítomnost verifikačních atributů, a pokud tyto nejsou nalezeny, je aplikace odmítnuta.

## 5.2 Pískoviště a důvěryhodnost MIDletů

Na úrovni aplikace se pohybujeme v tzv. sandboxu (pískovišti), který je podobný jako ten u apletů. MIDlety díky sandboxu nemohou přistupovat k informacím ze zařízení, jakými jsou například telefonní seznam, nebo kalendář. Není tedy možné cokoliv změnit a v tom případě ani poškodit. Ve svém sandboxu má každý MIDlet pouze vyhrazené úložiště (viz kap.2.1), kde je možno ukládat, či data měnit. [11] Prostředí sandboxu musí splňovat několik podmínek. Jednak musí být soubory `.class` správně preverifikovány, poté musí stažení, instalaci a provoz aplikace zajišťovat automatický mechanizmus, který programátor nemůže ovlivnit. Další podmínkou je, že aplikace má k dispozici pouze knihovny přidané výrobcem zařízení a nemůže nahrát žádnou další knihovnu rozšiřující přístup

k nativním funkcím poskytnutých výrobcem zařízení. Aplikace nesmí ani předefinovat, či přidávat nové třídy.

Druhým bezpečnostním prvkem MIDP 2.0 na aplikační úrovni je tzv. důvěryhodnost MIDletů. Narozdíl od předchozích druhů bezpečností, které zaručovaly bezpečnou funkci zařízení a předcházely tak smazání, nepovolenou editaci dat, případně získávání osobních dat z telefonu a posílání těchto dat ven po síťovém rozhranní, nám důvěryhodnost MIDletů chrání především finance za operátorem poskytované služby. Důvěryhodným se MIDlet stává tehdy, pokud je podepsaný vyšší autoritou a jako takový má umožněn přístup k rozhraním, která jsou považována za citlivá.

Jako citlivá rozhranní jsou označena všechna ta, která se připojují k síti, ať už se jedná o datagramové spojení, TCP spojení, nebo posílání krátkých textových zpráv (viz tab. 5.1) - tedy služby, které jsou operátorem zpoplatněny. Pokud bychom přístup na tato rozhranní žádným způsobem nehlídali, mohlo by se stát, že by nevinně se tvářící aplikace mohla v pozadí posílat krátké textové zprávy na draze zpoplatněná čísla a tak „vydělávat“ tvůrci aplikace, případně v pozadí posílat důvěrná data. V tab.5.1 jsou uvedeny třídy, ke kterým mají přístup nedůvěryhodné a důvěryhodné MIDlety bez povinnosti potvrzování. V tab.5.2 jsou oproti tomu uvedeny třídy, pomocí kterých přistupujeme k citlivým rozhranním a potřebujeme určitý stupeň uživatelského potvrzení.

| | |
|---|---|
| `javax.microedition.rms` | ukládání perzistentních dat |
| `javax.microedition.midlet` | životní cyklus aplikace |
| `javax.microedition.lcdui` | uživatelské rozhranní |
| `javax.microedition.lcdui.game` | herní rozhranní |
| `javax.microedition.media` | přehrávání médií |

Tab. 5.1: Bezpečné třídy

| | |
|---|---|
| javax.microedition.io.Connector | otevření spojení |
| javax.microedition.io.Connector.http | http spojení |
| javax.microedition.io.Connector.https | bezpečné http |
| javax.microedition.io.Connector.socket | TCP spojení |
| javax.microedition.io.Connector.datagram | datagramové spoj. |
| javax.microedition.io.Connector.ssl | zabezpečení |
| javax.microedition.io.Connector.serversocket | spojení se serverem |
| javax.microedition.io.Connector.datagramreciever | příjem |
| javax.microedition.io.Connector.comm | komunikace |
| javax.microedition.io.PushRegistry | Push Registry |
| javax.wireless.messaging | posílání SMS zpráv |

