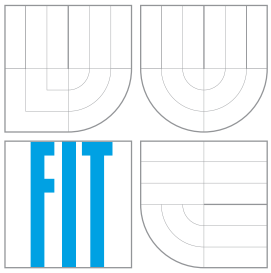
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYMBIAN S60

SYMBIAN S60

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN POKORNÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR CHMELAŘ

BRNO 2009

zde vložit zadání BP

zde vložit zadání BP

zde vložit zadání BP

Abstrakt

Symbian OS spolu se svými nadstavbami (uživatelské rozhraní, sada aplikací) tvoří softwarovou základnu pro mobilní telefony kategorie smartphone. Příkladem může být platforma S60, jíž se zaobírá tato práce. Účelem je na ukázkové aplikaci demonstrovat některé rysy vývoje pro tuto platformu spolu s prostředky pro přístup k relevantním druhům dat a pro provádění souvisejících operací. Aplikace umožňuje uživatelsky přizpůsobitelná upozornění na příchozí krátké textové zprávy (SMS) podle různých kritérií. Vytvořena je v jazyce C++ (se svými specifiky pro Symbian) pomocí vývojového prostředí (IDE) Carbide.c++ 2.0 a standardního balíku pro vývoj aplikací (SDK) pro S60 ve verzi 3rd Edition (počáteční verze a Feature Pack 1), orientovaných na tento programovací jazyk. Významná část je věnována teoretickému přehledu ohledně systému Symbian a platformy S60.

Abstract

Symbian OS with its additions (consisting of user interface, application suite) forms a software base for so-called smartphones. An example of this can be S60 platform, which is the aim of this thesis. Its purpose is to demonstrate, on an example application, some characteristics of the development for this platform as well as the means of an access to relevant sorts of data and the means of performing relevant operations. The application enables customizable notifications for incoming short text messages (SMS) according to various criteria. It is developed in (Symbian-specific) C++ with the use of C++ orientated development environment (IDE) Carbide.c++ 2.0 and standard development kit (SDK) for S60 3rd Edition (initial release and Feature Pack 1). Significant part is dedicated to a theoretical overview covering Symbian OS and S60 platform.

Klíčová slova

Symbian OS, platforma S60, smartphone, Nokia, Carbide.c++, SMS zpráva, uživatelsky přizpůsobitelné upozornění na příchozí zprávu, ukázková aplikace

Keywords

Symbian OS, S60 platform, smartphone, Nokia, Carbide.c++, SMS message, customizable notification of an incoming message, example application

Citace

Jan Pokorný: Symbian S60, bakalářská práce, Brno, FIT VUT v Brně, 2009

Symbian S60

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Chmelaře. Uvedl jsem všechny prameny, ze kterých jsem čerpal.

.....

Jan Pokorný
15. května 2009

Poděkování

Děkuji vedoucímu práce, panu Ing. Petru Chmelařovi, za podnětné připomínky a pomoc s praktickým testováním výsledné aplikace.

Obsah

Úvod **1**

1 Teoretické pozadí **2**
1.1 Symbian OS . . . 2
1.1.1 Historie Symbianu . . . 2
1.1.2 Vlastnosti systému . . . 3
1.1.3 Vlastnosti jádra . . . 4
1.1.4 Model systému . . . 4
1.2 Platforma S60 . . . 7
1.2.1 Nadstavby uživatelského rozhraní Symbianu . . . 7
1.2.2 Historie S60 . . . 8
1.2.3 Vlastnosti platformy S60 a jejího uživatelského rozhraní . . . 8
1.2.4 Technologie pro vývoj aplikací u třetí (páté) edice S60 . . . 10
1.3 Průřez programovacími koncepty a specifiky C++ pro Symbian/S60 . . . 11
1.3.1 Základní struktura C++ aplikace pro S60, forma její distribuce . . . 13

2 Vyhovění nárokům na aplikaci ze strany dostupného API **14**
2.1 Specifikace požadavků na aplikaci . . . 14
2.2 Uživatelské zprávy . . . 15
2.3 Sockety, příjem zpráv pomocí API „SMS Utilities“ . . . 16
2.4 Doručení zpráv do cílové složky . . . 19
2.5 Práce s databází kontaktů . . . 21
2.6 Přehrání zvuku . . . 22
2.7 Informace o aktuálním profilu . . . 22
2.8 Ovládání vibračního prvku telefonu . . . 23

3 Realizace demonstrační aplikace **24**
3.1 Přípravná fáze, vývojové prostředky . . . 24
3.2 Návrh aplikace . . . 25
3.3 Implementační detaily . . . 29

4 Závěr **32**

Seznam použitých zdrojů **33**

Rejstřík **38**

Seznam příloh **39**

A Uživatelská příručka **40**

Úvod

Vznik mobilních telekomunikačních sítí, zejména *GSM*[1], významně změnil pohled na běžně dostupné prostředky dorozumívání lidí na dálku. Oproti klasické telefonní síti odpadá fixování na tel. rozvody, spojení se sítí zajišťuje bezdrátové přenosné zařízení – mobilní telefon (dále také, v závislosti na kontextu, jen *telefon*, popř. *MT*). Při současné popularitě mobilní telekomunikace jej málokdo nepoužívá. Na základě různých požadavků spotřebitelů docházelo s postupným rozšiřováním funkcí MT k jejich rozdělení do několika kategorií.

Vznikl, mimo jiné, i segment „chytrých telefonů“, tzv. *smartphones*, což není přesně ohraničený pojem. Za obecný znak takových MT však můžeme považovat vybavení operačním systémem (dále také *OS*) a základní sadou aplikací s možností přidávat aplikace další, ať už je jejich vývoj dostupný všem, nebo jen omezenému okruhu partnerů výrobce telefonu/OS. Smartphony také zastanou roli zařízení *PDA*[2], o ty samostné tak opadl zájem.

Mezi ostatními stávajícími OS pro MT, jakými jsou *BlackBerry OS*, *Windows Mobile*, *iPhone OS*, případně i *Linux*, si první pozici (dle [9]) drží *Symbian OS* (dále také *Symbian*)[3]. Právě s tímto systémem — počítaje v to i jeho nadstavbu *S60 platform* (dále také *S60*, *platforma S60*) společnosti Nokia — je spjata tato bakalářská práce.

Jejím cílem je vhodným způsobem na konkrétní aplikaci demonstrovat možnosti tohoto systému, zejména ve vztahu k uživatelským datům a k systémovým událostem. Pro tento účel nabídnul vedoucí práce o aplikaci konkrétní představu, jíž jsem se také řídil (viz zadání práce, umístěné za titulním listem). Z možných technologií pro implementaci jsem zvolil jazyk *C++* jakožto nativní programovací jazyk pro Symbian.

Vlastní text je rozdělen do tří částí. V kapitole 1 je práci zasazena do kontextu vlastností Symbianu a jeho nadstaveb uživatelského rozhraní s těžištěm v platformě S60. Kapitola 2 představuje most mezi teorií a realizací demonstrační aplikace. Přibližuji v ní, jakých odpovědí se mi dostalo ze strany dokumentovace programovacího rozhraní pro C++ na otázky, jakým způsobem dostát dílčím požadavkům na aplikaci. Abych k tomu mohl přistoupit, musel jsem je napřed podrobněji specifikovat. Kapitola 3 se pak už přímo týká vývoje zmíněné ukázkové aplikace. To zahrnuje přípravnou fázi, kde se zabývám mj. vývojovými prostředky, dále fáze návrhu a implementace. V závěru (kap. 4) jsou pak diskutovány praktická funkčnost aplikace, její současný stav vývoje, související problémy a její případná budoucí rozšíření.

Za ním je seznam použitých zdrojů — v češtině je mi známa pouze kniha „Harrison: Programujeme aplikace pro Symbian OS“, avšak z důvodu ne úplné aktuálnosti a pro své nedobré zkušenosti s překlady minoritních publikací jsem se ji rozhodl vynechat a soustředit se pouze na zahraniční tituly a webové zdroje[4]. Následuje rejstřík, na samém konci je pak seznam příloh spolu s jedinou textovou přílohou, uživatelskou příručkou k vytvořené aplikaci.

[1] *Global System for Mobile communications*, standard pro mobilní komunikaci

[2] *personal digital assistant*, občas se používá český překlad *osobní digitální pomocník*

[3] výslovnost k poslechnutí např. na `<http://forvo.com/word/symbian/>`

[4] odkazy (většinou na web) ilustrativního charakteru umisťuji sem, do poznámek pod čarou

Kapitola 1

Teoretické pozadí

Přibližně stejný prostor je vyhrazen Symbianu a jeho nadstavbě S60. Narozdíl od těchto podkapitol je část o používaných konceptech a idiomech i o struktuře kódu pro nativní aplikace poměrně stručná, protože existuje mnoho materiálů, které se touto problematikou do detailu zabývají. U většiny odborných výrazů je uvedena česká i anglická varianta.

1.1 Symbian OS

Tato podkapitola vychází, kromě explicitně zmíněných pramenů, z následujících zdrojů (v textu jsou případně poznačeny také konkrétní strany):

- celá podkapitola: [4], [48]
- historie Symbianu: [10], [32], [34]
- vlastnosti systému, jádra a model systému: [6] (kap. [7]), [45]

1.1.1 Historie Symbianu

Symbian má svůj původ v projektu *EPOC*[1] od *Psion Software* (divize společnosti *Psion* specializovaná na vývoj operačního systému a aplikačního software), prvně vydanému v roce 1997. Byl to 32bitový OS pro vestavěné systémy na bázi *procesorů* (*CPU*[2]) *ARM*[3], napsán „od nuly“ v *C++*[4] jako následovník předchozích OS Psionu — 16bitového[5] a dvou generací 8bitových [4] (s. 19). Všechny tyto systémy byly používány v produktech Psionu (digitální organizéry, PDA), ovšem EPOC byl už také zamýšlen pro licencování dalšími stranami. Zaměření EPOCu na malá přenosná zařízení s omezenými výpočetními zdroji napájená bateriemi a systémem umístěným v *ROM*[6] předurčovalo možné zájemce.

Nejvýznamnějším počinem na tomto poli byl v roce 1998 vznik společnosti *Symbian*, sdružující Psion a výrobce MT *Ericsson*, *Motorola* a *Nokia* (později se přidala *Matsushita* a pak i někteří další), za účelem vývoje EPOCu pro v té době výhledově plánovaná zařízení

[1] zpětně označován jako *EPOC32*, kvůli jednoznačnosti — viz dále

[2] *central processing unit*

[3] 32bitové procesory s redukovanou instrukční sadou, viz `<http://en.wikipedia.org/wiki/ARM_architecture>`

[4] podotkněme, že v té době ještě nebylo C++ standartizováno, k tomu došlo v roce 1998, viz `<http://en.wikipedia.org/wiki/C++#Language_standard>`

[5] zpětně označován jako *EPOC16*

[6] *read-only memory*, označuje paměť jen pro čtení

s vlastnostmi MT i PDA (zmíněné smartphony). Po jeho uvedení ve verzi *EPOC Release 5* (*ER5*)[7], došlo k přejmenování projektu a příští verzí byl již Symbian OS 6.0.

Z pozdějších vývojových změn zmiňme alespoň zavedení bezpečnostního modelu pod označením *Platform Security* od verze 9 [8]. Zavádí sadu „oprávnění“ (*capabilities*), kdy v závislosti na jejich udělení konkrétní aplikaci mohou být této zpřístupněny jinak odepřené, chráněné funkce systému (především operace s jeho „choulostivými“ částmi). Taková oprávnění může v případě aplikace s méně náročnými požadavky na ně udělit přímo uživatel během instalace (pokud to výrobce daného MT umožní), v opačném případě musí být daný instalační soubor (tzv. *SIS*) digitálně podepsán. To se děje v rámci programu *Symbian Signed*, jehož styčným bodem je webový portál `symbiansigned.com` [43]. Ačkoli má především komerční ráz, tj. je orientován na výrobce placených aplikací, poskytuje možnosti i pro vývojáře *freeware*, *open source*[8] či vlastních pokusných aplikací. Zavedení Platform Security bylo, krom nového *ABI*[9], i hlavní příčinou porušení binární kompatibility mezi verzí 8 a 9.

V době psaní této práce je 100% vlastníkem společnosti Symbian Nokia [30]. Toho času nejnovější verzí je Symbian OS 9.5, avšak žádný jím vybavený MT zatím není na trhu.

1.1.2 Vlastnosti systému

Symbianu je úzce zaměřen na mobilní zařízení, primárně smartphony. Odtud plynou jeho cílová skupina v podobě běžných uživatelů („netechniků“), jeho hlavní poslání v provozování široké škály uživatelských aplikací a komunikačních služeb i akcent na grafické uživatelské rozhraní (*GUI*, dále také *UI*). Nelze opomenout ani potřebu úsporného provozu na baterie a podpory národních prostředí i lokalizace aplikací. Z hlediska uživatelů nelze tolerovat nestabilitu, systém by měl být schopen nepřetržitého provozu, bez restartu, i celé roky.

Symbian lze klasifikovat jako 32bitový jednouživatelský víceúlohový (*multitasking*) operační systém s obecně *preemptivním prioritním plánováním*[10] a ochranou paměti. Tradiční je i jeho návaznost na CPU ARM a uložení v ROM, kde také bývá obyčejně přímo spouštěn (technika *XIP*[11]). Základním souborovým systémem je *FAT*[12] (*DOS*ovské pojetí cest, např. `C:\sys\`), ovšem pro persistenci dat Symbian nabízí i prostředky vyšší úrovně [4] (s. 68–70).

Již od dob EPOCu byl systém kompletně podřízen *objektově-orientovanému* (dále také *OO*) *návrhu*, implementovanému v C++ (na nejnižší úrovni je místy použit i *asembler* a *C*). Typickým znakem je modularita na všech úrovních systému formou „zásuvných modulů“ (*plug-ins*) a používání tzv. *frameworků*[13]. Poskytování služeb a sdílení prostředků funguje na principu *klient–server*, přičemž „odběratelů“ je obecně více. V Symbianu je uplatňováno paradigma *řízení událostmi* (*event-driven/based*), hojné je *asynchronní* pojetí systémových aktivit, kdy svou roli sehrává mechanismus blízký návrhovému vzoru *observer*[14]. Sem zapadá např. model komunikace mezi klientem a serverem typu „žádost o službu – zpětné volání (tzv. *callback*) žadatele“. Struktura aplikací podléhá návrhovému/architektonickému vzoru *model–view–controller* (*MVC*)[15], přičemž izolovaný celek reprezentující aplikační data a

[7] resp. po verzi *ER5u*, kde ‚u‘ značí podporu pro Unicode

[8] *freeware* — nezpoplatněný software; *open source* — software s otevřeným zdrojovým kódem

[9] *application binary interface*, viz `<http://en.wikipedia.org/wiki/Application_binary_interface>`

[10] výjimku tvoří tzv. *aktivní objekty*, kdy je v kontextu jednoho vlákna zaručen nepreemptivní chod (viz podkap. 1.3, odrážka e) a některé úseky nanojádra (viz podkap. 1.1.3)

[11] *Execute in place*, způsob spuštění programu přímo z místa jeho uložení, viz `<http://en.wikipedia.org/wiki/Execute_in_place>`

[12] *File Allocation Table*, viz `<http://en.wikipedia.org/wiki/File_Allocation_Table>`

[13] softwarová/programovací struktura, viz `<http://en.wikipedia.org/wiki/Software_framework>`

[14] občas se používá překlad *pozorovatel*; viz `<http://en.wikipedia.org/wiki/Observer_pattern>`

[15] viz `<http://en.wikipedia.org/wiki/Model-view-controller>`

související operace, tzv. *model* (v tomto kontextu se označuje, coby výkonná jednotka, spíše jako *engine*), je víceméně přenositelný mezi jednotlivými UI nadstavbami [1] (s. 40–41).

1.1.3 Vlastnosti jádra

Předně, základní charakteristika *jádra* (*kernelu*) splývá s již uvedenými vlastnostmi systému jako celku. Původní jádro, zpětně označované jako *EKA1*[16], bylo od Symbianu verze 9 definitivně nahrazeno novým jádrem *EKA2*[16], které lépe odpovídalo potřebám svého nasazení, a k němu se bude také vztahovat další text (stejně jako k Symbianu verze 9.x).

Kernel (EKA2) činí ze Symbianu *operační systém reálného času* (přesněji *soft RTOS*[17]), tím pádem lze modul časově kritické GSM signalizace provozovat na stejném CPU jako zbytek systému a tak ušetřit pro tuto úlohu vyhrazenou výpočetní jednotku. Jádro je *vícevlákenné* (*multi-threaded*, viz [7] s. 13), i zde hraje roli preemptivní plánování (upřednostňovány jsou prioritní operace kernelu).

Co se však týče jeho zařazení (dle klasifikace, jak ji uvádí např. [46]), je situace obtížnější. Nejblíže má k architektuře *mikrojádra*, tedy základního minima podpůrných služeb (přístup do paměti, *IPC*[18], atp.), nad kterými se teprve buduje hlavní funkcionalita operačního systému, a to formou tzv. *serverů* poskytujících služby [4] (s. 286). Z takových serverů mimo sféru jádra jmenujme *User Library* (mj. abstrakce nad službami kernelu), *File Server* (souborové služby) a servery zastřešující grafické rozhraní, síťové a telefonní služby. Odlišnost spočívá v tom, že kernel Symbianu v sobě zahrnuje *plánovač* (*scheduler*) a také (byť binárně oddělené) *ovladače zařízení* spolu s „rozšířeními“ jádra (*extensions*), které by správně měly být mimo něj. Tím částečně zasahuje na pole *monolitických jader* a proto se občas řadí mezi *jádra hybridní* [4] (s. 283).

