VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INTELIGENTNÍ SPRÁVA VYZVÁNĚCÍCH PROFILŮ V OS WINDOWS MOBILE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE VLADIMÍR KLEŠTINEC
AUTHOR

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INTELIGENTNÍ SPRÁVA VYZVÁNĚCÍCH PROFILŮ V OS WINDOWS MOBILE

INTELIGENT CONTROL OF RINGING PROFILES IN OS WINDOWS MOBILE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — VLADIMÍR KLEŠTINEC
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARTIN ČERMÁK,
SUPERVISOR

BRNO 2009

Abstrakt

Práce se zabývá vytvořením inteligentí správy vyzváněcích profilů pro platformu Microsoft Windows Mobile. Důležitou částí práce je komunikace s Radio Interface Layer a kalendářem. Celá aplikace má přívětivé a rychlé rozhraní nezatěžující uživatele.

Abstract

The work is dealing with creation of Inteligent Control of Ringing Profiles on platform Microsoft Windows Mobile. The important part of this work is communication with Radio interface Layer and calendar. Whole application has user friendly and fast graphical user interface.

Klíčová slova

Windows Mobile, vyzváněcí profily, C#/.NET Compact Framework

Keywords

Windows Mobile, ring profiles, C#/.NET Compact Framework

Citace

Vladimír Kleštinec: Inteligentní správa vyzváněcích profilů v OS Windows Mobile, bakalářská práce, Brno, FIT VUT v Brně, 2009

Inteligentní správa vyzváněcích profilů v OS Windows Mobile

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Martina Čermáka

. . . . . .
Vladimír Kleštinec
18. května 2009

Poděkování

Chtěl bych svému vedoucímu Ing. Martinu Čermákovi, za jeho odbornou pomoc při vytváření této práce.

© Vladimír Kleštinec, 2009.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **2**

2 Základní Teorie a Pojmy **3**
2.1 Přehled operačních systémů, pro mobilní zařízení . . . 3
2.1.1 Google Android . . . 3
2.1.2 Symbian OS . . . 4
2.1.3 iPhone OS . . . 5
2.1.4 Windows Mobile . . . 6
2.1.5 Shrnutí . . . 7
2.2 Významné části Windows Mobile . . . 7
2.2.1 Radio Interface Layer . . . 8
2.2.2 Registry . . . 9
2.2.3 Pocket Outlook Object Model . . . 11
2.3 XML . . . 11
2.4 Databáze . . . 13

3 Specifikace **14**

4 Analýza a návrh **15**
4.1 Změna vyzvánění . . . 15
4.1.1 Notifikace . . . 15
4.1.2 Zvuky . . . 16
4.2 Uložení dat . . . 17
4.3 Získání ID vysílače . . . 19
4.4 Přístup do kalendáře . . . 21
4.5 Programovací jazyk . . . 22

5 Implementace **23**
5.1 ChrisTec Managed Today Screen Item Framework . . . 23
5.2 Rozhraní aplikace . . . 23
5.3 Aplikační části . . . 25
5.3.1 Konfigurace . . . 25
5.3.2 Vyzváněcí profil . . . 25
5.3.3 Monitorování potřebných událostí . . . 27
5.3.4 Udržení notifikací ve funkčním stavu . . . 28

6 Závěr **30**

Kapitola 1

Úvod

Dnešní svět se dynamicky rozvíjí. Jedním z nejrychleji rozvíjejícím se odvětvím jsou mobilní zařízení. Výkonná zařízení se stala cenově dostupná pro značnou část populace. Vhodně zvolené aplikace pro denní využití mohou zařízením dodat značnou popularitu.

Cílem mé bakalářské práce bylo napsat aplikaci pro správu vyzváněcích profilů, která by nepostrádala klasickou funkčnost a zároveň nabízela netradiční možnosti. Mezi tato netradiční rozšíření patří například přepínání profilu na základě cell id přihlášeného vysílače. Aplikace by měla být maximálně uživatelsky přívětivá při zachování funkčnosti. Cílová platforma je Windows Mobile 6 Professional.