Tab. 5.2: Potenciálně nebezpečné třídy – přístup pouze s povolením

Pro přístup k citlivým rozhraním existují pro důvěryhodné MIDlety tři možnosti. Buď musí uživatel přístup k citlivému rozhranní povolit při prvním přístupu aplikace k rozhranní, nebo musí uživatel povolit přístup při každém spuštění MIDletu a jeho prvním přístupu na rozhranní a nebo musí uživatel povolit pokaždé, kdy se program pokouší k danému rozhranní připojit. Tab. 5.3 a 5.4 ukazují rozdíly, pro konkrétní telefon Nokia N95 [23], mezi jednotlivými citlivými rozhranními, pokud je aplikace nedůvěryhodná (viz tab.5.3) a důvěryhodná (viz tab.5.4). Stav po nainstalování aplikace do telefonu je v tabulkách označen jako „implicitně“, tento stav můžeme v manažeru aplikací AMS změnit na stavy, které jsou v tabulce označeny jako „ano“, oproti tomu nikdy nemůžeme aplikaci přepnout do stavu označeného heslem „ne“. Stavy u nichž je napsáno „nespec.“ nejsou pro daný telefon specifikovány, programátor tedy musí otestovat chování aplikace u konkrétního druhu telefonu.

| | bez přístupu | zeptat se vždy | zeptat se napoprvé | povoleno |
|---|---|---|---|---|
| Internet | ano | ano | implicitně | ne |
| SMS | ano | implicitně | ne | ne |
| Push Registry | ano | ano | implicitně | ne |
| Multimédia | ano | implicitně | ano | ne |
| Čtení uživ. dat | ano | implicitně | ne | ne |
| Editace uživ. dat | ano | implicitně | ne | ne |
| Autentizace | nespec. | nespec. | nespec. | nespec. |
| Volání | nespec. | nespec. | nespec. | nespec. |

Tab. 5.3: Přístup k rozhranním pro nedůvěryhodný MIDlet

| | bez přístupu | zeptat se vždy | zeptat se napoprvé | povoleno vždy |
|---|---|---|---|---|
| Internet | ano | ano | implicitně | ano |
| SMS | ano | implicitně | ne | ne |
| Push Registry | ano | ano | implicitně | ano |
| Multimédia | ano | ne | implicitně | ano |
| Čtení uživ. dat | ano | implicitně | ano | ano |
| Editace uživ. dat | ano | implicitně | ano | ano |
| Autentizace | ano | ne | implicitně | ano |
| Volání | nespec. | nespec. | nespec. | nespec. |

Tab. 5.4: Přístup k rozhranním pro důvěryhodný MIDlet

Když porovnáme obě tabulky (viz tab.5.3 a 5.4), vidíme, že se ve všech případech, kromě jednoho, zlepší přístup k citlivým rozhraním, pokud je MIDlet důvěryhodný. Důvěryhodným se stane po podepsání certifikační autoritou. Po podepsání můžeme bez potvrzování surfovat po internetu, bez potvrzování nechat manažer aplikací AMS automaticky spouštět aplikace atd. Jedinou výjimkou je přístup k síťovému rozhranní zajišťujícímu posílání krátkých textových zpráv, kde

i po podepsání MIDletu musíme nutit uživatele potvrzovat přístup k rozhranní pokaždé, když bude chtít poslat zprávu.

O tom, zda je MIDlet důvěryhodný rozhodují tzv. ochranné domény (Protection Domains). Pro GSM/UMTS zařízení (tedy mobilní zařízení pro sítě druhé a třetí generace) existují čtyři ochranné domény – nedůvěryhodná, ověřená třetí stranou, ověřená operátorem, ověřená výrobcem. Kromě nedůvěryhodné, je každá ochranná doména vázána na sadu certifikátů, z toho plyne, že každé doméně může být přiřazeno více certifikátů, ale každý certifikát může patřit pouze jedné ochranné doméně. Pro podepsání MIDletu používáme veřejný klíč, díky kterému se aplikaci přiřadí daná ochranná doména [24].

Ochranná doména ověřená výrobcem obsahuje certifikáty patřící přímo výrobci. Ochranná doména ověřená operátorem se používá pro určení operátora telefonní sítě a používá certifikáty dostupné ze SIM karty telefonu. Ochranná doména ověřená třetí stranou používá certifikáty třetích stran – tzv. certifikačních autorit CA (Certificate Authorities‘). Pokud jsme se dříve bavili o důvěryhodných a nedůvěryhodných MIDletech, tak důvěryhodným byl MIDlet nazýván v souvislosti s podepsáním programu právě certifikační autoritou. Tab. 5.4 je tabulkou pro ochrannou doménu ověřenou třetí stranou. Více o certifikačních autoritách a procesu podepsání aplikace viz kap.5.4.

## 5.3 End-to-end bezpečnost

Bezpečností na nejvyšší úrovni je myšleno zabezpečení přenosů dat mezi aplikací a například serverem umístěným vně mobilního zařízení. Například v profilu MIDP 2.0 je takovým bezpečnostním prvkem umožnění bezpečného spojení pomocí HTTPS.