Mimoto, srdce jádra tvoří *nanokernel*. Právě zde mají svůj původ real-timové vlastnosti Symbianu. Slouží jako prvotní obsluha pro všechna *přerušení*, která předává dále; mimo krátké, časově ohraničené *kritické sekce* je plánován preemptivně [7] (s. 6–7).

1.1.4 Model systému

Symbian byl od svého zrodu vyvíjen s ohledem na požadavky zúčastněných výrobců MT na mírně odlišná zařízení s rozdílným UI. Tak se zrodil koncept „referenčního návrhu“ (*reference design*), kdy rodina zařízení sdílí stejný referenční návrh (tzv. *DFRD*[19], viz [4] od s. 410). Kód Symbianu proto implementuje pouze nejelementárnější logiku UI (komponenty Uikon a Application architecture, Control Environment, viz dále), na níž je provizorně posazeno referenční provedení vrchní části UI. Obé má svůj původ v uživatelském rozhraní EPOCu s názvem *Eikon*. Zda pak bude tato vrchní vrstva nahrazena vlastní implementací UI, nebo licencovaným produktem, je pak na rozhodnutí výrobce. I tento aspekt konkrétního nasazení systému je zachycen v souhrnném modelu systému na obr. 1.1.

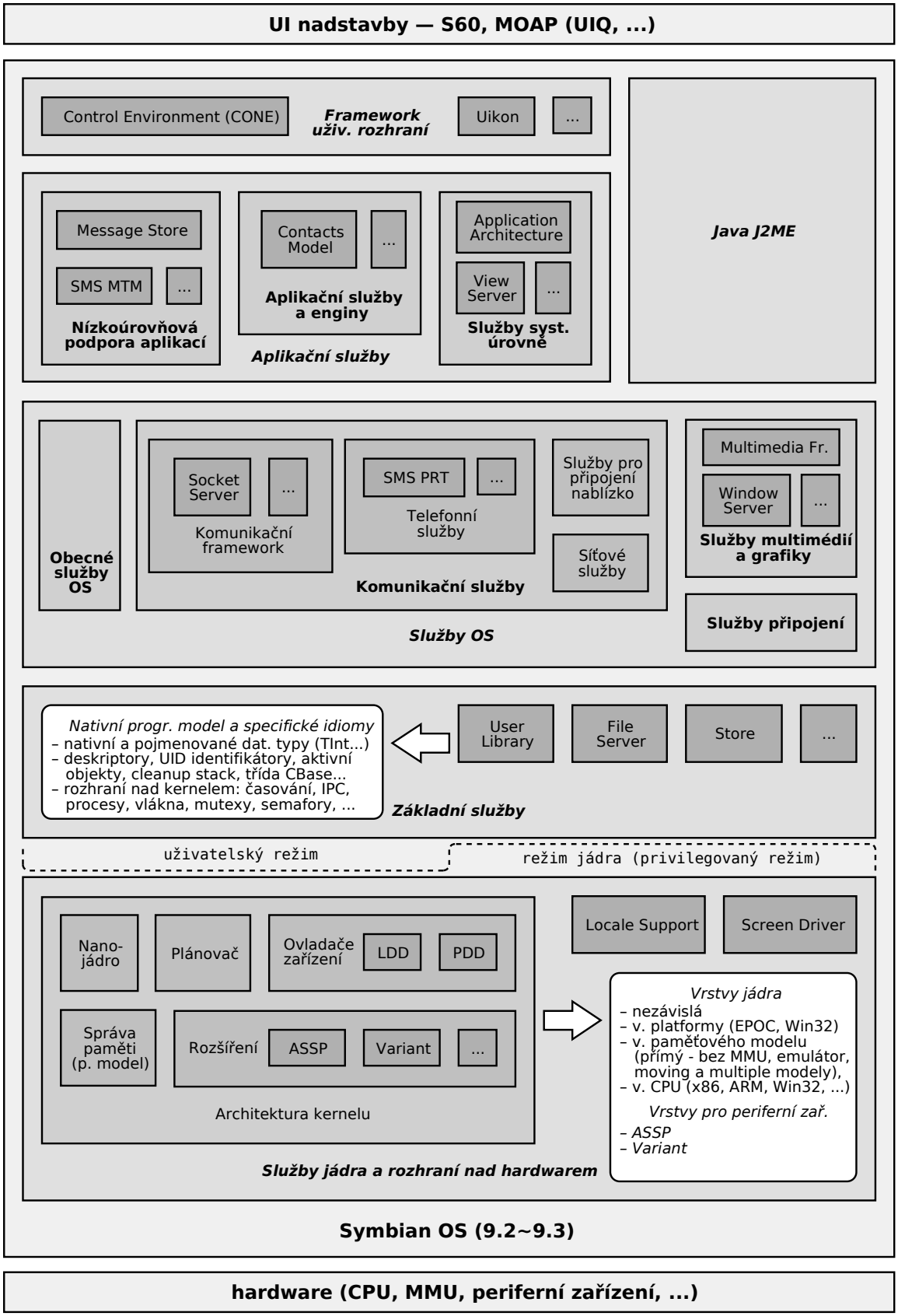
Ten vznikl zjednodušením schématu „Symbian OS v9.3: System Model“ [44], resp. „Block decomposition in the system model“ v knize [4] (obr. 5.2, s. 115) a doplněním některých informací (nejkonkrétnější bloky byly kvůli zachování popisnosti ponechány v angličtině, bloky ... vyjadřují neúplnost výčtu). Dopředu v něm rovněž byly vyznačeny některé konkrétní bloky významné pro praktickou část této práce a ta se k němu bude zpětně vracet.

[16] *EPOC Kernel Architecture*

[17] *real-time operating system*, viz `<http://en.wikipedia.org/wiki/Real-time_operating_system>`

[18] *inter-process communication*, meziprocesová komunikace, viz `<http://en.wikipedia.org/wiki/Inter-process_communication>`

[19] *Device Family Reference Design*



Obrázek 1.1: Model Symbianu (v kontextu jeho konkrétního nasazení) znázorňující jeho jednotlivé vrstvy včetně významných komponent

Jak je patrné, architekturu Symbianu lze promítnout do několika vrstev, kdy obecně každá vrstva abstrahuje funkcionalitu vrstev nižších a poskytuje služby vrstvám vyšším [4] (s. 113). Pro názornost významu jednotlivých vrstvech jsou abstraktní jednotky dané funkcionality systému znázorněny jako bloky, ty se podle svého zaměření v rámci dané vrstvy seskupují do logicky vyšších celků. Stručný popis těchto vrstev, odzdola nahoru, následuje[20].

Služby jádra a rozhraní nad hardwarem (*Kernel Services & HW Interface*): Jediná vrstva pracující v privilegovaném režimu. Kromě samotného jádra obsahuje komponenty, které tvoří rozhraní nad HW (logické /LDD/ a fyzické /PDD/ ovladače zařízení a „rozšíření“, která řeší další HW závislosti vztažené jak na *aplikačně specifický obvod* /ASSP[21]/, tak i na další HW konkrétního přístroje /variant/).

Základní služby (*Base Services*): Nad předchozí vrstvou staví „minimální použitelný operační systém“ prostřednictvím nejelementárnějších služeb (práce s pamětí, abstrakce souborového systému, atp.). Blok User Library představuje rozhraní nad kernelem.

Služby OS (*OS Services*): Po přechozí vrstvě dotváří „kompletní operační systém“ pomocí další serverů, frameworků a knihoven. Jde o tzv. *middleware*[22] vrstvu, tj. zajišťuje spolupráci mezi nižšími vrstvami bazálního OS a vyššími vrstvami orientovanými na podporu aplikací. Význačný je Window Server (zkráceně WServ), který zaštiťuje přístup k fyzicky vykreslovanému obrazu — formou abstrakce oblastí obrazovky, tzv. *oken* (viz podkap. 1.2.3) — a zároveň přístup k událostem zvenčí; klienti jej používají zpravidla pomocí bloku Control Environment (viz dále) [4] (s. 170, 182–184, 189).

Aplikační služby (*Application Services*): Podpůrná vrstva aplikací, která nijak přímo nesouvisí s UI. Skládá se z těchto kolekcí služeb:

- *Nízkoúrovňová podpora aplikací* (*Lower-level application support*): aplikační logika obecnějšího charakteru (podpora pro aplikace výměny zpráv, pro prohlížení webu, atp.)
- *Aplikační služby a enginy* (*Application services and engines*): logika pro konkrétní aplikace (např. služby pro *PIM*[23] aplikace, kterými jsou např. kartotéka kontaktů a kalendář)
- *Služby systémové úrovně* (*System level services*): logika používaná všemi aplikacemi; podstatný je blok Application Architecture zkráceně (App Arc), který představuje základní kámen pro výstavbu aplikace (viz [1], obr. 8.4 na s. 168), a View Server, jenž umožňuje přepínání tzv. *pohledů* (*views*, logické „stránky“ aplikací) [4] (s. 138–139).

Framework uživatelského rozhraní (*UI Framework*): Poskytuje zázemí pro využívání UI, včetně základních tříd pro jeho prvky. Uikon představuje nejvyšší úroveň UI, která je implementována ještě v rámci Symbianu, na ní už navazuje konkrétní provedení UI nadstavby. Obojí je propojeno s blokem Window Server (viz třetí z předešlých vrstev) pomocí abstraktnějšího bloku Control Environment (zkráceně CONE), který také zabezpečuje základní chování prvků UI včetně správy obdržených událostí (např. uživatelský vstup) [4] (s. 125–127). Vhodnou demonstrací souvislostí mezi těmito vrstvami budiž obrázek „Cone architectural relationships“ z průvodce systémem Symbian [42].

Java J2ME: Představuje analogii předchozích dvou vrstev pro běh *Java* aplikací. Zahrnuje implementaci *platformy Java* (v konfiguraci Micro Edition, CLDC/MIDP[24]).

[20] jedná se o především výtah z [4], s. 117–118; bližší detaily o vrstvách podá tatáž kniha
[21] *application-specific standard part/product*
[22] viz `<http://en.wikipedia.org/wiki/Middleware>`
[23] *personal information management*, doslovně přeloženo do češtiny jako „správa osobních informací“
[24] viz `<http://java.sun.com/javame/technology/#cldc>`

1.2 Platforma S60

Věnujme nyní pozornost také vrstvě UI nadstavby (viz obr. 1.1). Nejprve je vysvětlen její význam a podán přehled hlavních představitelů, pak se zájem přesouvá na jednoho konkrétního zástupce spjatého s touto prací — platformu S60. Tato podkapitola vychází, kromě explicitně zmíněných pramenů, ze zdrojů [1] a [47].

1.2.1 Nadstavby uživatelského rozhraní Symbianu

Vrstva UI nadstavby obnáší jak samotné UI, tak základní aplikační vybavení, dále často nové technologie, služby (middleware knihovny) a nová prostředí pro běh aplikací. Po boku původních bloků Symbianu v modelu systému (viz obr. 1.1) se tak v jeho vyšších vrstvách objevují nové celky, což zjednodušeně vystihuje schéma „Symbian OS Layers“ v publikaci [25] (s. 38/43).

Z tohoto důvodu se tento celek často označuje jako *platforma*. V závislosti na otevřenosti platformy vůči vývojářům jsou zájemcům zpřístupněny vývojové prostředky a veřejná část *rozhraní pro programování aplikací* (tzv. *API*[25]) platformy ve formě balíků *SDK*[26].

Historicky jsou známy tyto hlavní platformy Symbianu [4] (s. 122):

- ***Series 60***, později přejmenována na ***S60***: Pochází od Nokie a je licencována také dalším výrobcům MT.
- ***Series 80*** **(*S80*)**: Také od Nokie, ale nelicencována. Již delší dobu nenašla uplatnění[27].
- ***Series 90***: Rovněž od Nokie a nelicencována. Upustilo se od ní, sloučena s S60[28]. Její určitý pozůstatek je v UI s názvem *Hildon* používaném Nokií u platformy *Maemo* [4] (s. 122).
- ***UIQ***: Z dílny ryze softwarové společnosti *UIQ Technology*, která jej licencovala partnerům z oblasti výrobců MT. Začátkem roku 2009 ji však postihl krach[29].
- ***MOAP***: Využívá se v rámci *3G*[30] služeb pod značkou *FOMA* nevýznamějšího japonského mobilního operátora *NTT docomo*.

Vývoj na tomto poli je aktuálně poznamenán iniciativou Nokie „vytvořit ze Symbianu a jeho roztříštěných platforem jednotnou otevřenou platformu pro konvergovaná mobilní zařízení a dát tak tomuto odvětví impuls k rychlejšímu rozvoji“ [33]. Za tímto účelem byla založena společnost *Symbian Foundation*, jejímž prostřednictvím by také měly být některé komponenty nové platformy zveřejněny jako *open source*[31] [29].

Dáme-li tuto iniciativu, která se týká spíše budoucnosti, stranou, aktuálními platformami jsou S60 a MOAP. První je, jak také naznačuje její slogan [17], „Open to new features“ čili platformou novým aplikacím (potažmo jejich vývojářům) otevřenou, druhá nikoli[32].

[25] *Application Programming Interface*; v textu často i stručněji jako *programovací rozhraní*
[26] *Software Development Kit*
[27] viz `<http://www.forum.nokia.com/main/platforms/s80/#devices>`
[28] viz `<http://wiki.forum.nokia.com/index.php/Series_90>`
[29] viz `<http://www.esato.com/news/article.php/id=1811>`
[30] třetí generace technologií mobilní telekomunikace, viz `<http://en.wikipedia.org/wiki/3G>`
[31] tedy včetně zdrojových textů
[32] viz `<http://en.wikipedia.org/wiki/MOAP>`

1.2.2 Historie S60

Platforma S60 vzešla v roce 2001 z popudu Nokie, na základě DFRD s názvem *Pearl*[33], jehož kořeny sahají k původnímu DFRD *Saphire* [4] (s. 412–413, 417–419). Poprvé byla nasazena v roce 2001 se Symbianem ve verzi 6.1[34] [4] (s. 320). K přejmenování z původního *Series 60* na nynější název *S60* došlo v roce 2005 [31].

Pragmatickým zájmem Nokie[35], coby původce této platformy, bylo tuto rozšířit jako standardní platformu pro smartphony formou licencování i mezi ostatní výrobce MT. Takovými partnery byly v průběhu času zejména společnosti *Sendo*, *Siemens* (*BenQ–Siemens*), *Panasonic*, dále *Samsung*, *Lenovo* a *LG Electronics*.

Majoritní verze platformy S60 se udává řadovou číslovkou na způsob „o kolikátou *edici* (*edition*) jde“[36], minoritní je verzována pomocí pořadového čísla „balíku nové funkcionality“ zvaného *feature pack* (*FP*)[37] a u čerstvé majoritní (tj. nulté minoritní) verze se neuvádí.

Doposud vznikly čtyři edice S60: od *S60 1st Edition* po *S60 5th Edition*, přičemž ze třetí edice se přešlo přímo na pátou, čtvrtá byla přeskočena[38]. Binární nekompatibilita, zmíněná v podkapitole 1.1.1, se této platformy dotkla s verzí *S60 3rd Edition* (počáteční vydání), pro niž je zapotřebí kód aplikací určený pro starší edice — přinejmenším — znovu překompilovat [20] (s. 14), [19] (s. 20–21). Pátá edice je již se třetí edicí opět víceméně kompatibilní. I z tohoto důvodu se další text i praktická část práce zaobírá pouze těmito dvěma nejnovějšími řadami.

1.2.3 Vlastnosti platformy S60 a jejího uživatelského rozhraní

Odhlédneme-li od nejčerstvějších inovací, je tato platforma oproti zbylým specifická svým zaměřením na ovládání jednou rukou[39] a přímočarým ovládáním. Kromě standardní[40] numerické klávesnice je u ní ustálena také sada základních tlačítek pro určité funkce. Mezi ně se řadí dvojice (trojice) *výběrových tlačítek*[41] (tzv. *softkeys*), které provádí akce vypsané v příslušných pozicích spodního řádku na displeji (viz dále), navigační tlačítka (nahoru, dolu, doprava, doleva a výběr), tlačítka pro přepínání aplikací (*application key*), mazání textu/zvolené položky (*clear key*), příjem a odmítnutí hovoru [1] (s. 22–23), [23] (s. 11–13).

Na nejvyšší úrovni bývá uživatelské rozhraní (nativních) aplikací pro S60 koncipováno ze dvou odlišných pojetí: pohyb ve výčtu prvků a zobrazení podrobností [1] (s. 22). S60 rozvíjí v Symbianu zakotvenou architekturu *MVC* (viz podkap. 1.1.2) a princip *pohledů* aplikací (viz podkap. 1.1.4). Celkově nabízí tyto možnosti kompozice UI aplikací, tzv. *application architectures*[42]: pro Symbian tradiční architektura založená na ovládacích prvcích (*control-based architecture*), dále architektura založená na dialozích (*dialog/-based/ architecture*) a s přepínáním pohledů (*view/-switching/ architecture*). Implementační stránce výstavby aplikace v závislosti na zvolené architektuře se věnuje samostatná podkapitola 1.3.1.