Na začátku práce je čtenář seznámen s nejvýznamnějšími operačními systému na trhu, s jejich historií a možnostmi vývoje pro danou platformu. V této úvodní kapitole jsou také rozebírány technologie použité pro vývoj naší aplikace, jako je například XML, Radio Interface Layer a Pocket Outlook Object Model. Kapitola „Specifikace“ obsahuje základní specifikaci aplikace, to znamená základní představu o funkčnosti, kterou budeme s naší aplikací poskytovat. Následující kapitola rozebírá analýzu a návrh. Je zde diskutována realizace změny vyzvánění za pomoci úprav registrových hodnot. Dále pak technické řešení uložení dat do některého z navrhovaných úložišť. Nebyla opomenuta ani problematika zjištění identifikačního čísla vysílače, stejně jako přístup do kalendáře. Nakonec této kapitoly spadá volba implementačního jazyka na základě předchozích analýz. V poslední kapitole je popsána implementace daných komponent, použité frameworky a je zde ukázáno rozhraní naší aplikace. Závěr obsahuje zhodnocení celé práce a jejího přínosu.

Kapitola 2

Základní Teorie a Pojmy

2.1 Přehled operačních systémů, pro mobilní zařízení

Segment trhu mobilních zařízení je v současné době jeden z nejrychleji expandujících. Pro tato zařízení existuje mnoho operačních systémů. Tyto operační systémy mají rozdílné architektury i uplatnění. Liší se také podporou a dostupností vývojářských nástrojů. Každý z nich podporuje jiné programovací jazyky a má jinak rozsáhlé API. Jsou zde zastoupeny systémy od nejmladších až po relativně staré hráče. V následujícím přehledu popíši ty nejrozšířenější.

2.1.1 Google Android

Samotný název Google android je souhrnné označení pro operační systém a soubor klíčových aplikací pro tuto platformu. Je nejmladší ze zde prezentovaných operačních systémů - uvedení na trh nastalo teprve v roce 2007. Vývoj nativních aplikací není zatím oficiálně podporován. Programovat se dá pouze v jazyce Java, pro který společnost Google vytvořila vlastní knihovny. Samozřejmě je to interpretovaný jazyk. Nejedná se ovšem o plnou implementaci jazyka Java, protože společnost Google neimplementovala všechny standardní knihovny, za což je kritizována. Oficiálně podporované vývojové prostředí je Eclipse (3.2 a novější). Více informací můžete nalézt v [1]. Základní úvod do architektury je uveden zde [2].

Základní architektura sytému je zobrazena na obrázku 2.1. Na tomto obrázku můžeme vidět grafické znázornění do pěti logických úrovní, které nyní popíši.

Applications - základní aplikace dodávané společně s OS. Jedná se například o emailový klient, kalendář, program na zasílání SMS a internetový prohlížeč.

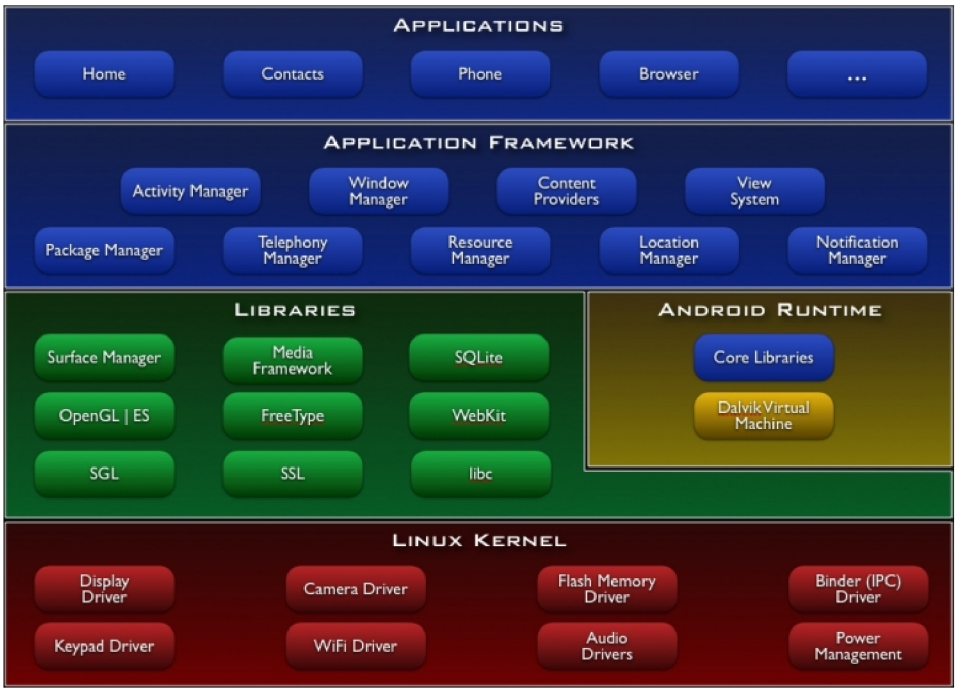
Application framework - poskytuje základní služby systému:

- Content provider, který dovoluje aplikacím jednoduše sdílet data mezi sebou. Zajišťuje jednoduchý přístup ke kontaktům.
- Resource Manager zajišťuje přístup k neprogramovým zdrojům jako lokalizovaným řetězcům a grafice.
- Notification manager umožňuje všem aplikacím zobrazit vlastní výstrahu v status baru.

Libraries - souhrn C/C++ knihoven používaných různými komponentami systému. Přistupuje se k nim pomocí aplikačního frameworku. Jedná se například o SQLite knihovny sloužící pro efektivní uložení dat databáze, 3D knihovny jako OpenGL ES pro zobrazování 3D grafiky.

Android Runtime - set knihoven jádra operačního systému a Dalvik Virtual Machine (DVM). Jedná se o implementaci virtuálního stroje jazyka Java. Každá aplikace má svůj vlastní proces a svůj vlastní DVM. Byl napsán tak, že zařízení může mít spuštěno několik DVM současně.

Linux kernel - Google Android používá Linuxové jádro verze 2.6, které vytváří abstraktní vrstvu nad hardwarem telefonu.



Obrázek 2.1: Architektura Google Android, obrázek byl převzat z [2].

2.1.2 Symbian OS

Tento velmi rozšířený operační systém vznikl v roce 1998 na základě spolupráce firem Ericsson, Nokia, Motorola a Psion. Během roku 2009 by se měl Symbian pod záštitou Symbian Foundation stát open source. Nejnovější stabilní verze systému je v. 9.5 s grafickým uživatelským rozhraním UIQ nebo S60. UIQ je vyvíjeno firmou Sony Ericsson a S60 firmou Nokia. S60 je určena pro telefony bez dotykového displaye, na rozdíl od UIQ, které jej podporuje. Mnoho společností dodává velmi užitečné knihovny pro Symbian, bohužel je řada z nich placená. Symbian podporuje velké množství programovacích jazyků. Hlavními jsou Java, Python, Open C/C++ a Symbian C++. Open C++ je implementace standardního C++ známého z desktopu. Oproti tomu Symbian C++ je lehce výkonnější a dovoluje využít celý potenciál zařízení. Všechny vývojářské nástroje jsou k dispozici zdarma. Standardní

SDK je postaven na IDE Eclipse, instalace přidá automaticky plugin do vývojového prostředí. SDK je dostupné v různých verzích podle platformy, pro kterou je aplikace určena. Samozřejmostí je dodávaný emulátor, ve kterém lze jednoduše otestovat aplikaci.

Základní architektura systému je složena ze čtyř vrstev, které teď popíši:

Application Services Layer je nejvyšší vrstva systému, je zde PIM Applications Services zprostředkující přístup ke kalendáři a kontaktům. Obsahuje také podporu zpráv, vzdáleného managementu a virtuální stroj Java J2ME

Generic Middleware zajišťuje například vykreslování textu, práci s http, telnet a ftp protokolem.