## 5.4 Certifikace MIDletu

MIDlet se stane důvěryhodným autentizací vlastníka aplikace a dále se díky této autentizaci MIDletu přidělí ochranná doména, která vlastníkovi umožní přístup k chráněným třídám (viz kap. 5.2). Celý mechanizmus podepisování a autentizace v JavaME je založený na infrastuktuře veřejných klíčů X.509 PKI (Public Key Infrastructure), jejíž konečný výstup je uložen do souboru JAR jako atribut. Infrastruktura veřejných klíčů X.509 využívá pro podpis šifrovací algoritmus RSA. Zařízení si pak podpis v JAR souboru ověří a v případě kladného výsledku následně dokončí autentizaci přidělením certifikátu k dané ochranné doméně.

Získání certifikátu je možné rozdělit do tří základních kroků. Nejdříve si vlastník aplikace musí vymezit požadavky, které na svoji aplikaci má. Například, pokud chce aplikaci pro zobrazování internetových stránek, musí si zjistit, jak zabezpečený je přístup pomocí JavaME k internetu. Tak zjistí, že by musel uživatel vždy po spuštění aplikace potvrdit, že se aplikace chce k citlivému rozhranní připojit. Dále je třeba zjistit, zda by podepsání aplikace něco změnilo. V tomto případě by po podepsání nemusel uživatel nic potvrzovat, takže by podepsání MIDletu jednoznačně smysl mělo (narozdíl od přístupu k SMS rozhranní, viz. kap.5.2). Po počáteční úvaze si vyhledá certifikační autoritu (viz. [27][28]) a pošle jí své klíčové údaje a veřejný klíč. Také je třeba brát na vědomí, že certifikace pomocí CA bude něco stát a je časově omezená (u [27] stojí certifikace aplikace zhruba 400 dolarů na jeden rok). Druhým krokem je vygenerování certifikátu certifikační autoritou a poslání tohoto certifikátu zpět k vlastníkovi a konečně třetím krokem je vložení certifikátu do aplikace.

## 5.5 Přístup WMA k síťovému rozhraní

Aby mohla aplikace posílat a přijímat zprávy, musí jí být uděleno povolení provést danou operaci. Mechanismy přidělování povolení jsou závislé na prováděné činnosti a jsou podrobněji rozebírány v jedné z předchozích podkapitol (viz. kap.5.2). První z takových činností je otevření samotného spojení. Běžící MIDlet zde vyžaduje přístup k metodám rozhraní `MessageConnection`. Pokud není tento přístup garantován, metoda `Connector.open()` musí vyhodit výjimku `SecurityException`. Pro poslání SMS zprávy musí být garantováno povolení `javax.microedition.io.Connector.sms`. Podobné je to i v případě povolení přijímat, či odesílat krátké zprávy. Zde musíme mít povolení `javax.microedition.io.Connector.sms.send` a `javax.microedition.io.Connector.sms.receive`. [5] Bohužel je bezpečnostní politika Javy natolik přísná, že přístup k SMS rozhranní je omezený jak v podepsané, tak v nepodepsané aplikaci. Omezení jsou tak velká (viz kap.5.2), že znemožňují normální využívání balíčku WMA zvláště pro skupinová odesílání SMS. Pokud bychom se rozhodli udělat aplikaci pro hromadné odeslání SMS, musíme otevřít dané spojení tolikrát, kolika příjemcům budeme hromadnou SMS posílat a vystavujeme tak uživatele aplikace nepříjemné povinnosti každý takový přístup k síťovému rozhraní ručně potvrdit.

Jako alternativní řešení ke skupinovému odeslání je tedy v aplikaci diplomové práce zvolena funkce „hlasování“ protože pro příjem zpráv v klientském módu (pouze příjem zpráv) není nutné přístup potvrdit a navíc se spojení musí znovu-otevřít pouze pokud změníme v nastavení aplikace jiný vstupní port. V jiném případě máme otevřeno pořád jedno spojení na kterém naslouchámе, zda přišla nová zpráva.

# Závěr

V diplomové práci byly dopodrobna probrány všechny možnosti programovacího jazyka JavaMe, se kterými se programátor setká při návrhu aplikace používající přístup k rozhraní posílající krátké textové zprávy.