[33] souběžně s tímto DFRD existovaly ještě *Quartz* (→UIQ) a *Crystal* (→Series 80)

[34] jednalo se o MT *Nokia 7650*

[35] důvody osvětluje [1], s. 13

[36] např. „S60 *2nd Edition*“

[37] např. „S60 2nd Edition, *Feature Pack 3*“

[38] jako zdvořilé gesto vůči asijským zákazníkům, pro které čtyřka představuje „nešťastné číslo“, viz `<http://blogs.s60.com/2008/10/s60-5th-edition-and-the-nokia-5800-xpressmusic-are-here#comment-2107>`

[39] nově však také disponuje dotykovovým ovládáním s použitím tzv. *stylusu*

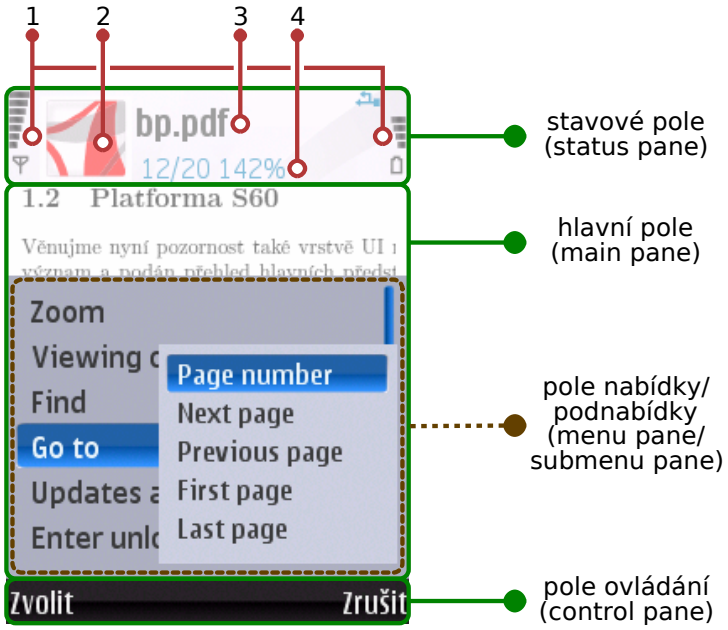
[40] dle doporučení *ITU-T*, viz `<http://en.wikipedia.org/wiki/E.161>`

[41] překlad převzat z příruček k MT Nokia

[42] podrobnosti, jako i ilustrační diagramy tříd a náznaky implementace podá [3] a [15]

Potřebu podpory různých obrazovkových rozlišení — a to jak v tradiční orientaci „na výšku“, tak i „na šířku“ — reflektuje S60 tzv. *škálovatelností* (*scalability*) na úrovni systému i konkrétní aplikace. Příkladem může být podpora pro vektorový formát ikon (*SVG-T* z rodiny formátu *SVG*[43]).

Jak bylo zmíněno u bloku Symbianu Window server (viz podkap. 1.1.4) jsou hierarchicky nejvyšší komponenty tvořící *obrazovku* (*screen*)[44] tzv. *okna* (*windows*). Okno může zabírat buď celou obrazovku, nebo pouze jeho část, kdy na zbytku obrazovky zůstává viditelný původní obsah. U S60 se okno dále člení na tzv. *pole* (*panes*), která mohou obsahovat rekurzivně další dílčí pole (*subpanes*). *Okno aplikace* (*application window*), které zabírá celou obrazovku, je obyčejně uspořádáno, jak včetně vyznačení základních prvků ukazuje obr. 1.2. Detailnější popis následuje (stejně jako celý tento odstavec, na základě [1] /s. 26, 194–209/ a [23] /s. 14–19/).



Obrázek 1.2: Typické uspořádání *okna aplikace* (Adobe Reader LE 2.5, motiv „Topaz 3rd by PiZero“)

Stavové pole (*status pane*) zobrazuje základní informace o běžící aplikaci a stavu zařízení. Najdeme zde (viz očíslované položky na obr. 1.2):

(1) ukazatel signálu a stavu baterie, včetně některých dalších indikátorů (např. ukazatel aktivního *USB* připojení na obr. 1.2)

(2) *pole kontextu* (*context pane*) — obvykle používáno pro ikonu aplikace

(3) *pole názvu* (*title pane*) — obyčejně nese jméno aplikace, případně označení aktuálního pohledu (*view*, viz dále)

(4) *navigační pole* (*navi pane*) — pro zobrazení dodatečných informací usnadňujících orientaci a navigaci v rámci aplikace; může být prázdné nebo mít podobu „záložek“ (*tabs*), textového popisku, obrázku, různých indikátorů, případně dalších prvků

[43] viz `<http://en.wikipedia.org/wiki/Scalable_Vector_Graphics#Mobile_profiles>`
[44] odpovídá fyzicky zobrazované ploše

Plocha, kde aplikace zobrazuje svůj obsah, se nazývá **hlavní pole (*main pane*)**. Z celkové velikosti obrazovky zabírá největší část. Pro její vykreslení se používá buď vlastní logika aplikace (např. u her), nebo je vyplněna standardním prvkem UI, často jde o *listbox* nebo jeho obdobu *grid* (viz dále). Spodní část tvoří **pole ovládání (*control pane*)** a zobrazují se zde popisky pro výše popsané softkeys, což se jinak označuje též *command button area* (*CBA*).

Podobně jako blok Symbianu Uikon poskytuje základní podporu pro prvky UI (viz podkap. 1.1.4), komponenta uživatelského rozhraní *Avkon* platformy S60 přináší jejich konkrétní ztvárnění. K nejdůležitějším patří (na základě [1], s. 210–271 a [23], s. 34–74):

- **seznam (*list*)** — standardně vertikální seznam (*listbox*), případně i jeho dvourozměrná interpretace jako *mřížka* (*grid*); dále se dělí na:
  - seznam položek nabídky (*menu list*): jakmile je nabídka vyvolána, zobrazí se „vyskakovací“ okno (*popup window*) s *polem nabídky* (*menu pane*) a v něm definované položky (viz obr. 1.2)
  - seznam s možností volby položky (*selection list*) a také s možností volby obecně více položek (*multi-selection list*), u něhož je účel výběru dán předem
  - seznam s možností označení více položek, nad nimiž pak lze hromadně vykonat dodatečně zvolenou operaci (*markable list*)
  - formulář (*form*): seznam editovatelných položek (polí), a to polí textových, polí s výběrem z roletového menu (*popup field*) nebo polí editovatelných pomocí „jezdce“ (*slider*); může být použit v režimu „prohlížení“ nebo „editace“
  - seznam nastavitelných položek (*setting item list*): viz předposlední vnější odrážka
- **sdělení (*note*)**: Spadají sem např. varianty, jež sdělují potvrzení akce (*confirmation note*), informaci o nastalé situaci (*information note*), varování či chybu (*warning*, *error note*), průběh nějaké činnosti nebo čekání na její dokončení (*progress*, *wait note*). Specifická jsou „sdělení s výběrem“ (*soft notifications*), u nichž může uživatel sdělenou událost prostřednictvím výběrových tlačítek buď ignorovat, nebo spustit odpovídající akci (např. zobrazit příchozí zprávu).
- ***dialog***, především jeho specializace **výzva (*query*)**, která si žádá interakci uživatele — např. výzva, aby uživatel potvrdil akci (*confirmation query*), aby zadal nějaký datový vstup (*data query*), nebo aby vybral z nabídnutého seznamu možností (*list query*); dělí se na výzvy globální (zůstavají zobrazené i po přepnutí aplikace do pozadí) a lokální (přesunují se na pozadí společně s aplikací).
- **položka nastavení (*setting item*)** a **stránka nastavení (*setting page*)** využívaná pro změnu nastavení konkrétní položky: Obojí souvisí se seznamem nastavitelných položek (viz odrážka označená prázdným kroužkem).
- ***editor***: Slouží např. pro zadání textu, čísla apod. Používá se u formulářů a položek, resp. stránek nastavení.

1.2.4 Technologie pro vývoj aplikací u třetí (páté) edice S60

Jak jsme si stanovili (viz podkap. 1.2.2), zabýváme se pouze posledními dvěma edicemi. Podrobnosti o nich z prostorových důvodů vynecháme[45], pouze nás budou zajímat dostupné

[45] lze odkázat např. přímo na prospekty značky S60: `<http://www.s60.com/life/thisiss60/s60indetail/softwareversions/3rdedition>` pro 3rd Edition a `<http://www.s60.com/life/thisiss60/s60indetail/softwareversions/5thedition>` pro 5th Edition

technologie umožňující vývoj aplikací pro ně, přičemž vycházet budeme hlavně z [16] a [24].

Pro S60, stejně jako pro samotný Symbian, je nativním programovacím prostředkem **jazyk *C++***. Ve srovnání s ostatními možnostmi nabízí aplikacím největší potenciál jak v otázce výkonu, tak v míře dostupné funkcionality. Ta je zde (avšak podobné je to i u dalších možností vývoje) v určitých případech chráněna zmíněným bezpečnostím modelem (viz podkap. 1.1.1). Podotkněme, že se nejedná o plně standardní C++, odlišnosti spočívají v ustálení mnohých programovacích idiomů a konvencí. Jejich dodržování — pokud už, jako v některých případech, není striktně vynucené — má sloužit jako prevence před možnými problémy, prostředek usnadňující programátorovu práci, případně zlepšující čitelnost kódu. Protože byla zvolena právě tato varianta pro praktickou část práce, jsou v podkapitole 1.3 ona specifika včetně základního postupu při sestrojení aplikace používající UI naznačena.

Pojmy ***Open C*** a ***Open C++*** používá Nokia pro standardní (byť s omezeními) verze příslušných programovacích jazyků[46]. Základní podpora pro aplikace vytvořené v *C* je přítomna již v samotném Symbianu. S60 k ní přidává některé další standardní knihovny a kompletní podporu pro Open C++. Ta je u novějších vydání platformy již přímo integrována, u ostatních ji lze dodat formou instalovatelných pluginů. Podpora pro vývoj založený na těchto prostředcích je pak poskytována formou dodatků k SDK. Podobným způsobem může být přidána podpora (zatím v testovací verzi) pro vývoj aplikací, u nichž uživatelské rozhraní používá technologii *Qt*, známou z osobních počítačů[47].

Jak je patrné z obrázku 1.1, je podpora ***platformy Java*** (*Java J2ME*/*Java ME*) obsažena přímo v Symbianu a S60 tuto technologii rovněž podporuje[48]. Jejím původcem je společnost *Sun Microsystems*.

Za další technologií pro vývoj aplikací, ***Flash Lite***[49], stojí Adobe. Jeho produkt *Flash* se zaměřuje na grafický interaktivní obsah a toto je jeho odnož určená pro mobilní zařízení.

Nelze opomenout ani vývoj pomocí moderního *dynamického* jazyka ***Python***. Jeho implementace pro S60[50] je obohacena o rozhraní pro přístup ke službám této platformy (potažmo i službám Symbianu) a umožňuje používat její nativní UI. Pro svoji jednoduchost představuje pro programátory snadno uchopitelný nástroj pro vývoj aplikací i ideální prostředek pro tvorbu jejich prototypů.

1.3 Průřez programovacími koncepty a specifiky C++ pro Symbian/S60

Jak bylo řečeno (viz podkap. 1.2.4), nativní C++ pro Symbian má oproti standardnímu *ANSI C++* jisté rozdíly. Hlavními příčinami je skutečnost, že vývoj Symbianu (tehdy jako OS Epoc, viz podkap. 1.1.1) započal ještě před ustanovením normy jazyka C++, a relativně dosti omezená kapacita paměti cílových zařízení.

Na toto téma existuje nespočet materiálů, za všechny jmenujme [18] a [25] (s. 84/89/–106/111/, 120–146, 162–184, 270–285), ze kterých těžil i následující výběr toho nejpodstatnějšího:

[46] viz <http://www.forum.nokia.com/Resources_and_Information/Explore/Runtime_Platforms/Open_C_and_C++/>

[47] viz <http://www.qtsoftware.com/developer/technical-preview-qt-for-s60>

[48] viz <http://www.forum.nokia.com/Resources_and_Information/Explore/Runtime_Platforms/Java.xhtml>

[49] viz <http://www.forum.nokia.com/Resources_and_Information/Explore/Web_Technologies/Flash_Lite>

[50] viz <http://www.forum.nokia.com/Resources_and_Information/Tools/Runtimes/Python_for_S60>

a. **alternativní mechanismus výjimek**: Přestože od Symbianu verze 9 je možné používat i standardní konstrukce `try` – `catch()` – `throw`, upřednostňován je stále tradiční, „ořezaný“ mechanismus výjimek. Založen je na tzv. *leaves*[51], které propagují informaci o vzniklé chybě „výše“, ty se zachytávají do „pasti“ v podobě makra `TRAP`. Informace, že funkce může takto „odejít“, bývá zpravidla vyjádřena pomocí koncového ‚`L`‘ v jejím identikátoru (viz odrážka f). Krom toho ještě existují obzvlášť závažné chyby typu *panika* (*panic*), jež takto odchytit nelze, vedou k ukončení vlákna/procesu/aplikace.

b. **úklidový zásobník**[52] **(*cleanup stack*)**: Slouží jako prevence před *úniky paměti* (*memory leaks*)[53], programátor sem ukládá ukazatele na objekty, jimiž zabranou paměť je nutné při vzniku výše popsané události *leave* uvolnit.

c. **dvoufázová konstrukce (*two phase construction*)**: Umožňuje s využitím úklidového zásobníku (viz odrážka b) bezpečné jednorázové vytvoření objektu (pomocí standardního C++ *konstruktoru*, ten bývá *neveřejný*) včetně provedení všech potřebných, potenciálně „odcházejících“ (viz odrážka a) operací (dle konvencí jsou vyčleněné v samostatné *neveřejné* metodě, typicky pojmenované `ConstructL()`). Tyto dvě fáze vytváření objektu bývají zapouzdřeny ve *veřejných statických* metodách objektu, obyčejně pojmenovaných `NewL()` a `NewLC` (podle toho, zda nechávájí ukazatel nově vytvořeného objektu v úklidovém zásobníku, či nikoliv).

d. **model *klient–server***: Jak bylo řečeno, Symbian se svou architekturou mikrojádra (viz podkap. 1.1.2 – 1.1.4) je zhusta protkán *servery*, poskytujícími přístup k systémovým prostředkům. Aplikace je využívají skrze příslušné API (odstiňuje konkrétní formu IPC), a to buď synchronním nebo asychronním způsobem.

e. **aktivní objekty (*active objects*)**: Představují doporučený, objektově zapouzdřený mechanismus pro neblokující požadavky na asynchronní služby serverů (viz odrážka d) — jakmile je požadavek vyřízen, je daný klient zpětně informován. Množinu takto vznesených požadavků (jeden aktivní objekt může mít nanejvýš jeden nevyřízený požadavek) v rámci jednoho vlákna spravuje programová smyčka *aktivní plánovač* (*active scheduler*). Ta je implicitní součástí aplikace, ovšem existují i pokročilejší možnosti práce s těmito prostředky. Jinou možností, jak dosáhnout takového cíle, je použití vláken; ty však v této roli nejsou tak efektivní (dle [2]) a tak snadno použitelné.

f. **identifikátory podléhají konvencím**: Podle typu programové konstrukce, kterou zastupují, se přidává na začátek nebo konec určené písmeno: počáteční ‚`K`‘ u konstant, počáteční ‚`i`‘ u datových členů tříd/struktur, atp.

g. **upřednostňovaná označení datových typů**: Např. `TInt`, `TBool`.

h. **řetězce**: Používání standardních řetězců potlačeno na úkor tzv. ***deskriptorů***, které si o sobě evidují svou aktuální délku a maximální dostupnou délku, implicitně podporují *Unicode*. Literály se definují pomocí makra `_LIT()`.

i. až na výjimky celá **implementace staví na třídách a jejich objektech**: Na vícenásobné dědičnosti se mohou podílet pouze třídy tvořící abstraktní rozhraní (tzv. *mixins*[54]), které se rozeznávají pomocí počátečního písmena ‚`M`‘ (viz odrážka f).

[51] česky lze volně přeložit jako „odchod“

[52] tento překlad převzat z obsahu knihy *Harrison: Programujeme aplikace pro Symbian OS*, který je elektronicky dostupný na `<http://knihy.cpress.cz/Pocitac/BookDoc.asp?DocID=0&BookID=2600&DownloadSection=11>`

[53] viz `<http://en.wikipedia.org/wiki/Memory_leak>`

[54] viz `<http://en.wikipedia.org/wiki/Mixin>`

1.3.1 Základní struktura C++ aplikace pro S60, forma její distribuce

K výše uvedeným specifikům se sluší také alespoň stručně uvést, jakým způsobem se v objektovém pojetí Symbianu, konkrétně v případě platformy S60 provádí základní výstavba nativní C++ aplikace a jak potom vypadá její výsledná forma určená pro distribuci na konkrétní zařízení.

Nejprve k základní struktuře aplikace. Ta se odvíjí od požadavků daných *frameworkem aplikací* (*application framework*), kterého jsme se již nepatrně dotkli v podkapitole 1.1.4 u bloku Application Architecture, a také závisí na zvoleném způsobu kompozice UI aplikace, které byly vyjmenovány v podkapitole 1.2.3.