OS Services Layer se dělí do několika základních skupin:

- Base services - obsahují standardní knihovnu jazyka C, dále pak knihovny pro podporu komprese, kryptografie a XML parser pro práci s XML.
- Communications services - v této skupině je mnoho služeb, zajímavé jsou například SMS stack, DHCP klient a knihovna pro podporu QoS.
- Multimedia and graphics services - zde přítomné knihovny usnadňují práci se zvukem, případně knihovny umožňují jednoduchou manipulaci s fotoaparátem
- Location based services - zaměřují se hlavně na podporu GPS modulů

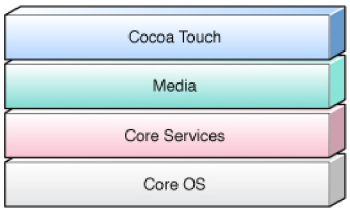
Kernel Services a Hardware Interface Layer - tyto vrstvy jsou nedosažitelné z pohledu uživatele. Jsou zde například ovladače file systému, USB, zvuku. HAL zajišťující abstrakci hardwaru.

Více informací o možnostech tohoto operačního systému je možno nalézt v [4].

2.1.3 iPhone OS

Hlavní výhoda iPhone OS je způsob interakce s uživatelem. Podporuje totiž multi-touch screen. To znamená, že systém detekuje stisknutí několika bodů na dotykovém display zároveň. Například Windows Mobile bude tuto funkci podporovat až v připravované verzi 7. Vývojářské nástroje s názvem Xcode tools jsou volně ke stažení, ovšem pouze pro Mac OS X ve verzi Leopard. Xcode tools obsahují emulátor a všechny nástroje potřebné pro základní editaci, kompilaci a testování kódu. Aplikace mohou být psány nativně v jazyce C - na nejnižších úrovních OS a v jazyku Objective-C na vyšších úrovních.

Architektura systému má čtyři základní vrstvy, které jsou znázorněny na obrázku 2.2. Pro bližší informace o systému, kompletní referenci a SDK je možno se podívat do literatury [3].



Obrázek 2.2: Architektura iPhone OS, obrázek přejat z [3]

Nyní následuje bližší popis jednotlivých vrstev.

Core OS je na nejnižším stupni této architektury. Je zde jádro operačního systému, síťová infrastruktura a správa souborového systému.

Core Services zahrnují frameworky poskytující služby pro manipulaci s řetězci, správu kolekcí, URL utility, management uživatelských kontaktů a nastavení.

Na vyšších vrstvách se mixují technologie založené na C a ObjC.

Media je vrstva, jejíž služby a frameworky jsou závislé na předchozí vrstvě Core services, poskytují grafické a multimediální služby Cocoa Touch vrstvě. Tyto služby umožňují 2D a 3D kreslení, Core Audio, Video přehrávání, OpenGL ES.

Cocoa Touch vrstva přímo podporuje aplikace pro iPhone OS. Nejdůležitější frameworky jsou:

- UIKit framework, poskytující objekty, které aplikace zobrazuje v rámci uživatelského rozhraní.
- Foundation framework definuje základní chování objektů, zavádí mechanismy pro jejich správu a poskytuje primitivní datové typy, kolekce a služby operačního systému.

2.1.4 Windows Mobile

Vývoj tohoto uzavřeného operačního systému sahá do roku 2000, kdy byl vydán Pocket PC 2000. Windows mobile jsou založeny na Windows CE (Windows Embedded Compact), která jsou určena do vestavěných (embedded) zařízení. Windows mobile je momentálně ve verzi 6.1, která oproti verzi 6 nepřináší žádné výrazné změny, pouze nový design hlavní obrazovky a zlepšení výkonu. Vývojářské nástroje jsou k dispozici zdarma, avšak pouze pro platformu Microsoft Windows. Zaintegrují se do instalace Visual Studia (minimální verze 2005 SP1). Vývojář pak může aplikace vyvíjet stejně jak je zvyklý pro desktop. Součástí SDK je samozřejmě plnohodnotný emulátor. Nástroje jsou v různých verzích, v závislosti na cílové platformě. Mimo současné SDK je taky balíček takzvaných „Remote Tools“,které jsou neocenitelnou pomůckou pro nahlížení do útrob, ať už reálného zařízení nebo emulátoru. Je zde Windows CE Remote File Viewer, který dovolí procházet soubory obsažené v zařízení, dále Windows CE Remote Process Viewer, díky kterému máme možnost sledovat běžící procesy a ukončovat je. Poslední je Windows CE Remote Registry Editor, poskytující plnou kontrolu nad registry připojeného zařízení.