Jedním z důležitých závěrů této práce je, že balíček Wireless Messaging API není tak mocným nástrojem, jak se na první pohled zdá. Zpočátku můžou programátora ohromit možnosti, které balíček nabízí. Zdá se, že by nebyl problém posílat hromadné zprávy a podle obsahu těchto zpráv očekávat od aplikací druhých stran zase příslušné automatické odpovědi, daly by se ve stanovený čas posílat zprávy informující druhou stranu o různých skutečnostech (pozice terminálu atd.). V podstatě cokoliv, co si člověk vymyslí v souvislosti s posíláním SMS. Proto i v zadání uvedené hromadné odesílání zpráv ve spolupráci s aplikací v telefonu by nabízelo ohromné možnosti, protože by si každý mohl psát zprávy v počítači a jednoduchým tlačítkem „odeslat“ by pak poslal žádost o odeslání do telefonu, který by danou funkci vykonal, jako jsou zvyklí uživatelé vyšších modelů telefonů Nokia. To vše uvedené v předchozích řádcích však v podstatě možné není, protože při vzniku aplikace narazíme na velká bezpečnostní omezení, která aplikaci nedovolí poslat zprávu bez uživatelova potvrzení.

Z toho důvodu byla do diplomové práce začleněna i kapitola o bezpečnosti jazyka JavaME, především pak o bezpečnostních omezeních v souvislosti s balíčkem Wireless Messaging API. Z této kapitoly je možné zjistit, že omezený přístup k rozhraní posílající SMS se nezlepší ani po certifikaci aplikace certifikační autoritou, protože jako na jediné rozhraní ze všech citlivých rozhraní, nemá certifikační proces na funkci konečné aplikace žádný vliv. Je nutné zmínit samozřejmě i důvody, které vývojáře jazyka JavaME vedly k těmto omezením. Jedná se především o předejití zneužití aplikace, která by mohla potají posílat zprávy na draze zpoplatněná čísla a vydělávat tak tvůrci aplikace nemalé peníze, než by uživatel zjistil, že byl „okraden“. Na jednu stranu jsou obavy vývojářů o

zneužití pochopitelné, na druhou stranu se jedná o takové omezení, že je skoro zbytečné Wireless Messaging API používat.

Vezměme si jako příklad aplikaci v zadání diplomové práce. Po zadání požadavku k odeslání hromadné zprávy napsané v počítači by uživatel musel stejně vzít mobilní telefon do rukou a každou ze zpráv jednotlivě potvrzovat, aby mohla být odeslána. Takovou aplikaci nebude nikdo používat.

Proto byl v diplomové práci, po konzultaci s vedoucím práce, opuštěn záměr udělat aplikaci posílající hromadné SMS z osobního PC, ale bylo nově zadáno, že se má podrobněji prozkoumat právě bezpečnostní politika JavaME a vytvořit aplikaci se zaměřením na hlasování. V tomto případě bylo cíle více než dosaženo, protože má uživatel možnost sám zahájit až 10 různých hlasování, sám je může charakterizovat příslušnou otázkou a třímístným kódem, díky kterému je možno z příchozích zpráv odlišit, zda se jedná o hlasování, případně kterému z hlasování je nutné přičíst hlas.

Aplikace pro hlasování ukazuje jedinou silnou stránku balíčku Wireless Messaging API a tou je alespoň částečně bezproblémový příjem krátkých zpráv, protože zprávy můžeme přijímat bez jakýchkoliv potvrzování. Pravděpodobně je to možné z důvodu, že ve většině zemí (výjimkou jsou například USA) není příjem krátkých textových zpráv zpoplatněn, nehrozí zde tedy žádné finanční úniky. Při zamyšlení se nad předchozími řádky je třeba však dodat, že abychom mohli zprávu bez potvrzení přijmout na daném portu, musí ji někdo poslat z JavaME aplikace, která příslušný port umí vyplnit, takže opět musí někdo ručně potvrzovat přístup na SMS rozhranní.

# Literatura

[1] ETSI, World Class Standards, http://www.etsi.org/WebSite/homepage.aspx, vyhledáno 10.12.2007

[2] KVAPIL, M., *Short Message Service*, http://www.penguin.cz/~mak/sms/sms.html, Matematicko-Fyzikální fakulta University Karlovy v Praze, Vyhledáno 10.12.2007

[3] NOVOTNÝ, V., *Odesílání zpráv v PDU* formátu, Vysoké učení technické v Brně

[4] MITOŠINKA, P., Transport krátkých textových zpráv SMS v síti GSM, http://mobil.idnes.cz/mob_tech.asp?r=mob_tech&c=A981217_0004277_mob_tech, vyhledáno 10.12.2007