Z programátorského hlediska tyto požadavky představují nutnost vytvořit sadu předepsaných tříd dědících předepsaným způsobem z k tomu vyhrazených tříd frameworku. U S60 jde, v krátkosti, o tyto (na základě zdrojů [3] a [15], kde je možné nalézt bližší podrobnosti včetně ilustračních diagramů):

1. **třída aplikace** (dědí ze třídy `CAknApplication`): Používá se ve fázi spouštění aplikace a slouží pro vytvoření „dokumentu“.
2. **třída dokumentu** (dědí ze tř. `CAknDocument`): Slouží pro vytvoření „uživatelského rozhraní“ a nabízí metody, které se váží s persistencí aplikačních dat.
3. **třída uživatelského rozhraní**: Má na starosti zpracování některých událostí, může reagovat na uživatelské vstupy. Konkrétnější vlastnosti se liší v závislosti na použité architektuře aplikace:
   (3c) pro tradiční arch. Symbianu (dědí ze tř. `CAknAppUi`): přímo vlastní ovládací prvky tvořící pohled aplikace (ty dědí ze tř. `CCoeControl`)
   (3c) pro arch. dialogů (dědí ze tř. `CAknAppUi`): pohledy aplikace jsou tvořeny dialogy
   (3c) pro arch. platformy S60 používající přepínání pohledů (dědí ze tř. `CAknViewAppUi`): vlastní pohledy (ty dědí ze tř. `CAknView`), které představují logické „stránky“ aplikace, a ty dále vlastní samotné ovládací prvky tvořící tyto pohledy (stejně jako u tradiční arch. dědí ze tř. `CCoeControl`); aktivní přitom může být současně pouze jeden pohled (při přepnutí se ten původní deaktivuje), aktivovat je možné také pohledy externí (pohledy jiných aplikací)

Pro distribuci obecně všech aplikací vytvořených v nativním C++ pro Symbian, ty určené pro platformu S60 nevyjímaje, je zaveden tzv. *Symbian OS Installation System* (zkráceně *SIS*). Ten kromě prostředků umožňujících provedení instalace aplikace na straně Symbianu zahrnuje také specifický formát instalačních souborů. Do nich se zabalí sestavená aplikace a všechny další potřebné soubory (přeložené soubory *zdrojů*, ikona aplikace atp.) a v této formě pak lze aplikaci přenést na konkrétní zařízení, případně poskytnout koncovým uživatelům. Zdůrazněme, že jediná cesta, jak aplikaci nahrát do telefonu a zde ji používat, vede právě přes její instalaci.

Tyto instalační soubory mají koncovku `.sis`, od verze Symbianu 9 se používá také koncovka `.sisx`, která značí, že daný soubor instalace je digitálně podepsaný. Související problematika bezpečnostního modelu (*Platform Security*) již byla nastíněna v podkapitole 1.1.1. Od jejího zavedení u S60 3rd Edition je takový podpis instalačních souborů pro tuto platformu důsledně vyžadován. Typů podpisu je povícero, pro nás nejdostupnější je možnost podepsání SIS souboru ve vlastní režii, tj. bez nutnosti vlastnit nějaký certifikát, aplikace je pak takzvaně *self-signed*.

Kapitola 2

Vyhovění nárokům na aplikaci ze strany dostupného API

Nejdříve jsou upřesněny požadavky na výslednou ukázkovou aplikaci, poté může být API podrobeno bližšímu průzkumu s cílem zjistit, jakým způsobem — pokud vůbec — lze takové požadavky splnit. Vychází se přitom z dokumentace, která je přibalena k C++ SDK pro S60 3rd Edition (FP1) [21][1] a z komentářů v hlavičkových souborech tamtéž.

2.1 Specifikace požadavků na aplikaci

Vyjděme z požadavků plynoucích ze zadání práce: prvotním cílem je možnost nastavení individuálního zvukového ohlášení příchozí zprávy podle toho, do které skupiny kontaktů její odesílatel patří. Pro účely aplikace postačí, pokud se zaměříme pouze na zprávy *SMS*[2] (dále také jen *zprávy*).

Lze odtušit, že pouhá příslušnost odesílatele ke skupině zcela jistě nevyčerpává všechny běžně využitelné možnosti, jakými lze rozlišovat příchozí zprávy. Proto by aplikace měla, alespoň v návrhu, počítat s budoucím navýšením počtu těchto kritérií. Na jejich základě pak uživatel může určovat konkrétní pravidla pro selekci zpráv (*filtry*), s každým je spjat individuální způsob upozornění na odpovídající zprávu. Nabízí se kupříkladu hledisko prefixu masky telefonního čísla odesílatele[3]; zde za zmínku stojí fakt, že identifikace odesílatele nemusí nutně znamenat tel. číslo, v ojedinělých případech má totiž formu obecného řetězce[4].

V případě zahrnutí dvou či více filtračních kritérií do aplikace vyvstává otázka, zda nabídnout složená pravidla (kombinace pomocí logických operátorů `AND`, `OR` apod.) — pro jednoduchost (jak z pohledu ovládání uživatelem, tak z pohledu implementace) nejsou složená pravidla požadována. Potom však bude potřeba zvolit vhodné pořadí uplatňování jednotlivých kritérií i filtrů definovaných v rámci nich; jedním z úkolů UI bude intuitivní formou prezentovat uživateli právě tuto informaci o obou pořadích.

[1] stejný obsah dá dohromady kombinace zdrojů [11] (S60) a [35] (Symbian)

[2] *Short Message Service*, služba krátkých textových zpráv, viz `<http://en.wikipedia.org/wiki/SMS>`

[3] ze zkušenosti využitelné např. u zpráv doručených (ve formě SMS) prostřednictvím *e-mailu* běžně zaslaného na u mobilního operátora vyhrazenou e-mailovou adresu, konkrétně u české Telefonicy O_2 — začátek tel. čísla odesílatele je neměnný, číslo se liší až v několika posledních číslicích

[4] např. zprávy odeslané skrze internetovou SMS bránu tuzemského operátora T-Mobile mívají v poli identifikujícím odesílatele místo obvyklého telefonního čísla řetězec „`t-zones SMS`“, což má mj. ten důsledek, že tohoto odesílatele nelze uložit jako kontakt a taková zpráva je pak kritériem příslušnosti odesílatele ke skupině kontaktů nepostihnutelná

Po příchodu zprávy se tedy budou postupně procházet uživatelem definované filtry s ohledem ona pořadí a pro upozornění uživatele se použije nastavení vázané na první pravidlo se shodou. Pokud se ke zprávě nenalezne žádný odpovídající filtr, bude uživatel o nové zprávě vyrozuměn standardním způsobem, dle aktuálního profilu.

Co se týče uživatelských voleb způsobu ohlášení zprávy, k dispozici by měla být přinejmenším možnost vybrat zvukový soubor, v tomto kontextu označován jako *tón*[5]), který se má při obdržení zprávy přehrát (včetně možnosti nepoužít tón žádný). Uživatelé by rovněž mohli ocenit volbu hlasitosti přehrání, případně volby, zda použít vibrační prvek telefonu.

Téměř samozřejmým požadavkem je uchování uživatelem definovaných položek filtrů a souvisejících nastavení i po ukončení aplikace (resp. i po vypnutí telefonu), a jejich opětovné načtení s jejím novým spuštěním. Z hlediska UI aplikace postačí, mimo zmíněné nezatajení informace o pořadí průchodu filtry, pokud její ovládání uživatele, zvyklého na integrované aplikace platformy S60, nijak nezaskočí a bude ji schopen intuitivně použít[6]. Aplikace by měla být lokalizována alespoň do češtiny a angličtiny.

2.2 Uživatelské zprávy

Podívejme se nyní na první fázi, kterou musí aplikace projít bezprostředně po výskytu nové příchozí zprávy v systému. Celkem existují tři způsoby, jak se z aplikace dozvědět o této události:

1. prostřednictvím navázané relace se *serverem zpráv* (*Message Server*)
2. skrze veřejné API vztahující se na komponentu *Log Engine*
3. pomocí zprostředkovaného interního API „SMS Utilities“

Ad 1: *Server zpráv* (*Message Server*) a jejich uložiště představují celek, který je na obr. 1.1 označen jako Message Store. Tento server umožňuje pomocí příslušného API základní manipulaci se zprávami (např. kopírování, odstranění, získání obecných informací o zprávě); implementace služeb pro jednotlivé druhy zpráv (SMS, e-mail atp.) se do připraveného frameworku začleňují jako jednotky zvané *MTM* (*message-type modules*) [1] (s. 375∼), [4] (s. 143–145). Z nich pro nás má význam pouze *SMS MTM*, představovaná na obr. 1.1 stejnojmenným blokem.

Komunikační spojení (relaci, sezení, *session*) mezi *message serverem* a jeho konkrétním klientem zapouzdřuje třída `CMsvSession`. Skrze ní je také možné přistoupit k abstaktní jednotce zprávy zvané *entry* (záznam). Jednotlivé záznamy mohou kromě celých zpráv reprezentovat také vnořené části jiných zpráv (např. přílohy e-mailu), složky zpráv či údaje příslušných konkrétních služeb. Záznamy jsou organizovány ve stromové struktuře, jak ukazuje schéma „Index structure“ z průvodce Symbianem [41].

Nejvyšší úrovní abstrakce záznamu, která zároveň zapouzdřuje základní manipulaci s ním, je třída `CMsvEntry`. Na nižší úrovni se pak rozeznává jednak souhrn základních informací o záznamu (zprávě) jako typ, velikost, datum a čas, které zapouzdřuje třída `TMsvEntry` a jednak samotný obsah (např. tělo zprávy), jenž se ukládá odděleně do vyhrazeného uložiště (*Message Store*) a který je zapouzdřen třídou `CMsvStore`. Zmíněný souhrn základních informací o záznamu představuje datovou položku indexu *serveru zpráv*, tzv. *index entry*.

Ke sledování událostí, jež se vyskytnou na straně *serveru zpráv*, je nutné připravit třídu „pozorovatele“, která podědí abstraktní třídu `MMsvSessionObserver` a implementuje její

[5] tento výraz je běžně používán v aplikaci správy profilů na MT Nokia a to i v české lokalizaci

[6] částečně lze odkázat na podkapitolu 1.2.3, část pojednávající o standardním *oknu aplikace*

metodu rozhraní `HandleSessionEventL()`. Skrze ní pak server, po vytvoření komunikační relace s ním, informuje o výskytu relevantních událostí. Samotná relace se serverem se naváže voláním statické metody `OpenSyncL()` (synchronní varianta), resp. `OpenAsyncL()` (asynchronní v.) třídy `CMsvSession`, přitom se jí jako argument předá odkaz na instanci oné třídy „pozorovatele“.

Nokia tuto možnost záchytu události příchozí zprávy demonstruje v aplikaci „SMS Example“ [22]. Experimentováním s touto aplikací se však došlo k nepříznivému zjištění, že systém se k takové události dostane vždy dříve a bez ohledu na následnou programátorovu manipulaci se zprávou (coby reakci na upozornění skrze `HandleSessionEventL()`) je přehrán přednastavený tón (podle profilu) a přinejmenším dočasně se zobrazí *soft notification* (viz prvky UI v podkap. 1.2.3) upozorňující na novou zprávu. Podobné chování výsledné aplikace by bylo asi jen stěží akceptovatelné.

Ad 2: Podobně nevyhovující chování, kdy se systém k události příchodu zprávy dostane dříve, než je nějaká možnost zasáhnout dána aplikaci, lze očekávat i v případě použití *Log Engine*. Proto tato možnost není dále zkoumána.

Ad 3: Daleko více nadějí vzbuzuje API označené „SMS Utilities“, které, ač je neveřejné, bylo zpřístupněno v balíku doplňkových rozhraní, jež rozšiřují možnosti standardního S60 C++ SDK[7]. Tyto balíky podporují pouze třetí a pátou řadu S60 a k mání jsou skrze [14]. Dobrým průvodcem těmito dodatkovými API je pak stránka na Wiki portálu Fora Nokia [28], kde jde také najít ukázkové aplikace využívající toto konkrétní API — jedna z nich demonstruje i možnost příjmu zprávy bez dalšího ohlášení. Solidní ukázku jeho použití pro tento účel podává rovněž webová stránka [5].

Experimentováním s kódem obou ukázek i vlastními pokusy se zmíněným API bylo možné dospět ke kódu, který je bez problémů funkční a přesně odpovídá potřebám aplikace, pouze je vhodné dodat, že se v mnohém podobá druhé citované ukázce[8].

Na bližší popis tohoto API bude lepší, i vzhledem k provázanosti s jedním významným prostředkem Symbianu – *sockety*, vyhradit samostatnou podkapitolu.

2.3 Sockety, příjem zpráv pomocí API „SMS Utilities“

API „SMS Utilities“ využívá princip tzv. *socketů*[9] („schránek“), které představují koncové body komunikace. Krátce se teď u nich zastavme.

Přestože pro snazší přenositelnost aplikací napsaných v jazyce *C* poskytuje Symbian rozhraní na způsob *BSD socketů*[10] [1] (s. 354), jeho nativní pojetí klientského rozhraní nad *sockety* (*Sockets Client API*) se mírně liší. Využit je tradiční model *klient–server*, kde ústřední bod tvoří *Socket Server* — na obr. 1.1 jej zachycuje blok stejného jména. Svým klientům nabízí přístup ke komunikačním službám na bázi *socketů*, přičemž většina operací se provádí asynchronně, a poskytuje informaci o dostupných protokolech [39], [40]. Ty jsou

[7] toto konkrétní API je přítomno v balících oněch doplňkových rozhraní pro všechny dosud vydané verze S60 počínaje 3rd Edition (počáteční vydání)

[8] jakkoli se může jevit, že jde pouze o bezmyšlenkovité převzetí tohoto kódu, je nutné zdůraznit, že jednak sám o sobě představuje takřka optimální a odladěné řešení pro tento dílčí úkol a jednak vše během vytváření vlastního prototypu s touto funkcionalitou bylo pečlivě testováno, konfrontováno se zkušenostmi ostatních vývojařů pohybujících se na diskusním fóru [13] a mnoho částí bylo podrobeno zkouškám „zda to nemůže být provedeno jinak, případně i lépe“ (povětšinou s negativním výsledkem)

[9] forma *IPC*

[10] viz `<http://en.wikipedia.org/wiki/Berkeley_sockets>`

implementovány typickým modulárním způsobem jako pluginy (jednotky zvané *PRT*; [4], s. 213–214); mezi ně spadá mj. síťový profil *TCP/IP*, *Bluetooth* a pro nás zajímavý protokol *SMS*, jehož modul lze na obr. 1.1 najít jako blok SMS PRT.

Spojení se *socket serverem* se realizuje voláním metody `Connect()` na objekt třídy `RSocketServ`, který tento komunikační kanál zapouzdřuje. Pak již může následovat samotné otevření socketu asociovaného s určitým protokolem. Socket je abstrahován třídou `RSocket` a u její instance stačí za tímto účelem zavolat metodu `Open()`, která jako své argumenty přijímá referenci na onu relaci se *socket serverem* a dále hodnoty určující konkrétní protokol, který se má použít.

Speciálně pro protokol SMS, s nímž se API „SMS Utilities“ pojí, vypadá otevření socketu, včetně počátečního navázání relace se *socket serverem*, následovně:

Kód 2.1: *otevření socketu pro použití s API SMS Utilities*

```
1 iSocketServ.Connect();
2 iSocket.Open(iSocketServ, KSMSAddrFamily, KSockDatagram, KSMSDatagramProtocol);
```

Figurují zde datové položky `iSocketServ` a `iSocket`, což jsou objekty zmíněných tříd po řadě `RSocketServ` a `RSocket`. Jedná se — stejně jako u jakýchkoli dalších datových položek uvedených v rámci tohoto textu, jejichž identifikátor začíná na ‚`i`‘ — o atributy patřící objektu, který tento kód provádí (jedna z konvencí, jež jsou vyjmenovány v podkap. 1.3).

Po úspěšném otevření socketu bývá zpravidla ještě zapotřebí nastavit jeho lokální adresu. Přidržíme se opět onoho API, jež pro tento účel definuje třídu `TSmsAddr`. Voláním její metody `SetSmsAddrFamily()` lze nastavit jednu z možností, které jsou uvedeny ve výčtu `TSmsAddrFamily`. Ty pak určují, jakým způsobem se má aktivní socket chovat. Bohužel je u položky tohoto výčtu `ESmsAddrRecvAny`, která by potenciálně měla zajistit obdržení veškerých zpráv, poznamenáno, že ji může používat výlučně jen jeden klient, a negativní výsledky experimentů ukázaly, že právě takový klient už nejspíš v systému působí[11]. Ze zbývajících položek výčtu se nabízí volba `ESmsAddrMatchText`, která má, dle poznámky, zpřístupňovat zprávy, jejichž text odpovídá danému vzoru. Ten je možné u objektu zmíněné třídy `TSmsAddr` nastavit pomocí metody `SetTextMatch` a, jak bylo zjištěno, pokud se jí jako argument předá prázdný řetězec (konstanta `KNullDesC8`), potom je dotyčný socket schopen přijímat veškeré příchozí zprávy.

Celý kód takového nastavení lokální adresy socketu, aby skrze něj bylo možné obdržet všechny příchozí zprávy, by mohl vypadat takto:

Kód 2.2: *nastavení adresy socketu pro příjem všech zpráv*

```
1 TSmsAddr smsAddr;                                 // Objekt "adresy" (SMS) socketu.
2 smsAddr.SetSmsAddrFamily(ESmsAddrMatchText); // Získávat všechny příchozí zprávy,
3                                                   // jejichž text odpovídá vzoru.
4 smsAddr.SetTextMatch(KNullDesC8);                 // Tímto vzorem je prázdný řetězec.
5 iSocket.Bind(smsAddr);                            // Samotné nastavení lokální adresy.
```

Posledním zásahem do socketu nutným pro umožnění příjmu zpráv skrze něj, je jeho nastavení do režimu „čtení“, a to tímto způsobem:

Kód 2.3: *nastavení socketu do režimu čtení*

```
1 iPckgBuf() = KSockSelectRead;
2 iSocket.Ioctl(KIOctlSelect, iStatus, &iPckgBuf, KSOLSocket);
3 SetActive();                  // Nastaví příznak, že požadavek na provedení
4                               // asynchronní operace je připraven na své vyřízení.
```

[11] dá se tušit, že jím je standardní systémová aplikace vyhrazená pro uživatelský přístup ke zprávám

kde `iPckgBuf()` je instanční proměnná daného objektu (lokální proměnnou zde, vzhledem k asynchronnímu provádění operace `Ioctl()`, nelze použít) a má typ `TPckgBuf<TUint>`. Za povšimnutí stojí proměnná `iStatus` použitá jako jeden z argumentů i následné volání `SetActive()`.