V současnosti jsou na trhu tři produktové řady:

Classic - určena pro zařízení bez telefonního modulu

Standard - podporuje zařízení s telefonním modulem, ale bez dotykového displaye

Professional - určena pro nejvýkonnější zařízení s dotykovým displayem a telefonním modulem

API je uspořádáno do sekcí podle účelu použití, jsou jimi například: Application and Services, zde můžeme najít nástroje, které umožňují jednoduchou synchronizaci s počítačem pomocí ActiveSync. Dále je tu Pocket Outlook Object Model (POOM) poskytující přístup ke kontaktům, událostem a schůzkám v kalendáři. V sekci Audio se nachází jednoduché API Wave Form umožňující práci se zvukem. Sekce Phone Features, která umožňuje zpřístupňuje správu SMS zpráv a volání. Obsahuje SIM management, což je API poskytující přistup k SIM kartě. Je zde také Radio Interface Layer (RIL), který umožňuje získat informace z vysílačů BTS. Pro správu napájení je zde API Power Management. V části Security se nacházejí knihovny pro efektivní správu certifikátů, vytváření hash a autentizaci.

Aplikace je možno vyvíjet nativně v jazyce C++, za podpory knihoven MFC a ATL. Tyto knihovny jsou pro programátory známé z desktopových systémů, takže není nutné se s nimi složitě učit zacházet. Windows mobile umožňují vývoj interpretovaných aplikací v jazyce C# na platformě .NET Compact Framework. Jedná se o podmnožinu klasického .NET Frameworku známého z PC. Tento framework je ve verzi 2.0 (nejnovější verze 3.5) dostupný standardně ve všech zařízeních s operačním systémem Windows Mobile 6. Technologie .NET je opět dobře známa všem desktopovým programátorům. Vývoj aplikací na platformě .NET je rychlý a efektivní. Je vhodný také z důvodu různých velikostí a rozlišení display u mobilních zařízení. Na různých zařízeních by aplikace vypadala různě, .NET Compact Framework však automaticky mění měřítko ovládacích prvků a tím šetří programátorovi práci. Každé WM 6 zařízení obsahuje v ROM krome .NET Compact Frameworku 2.0 SP2 také SQL Server 2005 Compact Edition. Tento SQL server je optimalizován pro přenosná zařízení a umožňuje jednoduchou synchronizaci s klasickými MS SQL servery.

Kompletní reference včetně podrobnějších informací je volně dostupná na webových stránkách [5]. Výborně popsané vývojové prostředí a způsoby programování v .NET Compact Frameworku lze nalézt v knize od Luboslava Lacka [7].

2.1.5 Shrnutí

Operačních systémů do mobilních zařízení existuje velké množství. Liší se od sebe jak podporou programovacích jazyků, kvalitou dokumentace, velikostí uživatelské základny, tak i obtížností programování. Každý operační systém má svá pro a proti. iPhone OS vyniká uživatelsky přívětivým rozhraním, na druhou stranu však programátora nutí pracovat v nepříliš rozšířeném Objective C. Google Android nedává programátorovi na výběr vůbec a nutí jej programovat v jazyce Java. Nejsnadnější vývoj je na platformě Windows Mobile, protože umožňuje velmi rychlý přechod pro desktopového programátora. Umožňuje mu psát programy tak, jak je zvyklý z klasických PC aplikací, ať už v C++/MFC nebo C#/.NET.