[5] SUN MICROSYSTEMS, *Wirelles messaging API (WMA) for Java 2 Micro Edition (JSR 205 Expert Group)*, 2004

[6] SUN MICROSYSTEMS, http://cz.sun.com/, vyhledáno 10.12.2007

[7] ZELENÝ, J., MIDP 2.0 – Novinky, http://nb.vse.cz/~zelenyj/it380/eseje/xticm07/midp20.htm, Katedra informačních technologií, Vysoká škola ekonomická v Praze, vyhledáno 10.12.2007

[8] BITTNEROVÁ, Rút L., *Co vás zajímá o J2ME, ale báli jste se zeptat*, http://interval.cz/clanky/co-vas-zajima-o-j2me-ale-bali-jste-se-zeptat/, vyhledáno 10.12.2007

[9] BITTNEROVÁ, Rút L., *J2ME v kostce – první MIDlet*, http://interval.cz/clanky/j2me-v-kostce-prvni-midlet/, vyhledáno 10.12.2007

[10] SUN MICROSYSTEMS, *java.util (Class Date)*, http://java.sun.com/j2se/1.4.2/docs/api/java/util/Date.html, vyhledáno 10.12.2007

[11] DOUBEK, M., *Messenger pro mobilní telefony implementovaný v J2ME s vlastním Java serverem*, Bakalářská práce, Elektrotechnická fakulta Českého vysokého učení technického, Praha 2007

[12] BITTNEROVÁ, Rút L., *J2ME a bezpečnost*, http://interval.cz/clanky/j2me-a-bezpecnost/, vyhledáno 10.12.2007

[13] BITTNEROVÁ, Rút L., *J2ME kam kráčíš aneb MIDP 2.0 na obzoru*, http://interval.cz/clanky/j2me-kam-kracis-aneb-midp-2-0-na-obzoru/, vyhledáno 10.12.2007

[14] NETBEANS, http://www.netbeans.org/, vyhledáno 10.12.2007

[15] NETBEANS, *NetBeans mobility pack for MIDP/CLDC 5.5 Quick Start Guide*, http://www.netbeans.org/kb/55/quickstart-mobility.html, vyhledáno 10.12.2007

[16] NOKIA, *JavaME Developer's Library*, Forum Nokia, http://www.forum.nokia.com/info/sw.nokia.com/id/3cfc525e-f0ec-491c-badd-085c0e2df8bf/Java_ME_Developers_Library.html, vyhledáno 10.12.2007

[17] JAVA COMMUNITY PROCESS, *Java Specification's Requests*, http://jcp.org/en/jsr/all, vyhledáno 10.12.2007

[18] NOKIA, *Device Specifications*, Nokia Forum, http://www.forum.nokia.com/devices/matrix_all_1.html, vyhledáno 10.12.2007

[19] NOKIA, *Device Details (N95)*, Nokia Forum, http://www.forum.nokia.com/devices/N95, vyhledáno 10.12.2007

[20] NOKIA, *Java Security Domains*, Nokia Forum, http://wiki.forum.nokia.com/index.php/Java_Security_Domains, vyhledáno 10.12.2007

[21] SUN MICROSYSTEMS, *How can a MIDlet be launched automatically?*, http://developers.sun.com/mobility/midp/questions/pushregistry/, vyhledáno 4.3.2008

[22] SUN MICROSYSTEMS, *The MIDP 2.0 Push Registry*, http://developers.sun.com/mobility/midp/articles/pushreg/ vyhledáno 4.3.2008

[23] NOKIA FORUM, *API access rights on phones S60* 3rd, http://wiki.forum.nokia.com/index.php/API_access_rights_on_phones%2C_S60_3rd_FP1, vyhledáno 10.12.2007

[24] NOKIA FORUM, *MIDP 2.0: Signed MIDlet Developer's Guide v2.0*, Nokia Corporation, 2006

[25] HOUSLEY, R., *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, Internet engineering task force, 1999

[26] SUN MICROSYSTEMS, *Mobile Information Device Profile for Java 2 Micro Edition v.2.1 (JSR 118 Expert Group)*, 2006

[27] VERI SIGN, *Code Signing for Digital* IDs, http://www.verisign.com/products-services/security-services/code-signing/digital-ids-code-signing/index.html

[28] SUN MICROSYSTEMS, *Java Verified Program Guide v1.0*, 2004

[29] MAHMOUD, Q.H., *Naučte se Java 2 Micro Edition*, Grada Publishing a.s., Praha 2002