Dosud byl totiž zamlčen předpoklad, že třída, která ve své implementaci využívá sockety a související asynchronní požadavky, bude dědit od abstraktní třídy `CActive`. Tím se stane *aktivním objektem* (viz odrážka e v podkapitole 1.3) a to umožní její zpětné volání potom, co je vznesený požadavek vyřízen. Lépe je vztah mezi sockety a aktivními objekty vyjádřen v průvodci systémem Symbian [40].

Třída potom vynuceně implementuje čistě virtuální metody z třídy `CActive`, a sice `RunL()` (uvedené zpětné volání, *callback*), `DoCancel()` (zrušení vzneseného požadavku) a volitelně může také předefinovat její metodu `RunError()` (volána, pokud v těle metody `RunL()` dojde k chybě). Zároveň se sdědí její datový člen `iStatus`, který je typu `TRequestStatus`, a právě ten se vkládá jako odkaz do volání asynchronních operací, jako tomu bylo i v kódu 2.3. Zde bylo také následně voláno `SetActive()`, což je metoda implementovaná v téže základní třídě `CActive`. Její význam je uveden v komentáři k tomuto volání; experimentálně bylo zjištěno, že `RunL()` je pak volán až s příchodem zprávy, nikoli po samotném nastavení socketu do režimu „čtení“, jak by se mohlo na první pohled zdát.

Pokud byly provedeny všechny dříve popsané kroky, pak by s výskytem nové příchozí zprávy v systému měl být automaticky zavolán onen *callback* `RunL()`, v němž stačí zprávu pro další zpracování vyčíst ze socketu. To se provede zhruba takto:

Kód 2.4: *vyčtení zprávy ze socketu (pomocí API SMS Utilities)*

```
1 // Buffer, do kterého se načte text zprávy (v kódování Unicode).
2 CSmsBuffer* smsBuffer = CSmsBuffer::NewL();
3
4 // Objekt reprezentující celou zprávu (typu "příchozí zpráva").
5 CSmsMessage* smsMessage = CSmsMessage::NewL(iFs,CSmsPDU::ESmsDeliver,smsBuffer);
6
7 // "Stream" obalující daný socket, z něj se dá zpráva přímo vyčíst.
8 RSmsSocketReadStream smsSocketReadStream(iSocket);
9
10 smsSocketReadStream >> *smsMessage;                 // Vyčtení příchozí zprávy.
11 smsSocketReadStream.Close();                        // Uzavření tohoto "streamu".
```

kde `iFs` je objekt třídy `RFs` zapouzdřující relaci se *serverem souborů* (na obr. 1.1 blok File Server), která je předem ustavena voláním `iFs.Connect()`.

Zpracování takto získané zprávy (`*smsMessage` z kódu 2.4) může spočívat v získání vybraných informací o ní, s nimiž je pak dále nakládáno. Za všechny dostupné údaje jmenujme identifikaci (zpravidla telefonní číslo) odesílatele, kterou lze získat pomocí metody `ToFromAddress()`, a text zprávy přístupný pomocí metody `Buffer()`.

Je důležité si uvědomit, že „odchycením“ zprávy skrze socket už tato nepokračuje standardní cestou do složky příchozích zpráv a je na zodpovědnosti programátora, aby v případě požadavku na toto chování provedl doručení zprávy svépomocí. Existuje sice jedna výjimka, kdy se takto obdržené zprávy i přesto ve složce doručených objeví, ale k tomu dojde až po restartu telefonu, tudíž je to v podstatě nepoužitelné.

Aby se zamezilo tomuto (možná poněkud překvapivému) „zjevení“ zpráv po restartu — objeví se jako nové zprávy, včetně odpovídajícího upozornění na ně — musí se po každém vyčtení zprávy ze socketu signalizovat, že zpráva byla v pořádku obdržena:

Kód 2.5: *signalizace úspěšného převzetí zprávy*

```
1 iSocket.Ioctl(KIoctlReadMessageSucceeded, iStatus, NULL, KSolSmsProv);
2 SetActive();    // Požadavek na provedení asynchr. operace opět čeká na vyřízení.
```

přičemž tentokrát je *callback* `RunL()` zavolán hned, jakmile je toto oznámení provedeno, nikoli až s další příchozí zprávou, jako tomu bylo v předchozím případě. Pokud je zmíněný *callback* volán jako reakce na provedení této operace, je pro příjem dalších zpráv skrze socket dále nutné zopakovat kroky z kódu 2.3. Takto vzniklá potřeba rozlišovat, z jakého důvodu je zrovna `RunL()` voláno, vede na zavedení jednoduchého *stavového automatu*, kdy se před voláním asynchronní operace uloží informace o tomto stavu v instanční proměnné a podle ní se pak řídí akce prováděné v uvedeném *callbacku*.

Při ukončení práce se socketem (např. po zavolání *destruktoru* objektu, který jej vlastní) je nanejvýš vhodné provést uzavření socketu i relace se *socket serverem*:

Kód 2.6: *ukončení práce se socketem*

```
1 iSocket.CancelIoctl();          // Zruší nevyřízené požadavky na Ioctl operace.
2 iSocket.Close();                // Zavře socket.
3 iSocketServ.Close();            // Ukončí relaci se socket serverem.
```

Ná závěr k tomuto API poznamejme, že v jeho distribuci, podobně jako u ostatních doplňkových rozhraní, není obsažena knihovna použitelná při sestavení aplikace pro emulátor, testování tak lze provádět pouze přímo na mobilním telefonu.

2.4 Doručení zpráv do cílové složky

Protože se socketem musí odchytávat veškeré příchozí zprávy, a to i ty, u nichž se později zjistí, že nepodléhají žádnému uživatelsky nastavenému filtru, je nutné ve vlastní režii vyřešit, jak posléze zprávy doručit do cílové složky zpráv a zároveň uživatele upozornit buď systémovým, nebo uživatelsky definovaným způsobem.

Bylo zjištěno, že pokud se vytvoří zpráva s příznakem „nepřečtená“ ve složce doručených zpráv (*Přijaté/Inbox*), reaguje systém na tuto událost tím, že provede stejné akce, jako při běžném obdržení zprávy — v závislosti na aktuálním profilu se přehraje zvuk a/nebo aktivuje vibrační prvek telefonu a na displeji se zobrazí upozornění na novou zprávu. Toho lze ve výsledné aplikaci s výhodou využít pro případ, že příchozí zpráva nebude vyhovovat žádnému uživatelem definovanému filtru.

Problematické však nadále zůstává doručení ostatních zpráv, u nichž se požaduje vlastní způsob jejich ohlášení, jelikož do něj by systém takto vstupovat neměl. Po sérii pokusů se jako nejoptimálnější řešení ukázal postup, kdy se, v závislosti na konkrétním aplikovaném filtru, zpráva doručí do některé z podsložek „vlastní složky zpráv“ (*Mé složky/My Folders*). Zvažováno bylo sice také vytváření zpráv přímo v oné „vlastní složce“, ovšem zvolené řešení může mít ve svém důsledku pro uživatele aplikace větší přínos, protože filtrované zprávy jsou zároveň také automaticky řazeny do podsložek, a tak jsou lépe dohledatalné než v případě neorganizované kupy zpráv v jedné či dvou složkách[12].

Zmíněný příznak zpráv „nepřečteno“ v tomto případě už nezpůsobí spuštění systémového upozornění, jeho použití navíc uživateli umožní snadno najít nepřečtené zprávy díky implicitnímu grafickému odlišení takových zpráv, potažmo i podsložek, které je obsahují.

[12]na druhou stranu je třeba dodat, že např. některé aplikace pro správu telefonu z PC, konkrétně *Nokia Communication Centre* z balíku *Nokia PC Suite* zprávy ve „vlastní složce zpráv“ zcela ignorují — tento problém však není zcela neřešitelný

Přehrání zvuku i další způsoby upozornění na novou filtrovanou zprávu se pak pochopitelně musí provést vlastními prostředky.

Řešení otázky doručení zprávy do cílové složky z pohledu API je vcelku přímočaré. Uplatní se *message server* a související třídy částečně popsané v předchozí podkapitole 2.2 (odst. „Ad 1“) a dále zatím jen zmíněný modul *SMS MTM*.

V prvé řadě, pokud se má zpráva doručit do podsložky zmíněné „vlastní složky“, je nutné nejdříve zjistit, zda již daná konkrétní složka neexistuje, a pokud ano, pak ji použít, v opačném případě musí být taková složka nejdříve vytvořena — pokusy ukázaly, že pokud by se tato kontrola vynechala a nová podsložka vytvářela vždy, vznikaly by nežádoucí duplicity složek.

Každý záznam *serveru zpráv* má svůj jedinečný identifikátor typu `TMsvId`. Konkrétní identifikátory pro běžné systémové složky zpráv, jako je složka příchozích či odeslaných, lze najít v hlavičkovém souboru dodaným v SDK (`msvids.h`), identifikátor pro „vlastní složku zpráv“ zde však bohužel chybí. Naštěstí je tato nedokumentovaná konstanta díky zvídavým vývojařům dohledatelná na diskusním fóru [13] (má hodnotu `0x1008`).

Pro získání všech podsložek „vlastní složky zpráv“ může sloužit tento kód:

Kód 2.7: *získání všech podsložek složky My Folders/Mé složky*

```
1 // Získá přístup k "vlastní složce zpráv" v rámci Message Serveru
2 CMsvEntry* myFoldersEntry = iMsvSession->GetEntryL(TMsvId(0x1008));
3
4 // Získá seznam jedinečných identifikátorů podsložek "vlastní složky zpráv".
5 CMsvEntrySelection* subfoldersIds =
6     myFoldersEntry->ChildrenWithTypeL(KUidMsvFolderEntry);
```

kde proměnná `iMsvSession` je objektem již uvedené třídy `CMsvSession` a představuje relaci se *serverem zpráv*, otevřenou popsaným způsobem (viz podkap. 2.2, odst. „Ad 1“). Přístup ke konkrétní podsložce se pak realizuje podobně jako na řádku 2 kódu 2.7 a odsud pak lze voláním na způsob `subfolderEntry->Entry().iDetails` obdržet název této podsložky.

Vytvoření nové podsložky zmíněné „vlastní složky zpráv“ lze v návaznosti na kód 2.7 dosáhnout takto:

Kód 2.8: *vytvoření podsložky zadané složky (navazuje na kód 2.7)*

```
1 _LIT(KSubfolderName, "Example subfolder");      // Název podsložky jako literál.
2 TMsvEntry subfolder;                 // Záznam, jaký si Message Server uchovává
3                                      // ve svém indexu, zde bude mít roli složky.
4 subfolder.iType = KUidMsvFolderEntry;           // Typ záznamu složka.
5 subfolder.iMtm = KUidMsvLocalServiceMtm;        // Neváže se na konkrétní MTM.
6 subfolder.iServiceId = KMsvLocalServiceIndexEntryId;  // Mezi lokálními složkami.
7 subfolder.iDetails.Set(KSubfolderName);         // Nastavení názvu podsložky.
8
9 // další případná nastavení vlastností podsložky...
10
11 myFoldersEntry->CreateL(subfolder);            // Vloží se jako potomek
12                                               // "vlastní složky zpráv".
```

Pro nástin, jakým způsobem vytvořit zprávu ve složce zpráv se spokojme se stránkou Wiki portálu Fora Nokia [26]. Tamní příklad ukazuje, že nejprve je nutné se skrze *message server* a dále registr dostupných MTM (představovaný třídou `CClientMtmRegistry`) dostat ke klientskému rozhraní *SMS MTM* zapoudřenému v obdržené instanci třídy `CSmsClientMtm`.

Pomocí ní se nastaví kontext v rámci záznamů na složku, v níž se má zpráva vytvořit, a následně se tak učiní. Za povšimnutí stojí, co vše se nastavuje: položky SMS hlavičky, položka indexu pro server zpráv (opět viz podkap. 2.2, odst. „Ad 1“) a samotné tělo zprávy.

2.5 Práce s databází kontaktů

Zaměřme se teď na požadavek schopnosti rozlišovat příchozí zprávy podle příslušnosti jejího odesílatele ke skupině (viz podkap. 2.1). Půjde nám hlavně o způsob získání této informace v situaci, kdy známe telefonní číslo odesílatele zprávy a hledáme potenciální shodu v dané množině skupin kontaktů. Je důležité si zároveň uvědomit, že jeden kontakt může patřit do obecně více skupin.

Správu kontaktů v Symbianu zajišťuje komponenta *Contacts Model* (viz stejnojmenný blok vyznačený na obr. 1.1). Příslušné veřejné API se skládá ze tří hlavních, hierarchicky závislých tříd: `CContactDatabase` představující databázi kontaktů, tu tvoří jednotlivé kontakty (těmito položkami mohou být i skupiny a šablony kontaktů), přičemž s kontaktem lze manipulovat prostřednictvím základní třídy `CContactItem`, a každý kontakt se dále skládá z dílčích datových polí jako jméno a příjmení, nad kterými lze operovat prostřednictvím třídy `CContactItemField` [36]. Podobně jako u záznamů *serveru zpráv* (podkap. 2.4) má i každý kontakt svůj unikátní identifikátor (`TContactItemId`).

Výchozí databáze kontaktů (mimo ni mohou existovat i další, ovšem tím se nemusíme zatěžovat) se aplikaci zpřístupní takto:

Kód 2.9: *přístup do výchozí databáze kontaktů*

```
1 CContactDatabase* contactDb = CContactDatabase::OpenL();
```

Potom již je možné ukazatel `contactDb` v dalším kódu používat k operacím nad touto databází. Logickou úvahou lze dojít k závěru, že pokud kontakt s daným telefonním číslem v databázi neexistuje, nemůže být ani zařazen do žádné skupiny. Lze tedy na `*contactDb` zavolat metodu, která vrátí seznam identifikátorů kontaktů, u kterých se telefonní číslo shoduje s číslem, zadaným ve formě „řetězce“[13]:

Kód 2.10: *vyhledání kontaktů se shodou na zadané tel. číslo*

```
1 _LIT(KTelnum, "+420123456789")                    // Hledané číslo jako literál.
2 CContactIdArray* contactIds = contactDb->MatchPhoneNumberL(KTelnum,12);
```

kde druhý argument udává, na kolika místech zprava se musí telefonní čísla shodovat[14]. Pokud je vrácený seznam prázdný, znamená to, že kontakt s daným číslem v databázi neexistuje a zpracování tím končí. Situaci, kdy jednomu číslu odpovídá více než jeden kontakt, lze považovat za natolik vzácnou, že nám v případě neprázdného seznamu postačí pro další zpracování pouze jeho první položka (identifikátor kontaktu). Následující kód rovněž ukazuje, jak k tomuto kontaktu zjistit, do kterých skupin kontaktů patří:

Kód 2.11: *zjištění skupin ke kontaktu (navazuje na kód 2.10)*

```
1 if (contactIds->Count() > 0)
2     // Alespoň jeden kontakt s daným číslem v databázi existuje.
3     {
4     // Zpřístupni první odpovídající kontakt (i kdyby jich bylo více).
5     CContactItem* contactItem = contactDb->ReadContactL((*contactIds)[0]);
```

[13] řetězcem se zde míní deskriptor nebo literál (viz podkap. 1.3, odrážka h)

[14] dle dokumentace větší hodnota, než je délka zadaného „řetězce“ nevadí, doporučeno je zadávat hodnotu větší nebo rovnu 7

```
6
7     // Získej k němu seznam identifikátorů skupin kontaktů, do kterých patří.
8     CContactIdArray* contactGroupIds =
9         static_cast<CContactCard*>(contactItem)->GroupsJoinedLC();
10    }
```

Poznamenejme, že třída `CContactCard`, na níž je nejprve nutné zpřístupněný kontakt přetypovat, je specializací třídy `CContactItem` a slouží výhradně pro kontakty jako takové (nikoli jejich skupiny či šablony). Získanou informaci o dotyčných skupinách pak lze porovnat se skupinami z uživatelem definovaných filtrů.

Ty v rámci nich nemá cenu ukládat jako jejich názvy, ale pouze jako zmíněné identifikátory. To značně zjednoduší zmíněné porovnávání a zajistí flexibilitu v případě, že danou skupinu uživatel později přejmenuje (identifikátor zůstane stále stejný).

2.6 Přehrání zvuku

Přejděme nyní k fázi upozornění uživatele v situaci, kdy se k dané příchozí zprávě nalezlo odpovídající pravidlo a tudíž se neuplatní upozornění systémové. Těžiště bude spočívat v přehrání uživatelem tónu.

Pro přehrávání a záznam zvuku i videa slouží v Symbianu specializovaný framework pro multimédia, na obr. 1.1 jej představuje blok Multimedia Framework (zkráceně *MMF*). Jeho strukturu, která je, jak je již zvykem, založená na pluginech, ilustruje obrázek „Multi Media Framework — Client Utility APIs“ z průvodce Symbianem [38] a nejdůležitější třídy jsou přehledně zachyceny na obrázku „Audio class diagram“ v knize [1] (obr. 13.1, s. 280).