2.2 Významné části Windows Mobile

Jelikož je má práce zaměřena na operační systém Windows Mobile, následuje nyní popis důležitých částí tohoto OS z hlediska mé aplikace.

2.2.1 Radio Interface Layer

Radio interface Layer (RIL) je pro nás velmi důležitá část komunikačního sub-systému. Poskytuje rozhraní, které zajišťuje komunikaci mezi software ovládajícím jádro telefonu a hardwarem sloužícím pro rádiovou komunikaci. Tato vrstva rádiového rozhraní umožňuje výrobcům vytvořit ovladač, který může být použit pro různá rádiová zařízení. RIL odstiňuje detaily rádiového hardwaru, díky tomu mohou výrobci jednoduše zařadit širokou paletu modemů různých typů a značek. Tímto se může produkt jednoduše odlišit od ostatních. Znamená to, že jeden ovladač umožňuje všem rádiovým zařízením pracovat pod jednou množinou CellCore funkcí. RIL se skládá ze dvou částí

- RIL proxy
- RIL driver

RIL proxy je Windows Embedded CE dynamická knihovna, která obsluhuje callback notifikace a vnitřní volání funkcí procesu do RIL ovladače. Rozhraní RIL (API) neinteraguje s ovladačem RIL, ale s RIL proxy.

RIL proxy je rozhraní poskytované firmou Microsoft, jedná se o abstraktní vrstvu odstiňující daný radio hardware. Zajišťuje jednotné funkce pro rozdílné radio moduly.

RIL driver je dodáván firmami vyvíjejícími daný rádiový hardware. Ukázkové implementace pro GSM moduly (Global System for Mobile communication) jsou k dispozici na webových stránkách firmy Microsoft. Tento ovladač je implementován jako Windows Embedded CE ovladač zařízení, který je ovládán pomocí programu Device Manager (device.exe). Ovladač poskytuje funkcionalitu zahrnující přenos dat, hlasu, krátkých textových zpráv (SMS). Poskytuje také informace o stavu komunikace (síla signálu, pokrytí sítě, příchozí hovory).

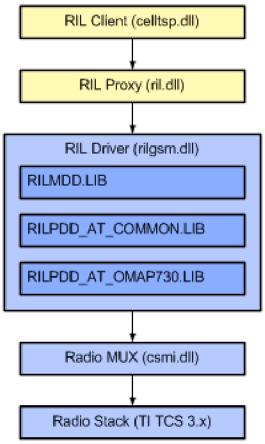
Další z funkcí RIL proxy je plnění role arbitra. Je totiž nutno zajistit výlučný přístup několika klientů k jedinému RIL ovladači.

Windows Embedded CE : Architektura ovladače RIL je založena na Windows Embedded CE standardu MDD/PDD. To znamená, že je ovladač rozdělen do dvou hlavních sekcí. Na obrázku 2.3 je vidět celá architektura RIL.

Windows Mobile : Když se daný modul poprvé zaregistruje s rozhraním RIL, předá 2 call back funkce. První je určena pro odpovědi na funkční volání. Druhá obsluhuje nevyžádané notifikace. Na obrázku 2.4, je znázorněna architektura RIL na Windows Mobile.

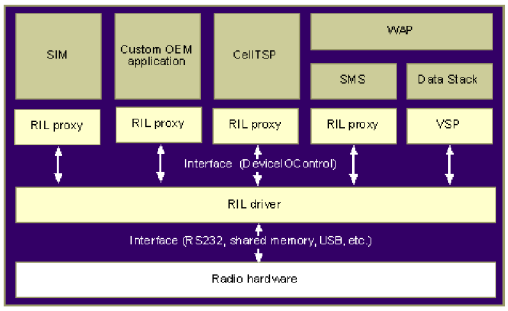
`Příklady`

- Při příchozím hovoru RIL využije callback funkci pro nevyžádané notifikace a upozorní pomocí ní každý modul na příchozí hovor.
- Pro zjištění jména operátora, síly signálu atd. se používají funkce z RIL API. Tyto funkce jsou volány asynchronně. To znamená, že funkce po zavolání ihned skončí a vrátí identifikátor odpovědi. RIL použije pro sdělení odpovědi callback funkci pro odpověď na funkční volání. Abychom mohli přiřadit odpověď k správnému funkčnímu volání, je callback funkci předán identifikátor odpovědi shodný