Přehrání zvukového souboru bez ohledu na jeho konkrétní formát (musí však být přítomen MMF plugin, který jej implementuje) se provádí skrze rozhraní třídy `CMdaAudioPlayerUtility`. Jelikož se požadavek na přehrání i některé další operace drží zavedeného asynchronního způsobu zpracování, obsahuje framework také callback rozhraní, které je v tomto případě zastoupeno třídou `MMdaAudioPlayerCallback`.

Od něj obyčejně dědí přímo třída, jež vlastní objekt oné třídy `CMdaAudioPlayerUtility`, přičemž jako referenci na tento callback, požadovanou při jeho konstrukci, vkládá sama sebe. Rozhraní callbacku se skládá ze dvou metod: `MapcInitComplete()` volané v okamžiku, kdy byl dokončen požadavek na otevření a přichystání zadaného zvukové souboru (ať úspěšně či nikoli), a `MapcPlayComplete()`, který je zavolán po dokončení požadavku na jeho přehrání.

Pro aplikaci nejdůležitější metody zmíněné třídy `CMdaAudioPlayerUtility` jsou `NewL()`, která má roli konstruktoru, `OpenFileL()`, která připraví zadaný soubor k přehrátí, `Play()`, který pak toto přehrání zajistí, a eventuálně `Stop()`, která jej ukončí.

Konkrétní ukázku použití tohoto API pro přehrání zvukového souboru dostatečným způsobem podává stránka na Wiki portálu Fora Nokia [27].

2.7 Informace o aktuálním profilu

Zkoušky ukázaly, že aktuální nastavení profilu (např. tichý profil), nemá na přehrání zvuku žádný vliv. Je zcela jistě žádoucí, aby se buď napevno, nebo volitelně podle preferencí uživatele nastavení v profilu zohlednilo.

V možnosti přístupu k profilům a podrobnostem o nich se bohužel SDK (potažmo API) pro počáteční verzi S60 3rd a SDK pro novější verze rozchází. V prvním případě toho lze dosáhnout pomocí neveřejného API, které je, podobně jako API „SMS Utilities“ (viz

podkap. 2.2, odst. „Ad 1“), zprostředkováno v balíku doplňkových API pro C++ SDK této konkrétní verze S60 (dostupný je skrze [14]). Zbývající případy pokrývá standardní veřejné API „Profiles Engine Wrapper“, jež bylo zavedeno od verze S60 3rd, FP1.

Obě zmíněná API si jsou sice svým použitím velmi podobná, ovšem vzhledem k drobným odlišnostem je potřeba pro podporu jak počáteční verze S60 3rd, tak i verzí novějších sáhnout k podmíněnému překladu řízenému symbolickou konstantou `__SERIES60_30__`, kterou definuje pouze SDK počáteční verze S60 3rd.

Tato API velmi zjednodušují výběr kritéria pro posouzení, zda je v závislosti na aktuálním profilu vhodné zvuk přehrát či nikoli, a to prostřednictvím metody `IsSilent()` volané na objekt třídy zapouzdřující konkrétní profil se svými nastaveními (`MProfile`/`MProEngProfile`). Tato metoda vrátí informaci, že profil je tichý, pokud je „typ vyzvánění“ („ringing type“) nastaven na hodnotu „tichý“ („silent“) a/nebo jsou vypnuty zvuky všech upozornění.

Objekt zmíněné třídy, který by představoval aktuální profil, lze získat jednotně voláním metody `ActiveProfileL()` na objekt třídy, která tvoří rozhraní nad subsystémem profilů (`MProfileEngine`/`MProEngEngine`). Ten se v prvním případě získá voláním dovezené funkce `CreateProfileEngineL()`, ve druhém voláním metody `NewEngineL` na *tovární třídu* `ProEngFactory`.

Výše uvedené konstrukce demonstruje následující úryvek kódu zohledňující obě API:

Kód 2.12: *zjištění, zda je aktuální profil ,,tichý‘‘*

```
// Přístup k subsystému profilů a následné získání aktuálního profilu.
#ifdef __SERIES60_30__
MProfileEngine* profileEngine = CreateProfileEngineL();
MProfile* profile = profileEngine->ActiveProfileL();
#else
MProEngEngine* profileEngine = ProEngFactory::NewEngineL();
MProEngProfile* profile = iProfileEngine->ActiveProfileL();
#endif

TBool isSilent = profile->IsSilent();           // Zjisti, zda je profil "tichý".

// Uvolnění prostředků.
profile->Release(); profile = NULL;
profileEngine->Release(); profileEngine = NULL;
```

2.8 Ovládání vibračního prvku telefonu

Pokud se spokojíme s jednoduchým ovládáním vibračního prvku telefonu, kdy jej pouze aktivujeme s předem určenou délkou trvání, pak je řešení z pohledu API velice přímočaré.

Tuto činnost zajišťuje „Vibra API“, zastoupené třídou `CHWRMVibra`. Přistupme rovnou k jednoduché ukázce použití, která nepotřebuje dalšího vysvětlování:

Kód 2.13: *aktivace vibračního prvku telefonu po dobu 1s*

```
CHWRMVibra* vibra = CHWRMVibra::NewL();
if (vibra->VibraStatus() != CHWRMVibra::EVibraStatusNotAllowed)
    // Pokud nejsou vibrace v profilu vypnuty.
    {
    // Spusť vibrace po dobu jedné sekundy (1000ms), synchronně.
    iVibra->StartVibraL(1000);
    }
delete vibra; vibra = NULL;
```

Kapitola 3

Realizace demonstrační aplikace

Přichází vyústění práce v podobě zprávy o praktickém provedení aplikace. Napřed jsou shrnuty dostupné vývojové prostředky a učiněna některá pro další postup významná rozhodnotí, pak již pojednávám o etapách vývoje aplikace. Vycházím přitom z dříve uvedených poznatků.

3.1 Přípravná fáze, vývojové prostředky

V současné době Nokia podporuje vývoj aplikací pro platformu S60 vedený v jazyce C++ oficiálně pouze na nejnovějších verzích operačního systému pro PC Windows z provenience společnosti Microsoft. Takto jsou totiž limitovány jak odpovídající balíky SDK, tak i momentálně jediné Nokií nabízené *integrované vývojové prostředí* (*IDE*[1]) s názvem *Carbide.c++* (toho času ve verzi 2.0).

Pro lepší představu o takovém SDK si uveďme, co všechno zahrnuje: překladový systém (jak pro koncová zařízení, tak pro emulátor), emulátor, hlavičkové soubory, knihovny a dokumentaci k veřejným API (jak platformy S60, tak i Symbianu samotného), ukázkové aplikace (demonstrující možnosti opět jak platformy, tak i samotného systému[2]) a dále některé další podpůrné nástroje, které např. umožňují simulovat různé speciální události během testování.

IDE Carbide.c++ je postavené na vývojové platformě *Eclipse*[3] a z jeho vlastností za vyzdvihnutí stojí[4] dobrá a komfortní podpora pro ladění aplikací (to může probíhat jak v emulátoru, tak v koncovém zařízení[5]), průvodci vytvořením nového projektu či třídy, které pokrývají nejčastější potřeby vývojářů a postarají se o vytvoření základní kostry daných součástí projektu, a dále také vestavěný nástroj s názvem *UI Designer*. Použitím tohoto „návrháře UI“ si lze ulehčit práci při tvorbě uživatelského rozhraní aplikace, jehož vzhled se vytváří vizuální formou, intuitivně pomocí přetahování komponent UI myší. Tímto způsobem se dá definovat jak obsah *hlavního pole* okna aplikace, tak také nabídky, sdělení, dialogy i volby v *poli ovládání* a dílčí složky *stavového pole* (viz podkap. 1.2.3). Zároveň s úpravami UI se dynamicky generuje odpovídající kód. Bližší informace o tomto IDE, spolu s možností jeho stažení, podá webová stránka [12].

[1] Integrated Development Environment

[2] ukázky aplikací ze strany Symbianu používají ze zřejmých důvodů pouze textový výstup, nikoli GUI

[3] viz `<http://www.eclipse.org/>`

[4] pomineme-li kvality samotného Eclipse jako automatické dokončování kódu (Content Assist), hypertextový pohyb skrze identifikátory v kódu a kontextovou nápovědu k nim

[5] tzv. *on-device debugging*

Refenčním přístrojem pro praktické zkoušky aplikace bude primárně telefon Nokia 6120 classic[6]. Proto jsem zvolil jako hlavní SDK pro S60 3rd Edition, FP1. Budu však také usilovat o kompatibilitu „dolů“, na úroveň 3rd Edition (počáteční vydání), čemuž byla předem podřízena celá kapitola 2. Přidržím se oné standardní linie vývoje včetně použití IDE Carbide.c++.

Vzhledem k tomu, že je možné lokalizovat také zobrazovaný název aplikace, a ten by zároveň měl být přiměřeně krátký, pojmenoval jsem aplikaci jako Hlásič SMS v české a SMS Alerts v anglické mutaci. Dále v textu budu používat českou variantu.

3.2 Návrh aplikace

Jak bylo uvedeno (podkap. 1.1.2), Symbian nabádá k dělbě aplikace na část týkající se UI a na část představující hlavní logiku aplikace, což umožňuje do jisté míry nezávislý vývoj těchto celků.

Návrh uživatelského rozhraní

Pokud jde o UI, od počátku jsem o něm měl jistou představu. Předznamenám jen, že se následující řádky notně opírají o teorii uvedenou v podkap. 1.2.3 a 1.3.1. Vycházel jsem hlavně z uvedené filozofie UI „výčet prvků – zobrazení detailů“, další inspiraci jsem čerpal přímo u integrovaných aplikací platformy S60.

Hlavní obrazovku Hlásiče SMS bude představovat seznam — ve formě listboxu — uživatelem definovaných položek filtrů, jimiž se bude dle shody s konkrétním pravidlem řídit způsob ohlášení zprávy. Po výběru konkrétního filtru či během přidání nového se pak zobrazí pohled umožňující editaci jeho položek nastavení. Protože položky listboxu jsou uspořádány vertikálním způsobem, lze požadavek na nezatajení informace o pořadí, v jakém se filtry prochází (viz podkap. 2.1) považovat za částečně splněný (viz dále).

Tuto představu jsem realizoval a dále rozvinul pomocí zmíněného nástroje UI Designer, který mě sám navedl na architekturu aplikace s přepínání pohledů. Vytvořil jsem výčtový pohled (view) obsahující *selection list* a pohled editační, který je vyplněn prvkem *setting item list*. Toto by bylo dostačující v případě, že by Hlásič SMS uživateli nabízel pouze jediné kritérium filtrování příchozích zpráv.

Pokud by jich měl zohledňovat více (na což také naráží jeden z požadavků na aplikaci, viz podkap. 2.1), bylo by nutné přinejmenším připravit další editační pohled(y). To proto, že se možnosti nastavení filtru pro jednotlivá kritéria budou zřejmě lišit, byť samotné položky nastavení upozornění na danou zprávu budou u všech jedny a tytéž. Všechny filtrační položky by se pak musely zobrazovat ve společném seznamu (např. graficky rozlišené pomocí ikon), což by však mohlo být mírně nepřehledné. Rozhodl jsem tudíž, že ke každému kritériu bude zároveň existovat samostatný výčtový pohled se seznamem příslušných filtrů. Přepínání mezi těmito výčtovými prvky bude vyřešeno typickým způsobem — pomocí záložek zobrazených v poli navigace, jimiž se lze přesouvat pomocí kláves „vlevo“/„vpravo“.

Pro každý pohled vygeneroval UI Designer odpovídající třídy jak pro samotné pohledy (ty dědí z třídy aplikačního frameworku `CAknView`), tak pro prvky, které obsahují — seznamy položek nastavení (editační pohled), resp. obecné prvky UI, jež vlastní listboxy (výčtový pohled). Postaral se také o vytvoření odpovídajících souborů „zdrojů“, tzv. *resources*[7], mezi

[6] viz `<http://www.forum.nokia.com/devices/6120_classic>`

[7] mají koncovku `rss`

nimiž je i hlavní *resource* soubor aplikace, kde jsou specifikovány zobrazovaný název aplikace, její ikona a další důležité atributy. Spolu s nimi připravil příslušné soubory s lokalizací[8], kde jsou konkrétní definice pro jinde symbolicky používané řetězce.

Protože je u všech výčtových, resp. editačních pohledů vyžadováno stejné základní chování (např. možnost smazat vybranou filtrační položku v seznamu), rostla během dalšího vývoje současně redundance takového kódu. Tu jsem později vyřešil přesunem společné funkcionality do nově vytvořených základních tříd, od nichž pak dědí třídy vztažené na konkrétní kritéria filtrování. Tím jsem se sice odřízl od možnosti dalších úprav s využitím nástroje UI Designer, ale to už v této fázi nevadilo, ba naopak, umožnilo mi vyčistit kód i další soubory od nepřehledných strojově vkládaných komentářů, které tento návrhář UI používal jako záchytné značky při automatickém generování kódu.

Zbývající část aplikace, tedy samotnou její logiku, lze dále rozlišit na engine, víceméně autonomní jednotku, která se stará o zpracování příchozích zpráv, a datový model aplikace.

Model dat

V obecném měřítku se lze na data aplikace dívat ze dvou úhlů: persistentní data uložena v *nevolatilní* (stálé) paměti a nestálá data v operační paměti, která existují pouze dynamicky za běhu aplikace a s jejím ukončením (případně vypnutím telefonu) dojde k jejich ztrátě. Výhodou operační paměti (*RAM*[9]) je, že bývá obvykle podstatně rychlejší. Vrstvu persistentních dat musí Hlásič SMS implementovat tak či tak (viz podkap. 2.1), je však třeba rozhodnout, kterým z následujících obecných způsobů se v návaznosti na tato data postavit k paměti RAM:

1. do RAM se načtou potřebná persistentní data pouze na co nejkratší dobu, což vede na časté čtení z permanentní paměti
2. jako předchozí případ, ale v RAM se uchovávají kopie nejčastěji čtených i zapisovaných údajů (tzv. *caching*), při požadavku na jejich čtení pak není třeba „sahat“ až do permanentní paměti, použije se tato datová *cache*
3. v RAM je uchováván aktuální obraz předem vybraných položek persistentních dat (ty, které se čtou nejčastěji nebo k nimž je potřeba rychlý přístup pro čtení), ty se jednorázově přečtou z permanentní paměti (např. při spuštění aplikace) a dále pak tato paměť slouží pouze pro jejich zápis (manipulace jako přidání, změna, odebrání údaje), pro jejich čtení se uplatní tato datová cache
4. jako předchozí případ, ale v RAM se uchovává kompletní aktuální obraz persistentních dat

Vzhledem k tomu, že pro nejčastěji požadované operace s daty, jako je získání textových popisků položek v seznamech filtrů a vyhodnocování shody s konkrétním pravidlem, se používá pouze malá část ze všech údajů ukládaných o položce filtru, a dále vzhledem k předpokládané delší době běhu Hlásiče SMS jsem zvolil možnost 3. Datová cache také zabrání nadměrně častému překladu mezi identifikátory skupin kontaktů a jejich názvem (viz závěr podkap. 2.5).

Protože Symbian nabízí více způsobů, jak zajistit persistenci dat — mimo souborový systém i prostředky vyšší úrovně: uložiště dat založené na datových proudech (*streams*) vhodné

[8] mají koncovku `rls`

[9] *random access memory*

pro persistenci objektů, podpora pro *XML*[10] a databázově založená persistence dat (skrze nativní API nebo nověji také pomocí podmožiny jazyka *SQL*[11]) — jsem se dále rozhodl, že přístup k permanentní paměti implementuji modulárně, s využitím abstraktního rozhraní. To do budoucna umožní bez výraznějších zásahů do kódu aplikace přecházet mezi uvedenými prostředky persistence dat. Zároveň budu takto vytvořený model dat spravovat zcela ve vlastní režii, metody pro persistenci dat nabízené třídou dokumentu (viz podkap. 1.3.1) ponechám nevyužité.

Model výkonné jednotky aplikace

Návrh enginu využívá zjištění uvedená v kapitole 2. Je „zavěšen“ na *třídu dokumentu* (viz podkap. 1.3.1) a navenek komunikuje pouze s objektem představujícím v předchozích odstavcích zmíněnou datovou cache. Používá jej ke zjištění, zda se pro zvolené kritérium filtrování příchozích zpráv a pro zadanou relevantní hodnotu aktuální zprávy (např. telefonní číslo odesílatele) nalezne shoda s některým z příslušných pravidel — pokud ano, obdrží také odpovídající nastavení, jakým způsobem se má tato zpráva ohlásit. Vnitřně je engine logicky členěn na moduly, které se starají (v pořadí jejich uplatnění) o:

- obdržení zprávy (třída `CSaEngineModuleReceive`)
- doručení zprávy do cílové složky (tř. `CSaEngineModuleDispatch`)
- upozornění zvukem a/nebo vibracemi (tř. `CSaEngineModuleNotify`)
- zobrazení informace o nové zprávě (tř. `CSaEngineModuleNotifyUI`)

K posledně jmenovanému modulu bych měl dodat, že operuje s prvkem UI, čímž porušuje pravidlo oddělení UI od další logiky aplikace. Měl jsem k tomu však dobrý důvod. Engine totiž, jak bylo zmíněno, představuje téměř zcela izolovanou jednotku v rámci běhu aplikace a proto přenést provedení tohoto dílčího úkolu ke zbytku UI by znamenalo „protunelovat se“ (vytvořit rozhraní pro přenos požadavku na zobrazení této informace) skrze nezvykle vysoké množství tříd. Třída `CSaEngineModuleNotifyUI` se dále ještě od ostatních uvedených liší tím, že ji nevlastní přímo hlavní část enginu (třída `CSaEngineCore`), nýbrž až jeho modul zastoupený třídou `CSaEngineModuleNotifyUI`. To usnadní implementaci ukončení přehrávání (vibrací) v případě, že uživatel zobrazené oznámení o nové zprávě uzavře dříve, než je příslušný tón vůbec dohrán.

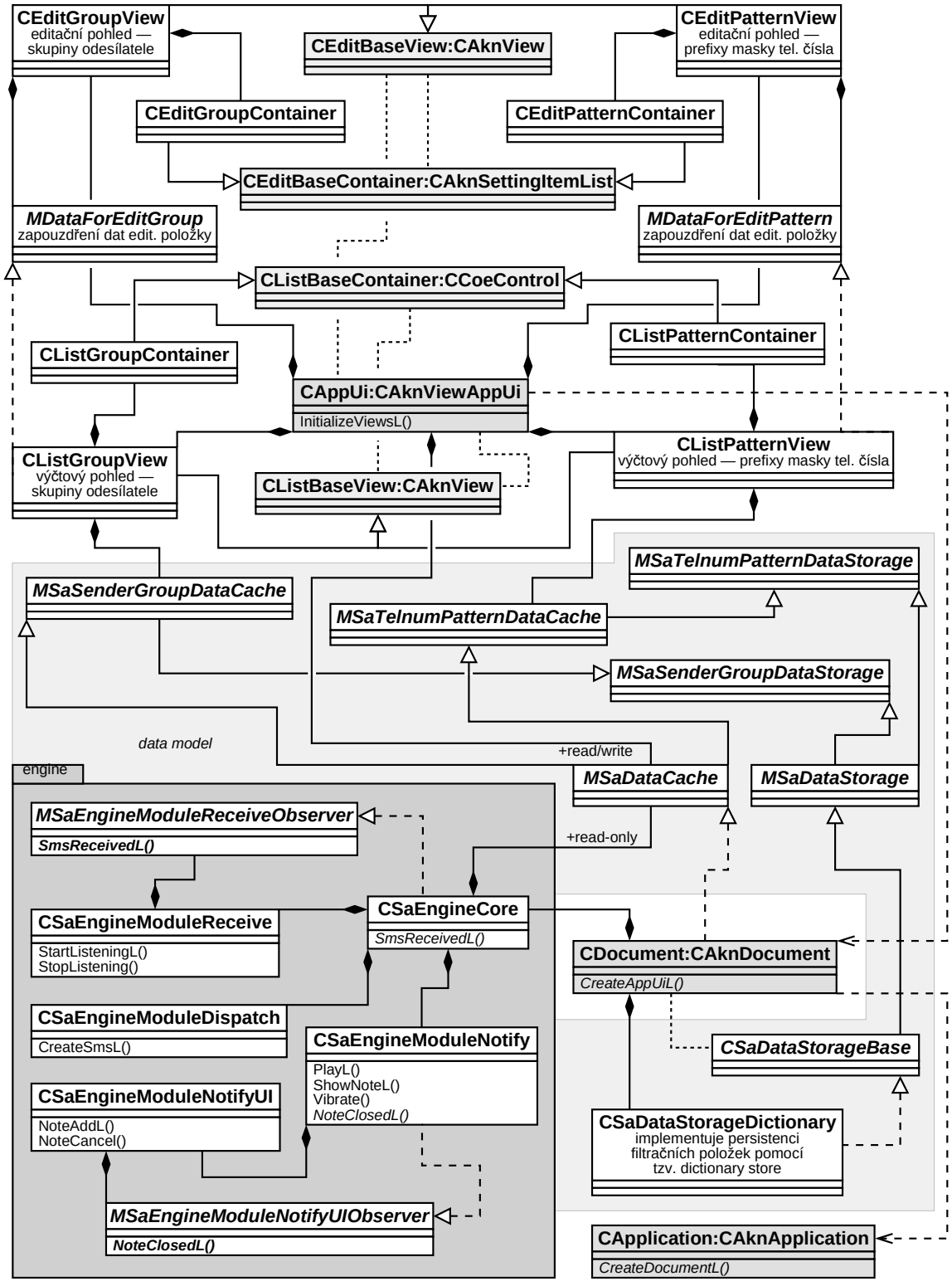
Diagram tříd

Po několika iteracích vývoje se architektura Hlásiče SMS ustálila v podobě, jak naznačuje diagram tříd v duchu *UML*[12] na obr. 3.1. Jak je patrné, aplikace v zachyceném stavu používá dvě filtrovací kritéria: příslušnost odesílatele zprávy ke skupině kontaktů a prefix masky jeho telefonního čísla.

[10] *Extensible Markup Language*, *značkovací jazyk* obecného použití, viz `<http://en.wikipedia.org/wiki/XML>`

[11] *Structured Query Language*, „klasický“ jazyk na poli relačních databází, viz `<http://en.wikipedia.org/wiki/SQL>`

[12] viz `<http://en.wikipedia.org/wiki/Class_diagram>`



Obrázek 3.1: Diagram tříd aplikace v pokročilejší fázi vývoje

3.3 Implementační detaily

Implementace Hlásiče SMS probíhala v úzké návaznosti na související části návrhu a každý dokončený úsek jsem následně testoval na referenčním přístroji, abych případně neinvestoval příliš mnoho úsilí do slepé cesty vývoje. Usoudil jsem, že i do budoucna bude lepší, pokud komentáře ke kódu budu psát v angličtině, která je *de facto* standard pro tuto činnost. Zároveň se tak vyhnu jazykové rozpolcenosti, kde by se čeština střetávala s angličtinou výhledově použitou např. v identifikátorech.

Mnoho při implementaci použitých technik již bylo uvedeno v předchozím textu (zejména v kap. 2 a podkap. 3.2 věnované návrhu aplikace), proto poukáži pouze na některé vhodné body implementace.

Kritéria filtrování

Nakonec jsem k předem danému kritériu pro filtrování podle příslušnosti odesílatele zprávy ke skupině kontaktů přidal ještě ono kritérium prefixu masky telefonního čísla navržené v části specifikace požadavků na aplikaci (podkap. 2.1). Prezentace jednotlivých položek filtrů podle těchto kritérií se pak řídí zásadou na individuální pohledy editační i výčtové, již jsem si stanovil ve fázi návrhu (viz podkap. 3.2, část „Návrh uživatelského rozhraní“).

Persistence dat

Rozhraní pro přístup k permanentní paměti (viz podkap. 3.2, část „Model dat“) jsem realizoval s využitím prostředku Symbianu poskytujícího ukládání dat založené na zmíněných streamech. Konkrétně jsem zvolil jeho variantu „slovníkové uložiště“ (*dictionary store*), jehož rozhraní představuje abstraktní třída `CDictionaryStore` a které podrobněji popisuje příručka Symbianu [37].

V rámci tohoto uložiště lze vytvářet jednotlivé datové streamy, k nimž se přistupuje pomocí jejich jednoznačných identifikátorů, které lze zjednodušeně považovat za celočíselný datový typ. Toto mapování se uchovává v kořenovém streamu tohoto uložiště. Persistenci dat s použitím souborového systému pak umožňuje konkrétní provedení tohoto rozhraní v podobě třídy `CDictionaryFileStore`. V aplikaci je přítomna jako datový člen třídy `CSaDataStorageDictionary` (viz diagram tříd na obr. 3.1), která implementuje zmíněné vlastní rozhraní pro persistenci dat.

Z pohledu aplikace toto uložiště používám tak, že mám pro jednotlivá kritéria pro filtraci příchozích zpráv předem definovány identifikátory streamů, v nichž uchovávám informaci o počtu příslušných uživatelem definovaných filtrů. Bezprostředně následující identifikátory jsou pak vyhrazeny pro data těchto filtrů, jedné položce filtru přísluší jeden stream. To si mj. vyžaduje kontroly, aby nedošlo k přetečení položek filtrů jednoho kritéria do oblasti vyhrazené pro kritérium jiné. To jsem vyřešil stanovením pevného maxima počtu položek v rámci jednoho kritéria[13].

Výměna informací mezi pohledy

U pohledů jsem potřeboval najít způsob, jakým si spolu výčtový a editační pohled mají vyměňovat informace, pokud je vyvoláno jejich vzájemné přepnutí.

V případě přepnutí z výčtového pohledu na editační musí totiž výčtový pohled, dříve než je jako důsledek tohoto přepnutí deaktivován, dát nějakým způsobem editačnímu pohledu

[13] 255 položek, což pro běžné použití postačuje s velkou rezervou

najevo, která položka filtru se má editovat, případně mu o takové položce přímo poskytnout jednotlivé údaje. To jsem vyřešil tak, že jsem vytvořil pro každé kritérium abstraktní mixin třídu (viz `MDataForEditGroup` a `MDataForEditPattern` v diagramu tříd na obr. 3.1), skrze jejíž rozhraní je možné obdržet nebo naopak uložit údaje o dané položce filtru. Toto rozhraní je implementováno na straně výčtového pohledu a reference na tuto třídu se pak poskytne pohledu editačnímu. Před samotným přepnutím pohledů z výčtového na editační se na straně výčtového pohledu předchystají příslušné údaje, ty pak mohou být v editačním pohledu upraveny, a výčtový pohled s nimi pak může při své opětovné aktivaci provést další operace.

V případě přepnutí v obráceném směru, tedy z editačního pohledu na výčtový, je již problém sdílený dat výše uvedeným způsobem vyřešen. Je ovšem nutné výčtovému pohledu sdělit, zda byl editační pohled ukončen s požadavkem na potvrzení či zamítnutí provedených změn, a ten se podle toho patřičným způsobem zachová.

Naštěstí metoda `ActivateLocalViewL()` třídy uživatelského rozhraní (viz podkap. 1.3.1), která umožňuje přepnutí na definovaný pohled v rámci téže aplikace, má i svou přetíženou verzi, díky níž lze tomuto pohledu zároveň zaslat „zprávu“. Toho v aplikaci využívám mj. právě na zmíněné rozlišení, zda byly provedené změny schváleny.

Zaslání aplikace na pozadí

Z pohledu implementace nepatrnou, ale z hlediska použitelnosti poměrně zásadní vlastností Hlásiče SMS je jeho přepnutí na pozadí, pokud uživatel na hlavní obrazovce aplikace (tj. v některém z výčtových pohledů) stiskne pravé výběrové tlačítko, přičemž o tomto chování informuje jeho popisek „Zpět“ („Back“). Toto tlačítko totiž obvykle slouží k ukončení aplikace, což bývá avizováno popiskem „Konec“ („Exit“). Ukončit Hlásiče SMS lze nadále standardní cestou pomocí stisknutí tlačítka pro ukončení hovoru, případně výběrem odpovídajícího příkazu z nabídky.

Důvod je prostý: typické použití Hlásiče SMS spočívá v tom, že uživatel tuto aplikaci spustí a nechá ji běžet na pozadí. Standardně lze aktuální aplikaci přepnout na pozadí pomocí tzv. *application key* (viz podkap. 1.2.3), jenže toto tlačítko bývá často situováno na daleko méně přístupném místě, než *softkeys*, která tvoří základ ovládání telefonu.

Implementačně je to provedeno pomocí třídy `TApaTask`, která souvisí s blokem Application Architecture (viz podkap.1.1.4). Umožňuje operace s „úlohou aplikace“ (*application task*), ta je spjata s konkrétní existující skupinou oken. Jednou z těchto operací je právě i přesun související skupiny oken na pozadí.

Uživatelská volba tónu

Volbu tónu formou dialogu se seznamem použitelných zvukových souborů lze realizovat poměrně snadno za použití API platformy S60 s názvem *Media Fetch*, konkrétně pomocí třídy `MGFetch`. Uživatel však může být takto provedeným výběrem tónu zklamán, protože mezi zobrazenými soubory nejsou zahrnuty standardní tóny, které se dají u nastavení v profilu běžně zvolit — tyto soubory jsou uloženy společně se systémem přímo v ROM.

Od verze S60 3rd, FP1 se ve veřejném API objevila alternativní možnost výběru tónu, která pokrývá i ony předpřipravené zvuky v ROM. Lépe řečeno, skrze rozhraní dotyčné třídy `MProEngAlertToneSeeker` lze obdržet seznam všech dostupných souborů, které je možno potenciálně použít jako tón. Na rozdíl ale od předchozího postupu musí být grafické zobrazení s výběrem souboru zajištěno svépomocí. Až na tuto drobnou komplikaci se tato varianta jeví lépe, než ta první.

Bohužel však není dostupná pro verzi S60 3rd Edition (počáteční vydání), kterou jsem se zavázal rovněž podporovat (viz 3.1). Situaci tedy řeším tak, že pro novější vydání S60 jsou k dispozici obě možnosti volby tónu, při sestavení pro počáteční vydání S60 3rd se druhá možnost vyjme pomocí podmíněného překladu, jaký již byl použit v podkap. 2.7. Dlužno dodat, že u aplikace většího významu by možná mělo smysl vytvořit vlastní, na verzi platformy nezávislou komponentu, která dokáže prozkoumat veškeré dostupné soubory a potom předložit seznam takto nalezených tónů. Otázkou však je, zda by taková komponenta nebyla příliš limitována zmíněným bezpečnostním modelem systému (viz podkap. 1.1.1).

Aplikace a bezpečnostní model

U konceptu zabezpečeného Symbianu s označením Platform Security (viz podkap. 1.1.1), jehož jsem se dotkl v závěru předchozí sekce, se ještě nakrátko zdržím.

Problémem je, že získat zpětně informaci o tom, jaká přesně oprávnění aplikace požaduje, může být časově náročné, protože je potřeba projít dokumentaci ke všem v aplikaci použitým prostředkům API, kde tato informace bývá uvedena. Carbide.c++ sice nabízí nástroj *Capability Scanner*, ten se však při srovnání se skutečným stavem zjištěným vlastními pokusy ukázal jako nespolehlivý, vhodný pouze pro orientační odhad těchto potřebných *capabilities*.

K onomu skutečnému stavu Hlásičem SMS požadovaných oprávnění jsem se dobral metodou „pokus – omyl“, když jsem zkoušel, která z Carbidem předem deklarovaných oprávnění mohu odstranit, aniž by to ovlivnilo chování aplikace. I navzdory poměrně radikálnímu zásahu do zpracování příchozích zpráv požaduje oprávnění pouze dvě: `ReadUserData` a `NetworkServices`. Obě jsou z kategorie nejzákladnějších oprávnění typu *User Capabilities*, což znamená, že je možné na instalační soubor použít zmíněnou možnost podpisu *self-signed* (viz podkap. 1.3.1). Ta se dá nastavit přímo v prostředí Carbide.c++ v konfiguraci pro tento úkol používaného nástroje *SIS Builder*.

Lokalizace aplikace

Nakonec bych jen letmo zavadil o způsob lokalizace aplikace do různých jazyků, s níž mi zpočátku pomohl zmíněný UI Designer (viz podkap. 3.2, část „Návrh uživatelského rozhraní“). Kromě intuitivní lokalizace řetězců v rámci standardních „zdrojů“ aplikace (menu a jeho položky, sdělení, dialogy apod.) lze poměrně jednoduše lokalizovat i obecné řetězce. Slouží k tomu *resource* typu `TBUF` a na jeho dynamické načtení v závislosti na aktuálním jazyku prostředí při běhu aplikace lze použít pomocnou třídu `StringLoader`. Pokročilejšími technikami se mi dokonce podařilo dosáhnout toho, že v české lokalizaci umí Hlásič SMS používat odpovídající české tvary ve sdělení o tom, kolik přišlo nových zpráv („1 nová zpráva“, „2 nové zprávy“ ... „5 nových zpráv“), přičemž u anglické lokalizace stačí rozlišovat jen jednotné a množné číslo.

Kapitola 4

Závěr

Výstup této práce, ukázkovou aplikaci podléhající specifikovaným požadavkům, se mi podařilo popsaným způsobem dotáhnout do stavu, kdy splňuje kladené nároky, je s drobnými výjimkami plně funkční a zároveň připravená na budoucího rozšíření.

Bude doděláno (testování /telefony, emulátor/, známé chyby/nekorektní chování /SMS na displej, zprávy s „odentrováním“/, zásadní problémy během vývoje /místy mizerná dokumentace , příliš omezená API, problém s ikonou aplikace – barevné přechody, problematický emulátor/, zhodnocení projektu, budoucí rozvoj /filtrování podle textu zprávy, případně s využitím reg. výrazů, XML pro ukládání dat, autostart, lepší imitace systému ohledně vibrací, pípání apod./, návaznost na ostatní práce /?/).

Seznam použitých zdrojů

[1] Digia. *Programming for the Series 60 Platform and Symbian OS*. Chichester, Velká Británie : Wiley, 2003. Pro informace o knize viz <http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470849487.html>. ISBN 978-0-470-84948-4. (odkazuje s. 4, 6, 7, 8, 9, 10, 15, 16, 22)

[2] HAAPANEN, A. Active objects in Symbian OS. Diplomová práce, Department of Computer Sciences, University of Tampere, Tampere, Finská republika, 2008. 51 s. Dostupné též na <http://www.cs.uta.fi/research/theses/masters/Haapanen_Aapo.pdf>. (odkazuje s. 12)

[3] JAKL, A. *GUI Architectures* [online]. symbianresources.com, 2008-05-22. [cit. 2009-04-23]. PDF soubor (k dispozici také ukázková úloha včetně řešení). Dostupné z: <http://www.symbianresources.com/tutorials/gui.php#gui>. (odkazuje s. 8, 13)

[4] MORRIS, B. *The Symbian OS Architecture Sourcebook: Design and Evolution of a Mobile Phone OS*. Chichester, Velká Británie : Wiley, 2007. Pro informace o knize viz <http://developer.symbian.com/main/documentation/books/books_files/arch/index.jsp>. ISBN 978-0-470-01846-0. (odkazuje s. 2, 3, 4, 6, 7, 8, 15, 17)

[5] PAYU, S. *Silent Receiving of SMS messages* [online]. SymbianTricks.info, 2008-10-09. [cit. 2009-05-07]. Dostupné z: <http://symbiantricks.info/tricks/silent_receiving_of_sms_messages>. (odkazuje s. 16)

[6] SALES, J. *Symbian OS Internals: Real-time kernel programming*. Chichester, Velká Británie : Wiley, 2005. Pro informace o knize viz <http://developer.symbian.com/main/documentation/books/books_files/os_internals/index.jsp>. ISBN 978-0-470-02524-6. (odkazuje s. 2, 33)

[7] SALES, J. – TASKER, M. *Introducing EKA2* [online]. Wiley. [cit. 2009-04-19]. PDF soubor, ukázková kapitola z knihy *Sales, J., Tasker, M. Symbian OS Internals* ([6]), 16 s. Dostupné z: <http://media.wiley.com/product_data/excerpt/47/04700252/0470025247.pdf>. (odkazuje s. 2, 4)

[8] SHACKMAN, M. *Platform Security – a Technical Overview* [online]. Symbian, 2006-05-10. [cit. 2009-04-19]. PDF soubor. Dostupné z: <http://developer.symbian.com/main/downloads/papers/plat_sec_tech_overview/platform_security_a_technical_overview.pdf>. (odkazuje s. 3)

[9] *Gartner Says Worldwide Smartphone Sales Reached Its Lowest Growth Rate With 3.7 Per Cent Increase in Fourth Quarter of 2008* [online]. Egham, UK: Gartner, 2009-03-11.

[cit. 2009-04-17]. Dostupné z: <http://www.gartner.com/it/page.jsp?id=910112>. (odkazuje s. 1)

[10] *Psion: History & Business* [online]. Psion Investor Relations Centre (skrze Internet Archive: Wayback Machine), 2002-12-15. [cit. 2009-04-17]. Dostupné z: <http://web.archive.org/web/20021215070547/http://psion.investor-relations.co.uk/invest/psion/history.html>. (odkazuje s. 2)

[11] *C++ Developer's Library 1.4* [online]. Nokia: Forum Nokia, 2008. [cit. 2009-05-07]. Dostupné z: <http://www.forum.nokia.com/document/Cpp_Developers_Library/>. (odkazuje s. 14)

[12] *Carbide.c++* [online]. Nokia: Forum Nokia. [cit. 2009-05-10]. Dostupné z: <http://www.forum.nokia.com/Resources_and_Information/Tools/IDEs/Carbide.c++/>. (odkazuje s. 24)

[13] *Symbian C++* [online]. Forum Nokia, Developer Discussion Boards. [cit. 2009-05-07]. Dostupné z: <http://discussion.forum.nokia.com/forum/forumdisplay.php?f=6>. (odkazuje s. 16, 20)

[14] *Extensions Plug-ins for S60 3rd Edition and S60 5th Edition SDKs* [online]. Nokia: Forum Nokia. [cit. 2009-05-10]. Dostupné z: <http://www.forum.nokia.com/info/sw.nokia.com/id/48a93bd5-028a-4b3e-a0b1-148ff203b2b3/Extensions_plugin_S60_3rd_ed.html>. (odkazuje s. 16, 23)

[15] *S60 3rd Edition C++ Developer's Library v1.0: UI architectures* [online]. Nokia: Forum Nokia, 2009. [cit. 2009-04-26]. Dostupné z: <http://library.forum.nokia.com/index.jsp?topic=/S60_3rd_Edition_Cpp_Developers_Library/GUID-E50EC0B4-A434-4C30-A1A9-1A976185FF28.html>. (odkazuje s. 8, 13)

[16] *Runtimes* [online]. Nokia: Forum Nokia. [cit. 2009-04-27]. Dostupné z: <http://www.forum.nokia.com/Resources_and_Information/Tools/Runtimes/>. (odkazuje s. 11)

[17] *S60 Open to new features* [online]. Nokia: Forum Nokia. [cit. 2009-04-20]. Dostupné z: <http://www.forum.nokia.com/Resources_and_Information/Explore/Software_Platforms/S60/Open_to_New_Features.xhtml>. (odkazuje s. 7)

[18] *S60 Platform: Comparison of ANSI C++ and Symbian C++* [online]. Nokia: Forum Nokia, 2006-05-30. [cit. 2009-04-26]. PDF soubor, verze 2.0. Dostupné z: <http://www.forum.nokia.com/info/sw.nokia.com/id/36f953bf-59eb-4f41-9e60-f51f494c5d14/S60_Platform_Comparison_of_ANSI_Cpp_and_Symbian_Cpp_v2_0_en.pdf.html>. (odkazuje s. 11)

[19] *S60 Platform: FAQ* [online]. Nokia: Forum Nokia, 2008-03-28. [cit. 2009-04-22]. PDF soubor, verze 1.12. Dostupné z: <http://www.forum.nokia.com/info/sw.nokia.com/id/c5a37690-f5c0-42f5-95bc-8e31dad39040/S60_Platform_FAQ_v1_12_en.pdf.html>. (odkazuje s. 8)

[20] *S60 Platform: Introductory Guide* [online]. Nokia: Forum Nokia, 2008-01-21. [cit. 2009-04-27]. PDF soubor, verze 1.6. Dostupné z: <http://www.forum.nokia.com/info/sw.nokia.com/id/fc17242f-9bb2-4509-b12c-1e6b8206085b/S60_Platform_Introductory_Guide_v1_6_en.pdf.html>. (odkazuje s. 8)

[21] *S60 Platform SDKs for Symbian OS, for C++* [online]. Nokia: Forum Nokia, 2006-10-06. [cit. 2009-05-07]. Dostupné z: <http://www.forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html>. (odkazuje s. 14)

[22] *S60 Platform: SMS Example* [online]. Nokia: Forum Nokia, 2008-10-08. [cit. 2009-01-29]. Dostupné z: <http://www.forum.nokia.com/info/sw.nokia.com/id/04802db9-ea13-48f3-b3e2-6ccf639bcbf7/S60_Platform_SMS_Example.html>. (odkazuje s. 16)

[23] *S60 UI Style Guide* [online]. Nokia: Forum Nokia, 2007-07-06. [cit. 2009-04-23]. PDF soubor, verze 1.3. Dostupné z: <http://www.forum.nokia.com/info/sw.nokia.com/id/04c58d5a-84c3-42db-83d5-486c1cf3e6b3/S60_UI_Style_Guide_v1_3_en.pdf.html>. (odkazuje s. 8, 9, 10)

[24] *Software Platforms* [online]. Nokia: Forum Nokia. [cit. 2009-04-27]. Dostupné z: <http://www.forum.nokia.com/Resources_and_Information/Explore/Runtime_Platforms/>. (odkazuje s. 11)

[25] *Symbian OS Basics* [online]. Nokia: Forum Nokia, 2008-01-08. [cit. 2009-04-23]. PDF soubor, výuková příručka, verze 3.1. Dostupné z: <http://www.forum.nokia.com/info/sw.nokia.com/id/981e8e7b-668c-449a-b0e0-e7e5f534c39d/Symbian_OS_Basics_Workbook.html>. (odkazuje s. 7, 11)

[26] *Create Local SMS* [online]. Forum Nokia, Wiki, 2009-03-31. [cit. 2009-05-09]. Dostupné z: <http://wiki.forum.nokia.com/index.php/Create_Local_SMS>. (odkazuje s. 20)

[27] *Playing audio files* [online]. Forum Nokia, Wiki, 2009-03-13. [cit. 2009-05-10]. Dostupné z: <http://wiki.forum.nokia.com/index.php/Playing_audio_files>. (odkazuje s. 22)

[28] *SDK API Plug-in* [online]. Forum Nokia, Wiki, 2009-05-04. [cit. 2009-05-07]. Dostupné z: <http://wiki.forum.nokia.com/index.php/SDK_API_Plug-in>. (odkazuje s. 16)

[29] *Mobile leaders to unify the Symbian software platform and set the future of mobile free* [online]. Nokia, 2008-06-24. [cit. 2009-04-20]. Tisková zpráva. Dostupné z: <http://www.nokia.com/A4136001?newsid=1230416>. (odkazuje s. 7)

[30] *Nokia acquires Symbian Limited* [online]. Nokia, 2008-12-02. [cit. 2009-04-19]. Tisková zpráva. Dostupné z: <http://www.nokia.com/A4136002?newsid=1274570>. (odkazuje s. 3)

[31] *Series 60, the world's leading smartphone platform, redefines its name and visual identity* [online]. Nokia, 2005-11-01. [cit. 2009-04-20]. Tisková zpráva. Dostupné z: <http://www.nokia.com/A4136002?newsid=1019076>. (odkazuje s. 8)

[32] *Symbian Fast Facts Q2 2008* [online]. Symbian. [cit. 2009-04-22]. Dostupné z: <http://www.symbian.com/about/fast.asp>. (odkazuje s. 2)

[33] *About the Symbian Foundation* [online]. Symbian Foundation. [cit. 2009-04-20]. Dostupné z: <http://www.symbian.org/about.php>. (odkazuje s. 7)

[34] *An interactive history of Symbian* [online]. Symbian. [cit. 2009-04-22]. Dostupné z: <http://www.symbian.com/about/timeline.asp>. (odkazuje s. 2)

[35] *Symbian Developer Library: Symbian OS v9.2* [online]. Symbian. [cit. 2009-05-07]. Dostupné z: <http://developer.symbian.com/main/documentation/sdl/symbian92/>. (odkazuje s. 14)

[36] *Symbian Developer Library: Symbian OS v9.2 — Contacts Model Overview* [online]. Symbian. [cit. 2009-05-10]. Navigace: Symbian OS v9.2 » Symbian OS guide » Application engines » Using Contacts Model (CNTMODEL) » Contacts Model overview. Dostupné z: <http://developer.symbian.com/main/documentation/sdl/symbian92/>. (odkazuje s. 21)

[37] *Symbian Developer Library: Symbian OS v9.2 — Introduction to dictionary stores* [online]. Symbian. [cit. 2009-05-12]. Navigace: Symbian OS v9.2 » Symbian OS guide » System libraries » Using Store » Stores » Dictionary stores » Introduction to dictionary stores. Dostupné z: <http://developer.symbian.com/main/documentation/sdl/symbian92/>. (odkazuje s. 29)

[38] *Symbian Developer Library: Symbian OS v9.2 — Multi Media Framework Client Overview* [online]. Symbian. [cit. 2009-05-07]. Navigace: Symbian OS v9.2 » Symbian OS guide » Multimedia » Multi Media Framework — Client API » Multi Media Framework Client Overview. Dostupné z: <http://developer.symbian.com/main/documentation/sdl/symbian92/>. (odkazuje s. 22)

[39] *Symbian Developer Library: Symbian OS v9.2 — Sockets Client Overview* [online]. Symbian. [cit. 2009-05-07]. Navigace: Symbian OS v9.2 » Symbian OS guide » Comms infrastructure » Using Sockets Server (ESOCK) » Sockets Client Overview. Dostupné z: <http://developer.symbian.com/main/documentation/sdl/symbian92/>. (odkazuje s. 16)

[40] *Symbian Developer Library: Symbian OS v9.2 — TCP/IP client programs* [online]. Symbian. [cit. 2009-05-08]. Navigace: Symbian OS v9.2 » Symbian OS guide » Networking » Using TCP/IP (INSOCK) » Introduction to TCP/IP » TCP/IP client programs. Dostupné z: <http://developer.symbian.com/main/documentation/sdl/symbian92/>. (odkazuje s. 16, 18)

[41] *Symbian Developer Library: Symbian OS v9.2 — The Message Server: Structure* [online]. Symbian. [cit. 2009-05-09]. Navigace: Symbian OS v9.2 » Symbian OS guide » Messaging » Using Messaging Framework » The Message Server » Structure. Dostupné z: <http://developer.symbian.com/main/documentation/sdl/symbian92/>. (odkazuje s. 15)

[42] *Symbian Developer Library: Symbian OS v9.2 — UI Control Framework Overview* [online]. Symbian. [cit. 2009-05-07]. Navigace: Symbian OS v9.2 » Symbian OS guide » Application framework » The UI Control Framework (CONE) » UI Control Framework overview. Dostupné z: <http://developer.symbian.com/main/documentation/sdl/symbian92/>. (odkazuje s. 6)

[43] *Symbian Signed* [online]. Symbian. [cit. 2009-01-29]. Dostupné z: <https://www.symbiansigned.com/app/page>. (odkazuje s. 3)

[44] *Symbian OS v9.3: System Model* [online]. Symbian, 2008-03-07. [cit. 2009-04-19]. PDF soubor. Dostupné z: <http://developer.symbian.com/main/downloads/files/publicD-2.1.pdf>. (odkazuje s. 4)

[45] *EKA2* [online]. Wikipedia, the free encyclopedia, 2008-12-05. [cit. 2009-01-26]. Dostupné z: <http://en.wikipedia.org/wiki/EKA2>. (odkazuje s. 2)

[46] *Kernel-wide design approaches* [online]. Wikipedia, the free encyclopedia, 2009-03-14. [cit. 2009-04-19]. Dostupné z: <http://en.wikipedia.org/wiki/Kernel_(computer_science)#Kernel-wide_design_approaches>. (odkazuje s. 4)

[47] *S60 (software platform)* [online]. Wikipedia, the free encyclopedia, 2009-01-24. [cit. 2009-01-27]. Dostupné z: <http://en.wikipedia.org/wiki/S60_(software_platform)>. (odkazuje s. 7)

[48] *Symbian OS* [online]. Wikipedia, the free encyclopedia, 2009-01-26. [cit. 2009-01-26]. Dostupné z: <http://en.wikipedia.org/wiki/Symbian_OS>. (odkazuje s. 2)

Rejstřík

active objects, *viz* aktivní objekty
aktivní objekty, 13
API SMS Utilities, 17
ARM, 3, 4

capabilities, 4

deskriptory, 13

EKA
    EKA1, 5
    EKA2, 5
engine, 5
EPOC, 3
    EPOC16, 3
    EPOC32, 3

IPC, *inter-process communication*, 5, 13

Message Server, 16
model
    klient–server, 4, 13, 17
    Symbianu, *viz* Symbian, model systému
MTM, *message-type modules*, 16

návrhový vzor
    MVC, *model–view–controller*, 4
    observer, 4

operační systém reálného času, *viz* RTOS

Platform Security, 4
platforma S60, 9–12
    historie, 9
    prvky uživatelského rozhraní, 11
    vlastnosti, 9–11

RTOS, *real-time operating system*, 5

S60, *viz* platforma S60
sockety, 17–20
    socket server, 17
Symbian
    historie, 3–4
    jádro, 5
    model systému, 5–7
    platformy, 8
    vlastnosti, 4
    vznik, 3
Symbian Signed, 4

Seznam příloh

A. Uživatelská příručka — součástí tohoto textu, od s. 40

B. Nosič CD, který obsahuje:
   - soubor `README_CS.TXT`/`README_EN.TXT`:
     - základní informace o této práci a přehled toho, co toto médium obsahuje (podobný tomuto)
   - soubor `PREREQ_CS.TXT`/`PREREQ_EN.TXT`:
     - informace o softwarovém vybavení pro sestavení aplikace včetně odkazů na zdroje, kde je možné tyto konkrétní prostředky získat (IDE Carbide.c++ 2.0, C++ SDK pro S60 3rd Edition, počáteční vydání a Feature Pack 1)
   - soubor `BUILD_CS.TXT`/`BUILD_EN.TXT`:
     - postup sestavení aplikace se zmíněným softwarovým vybavením (soubor `PREREQ_CS.TXT`/`PREREQ_EN.TXT`)
   - adresář `application`:
     - adresářová struktura projektu obsahující kompletní zdrojové texty výsledné aplikace, včetně dalších souborů nutných pro její sestavení a vytvoření instalačního souboru (mj. definiční soubor projektu, předpisy pro sestavení aplikace a vytvoření instalačního souboru, soubory definující použité „zdroje“ jako menu či dialog, grafická ikona aplikace, soubory s lokalizací aplikace do českého a anglického jazyka);
     - žádné SDK, IDE ani balíky doplňkových API do SDK toto médium neobsahuje, získat je lze z umístění uvedených v souboru `PREREQ_CS.TXT`/`PREREQ_EN.TXT`
     - ve standardním umístění, v podadresáři `sis`, je také připravena hotová aplikace v podobě instalačních souborů:

`SmsAlerts_3rd.sis`	pro 3rd Ed., nepodepsané
`SmsAlerts_3rd.sisx`	pro 3rd Ed., self-signed
`SmsAlerts_3rd_FP1+.sis`	pro 3rd Ed. FP1 a vyšší, nepodepsané
`SmsAlerts_3rd_FP1+.sisx`	pro 3rd Ed. FP1 a vyšší, self-signed

   - adresář `doc`:
     - dokumentace zdrojového kódu automaticky vygenerovaná nástrojem *Doxygen*[1]
   - adresář `text`:
     - text této práce ve formě PDF; v podadresáři `src` jsou pak její zdrojové soubory pro systém LaTeX[2] stejně jako použité obrázky a schémata

[1] viz `<http://www.stack.nl/~dimitri/doxygen/>`
[2] viz `<http://www.latex-project.org/>`

Příloha A

Uživatelská příručka

Bude doděláno.