Obrázek 2.3: Architektura RIL na Windows CE, obrázek byl přejat z [8]

s identifikátorem získaným původním voláním. Tato asynchronní architektura zjednodušuje práci s RIL rozhraním. Pokud chceme používat RIL funkce synchronním způsobem, musíme po funkčním volání blokovat průběh aplikace, dokud není zavolána callback funkce.



Obrázek 2.4: Architektura RIL na Windows Mobile, obrázek byl přejat z [8]

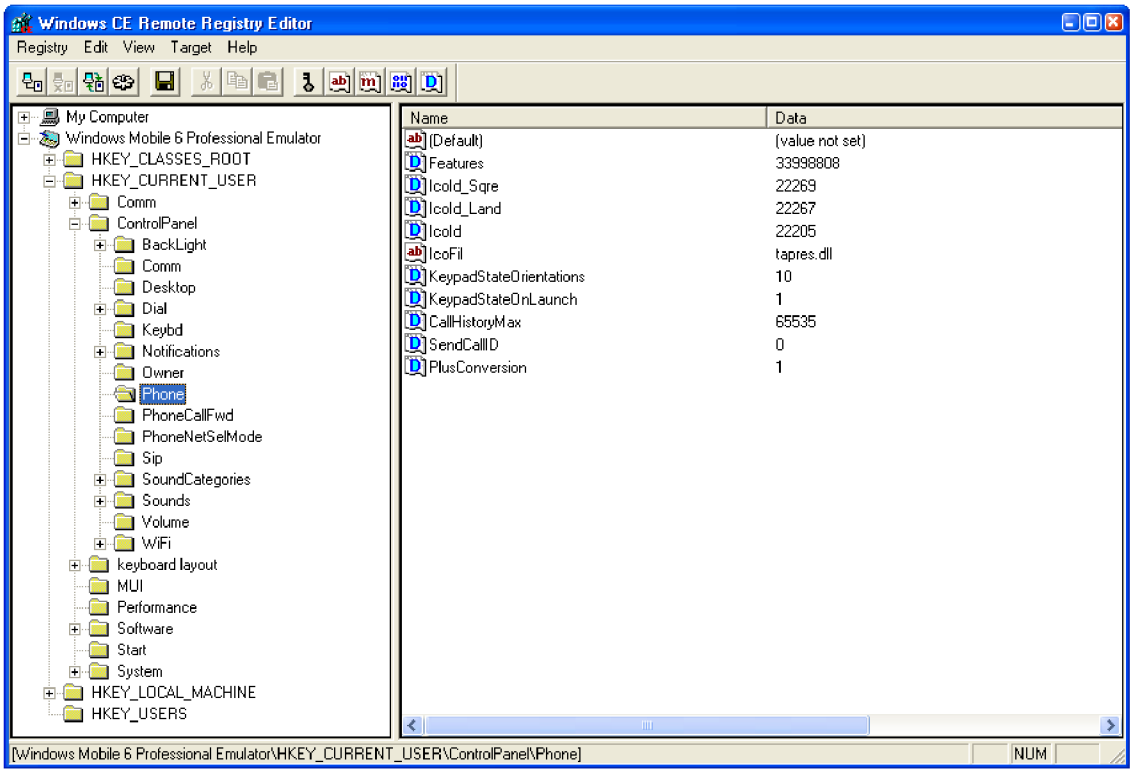
2.2.2 Registry

Registry systému Microsoft Windows jsou systémem definovaná databáze. Komponenty operačního systému i aplikace do nich mohou ukládat potřebná data a následně je číst. Tímto registry umožňují efektivní sdílení informací. Nejsou ovšem dobře přenositelné, protože struktura registrů se liší v závislosti na verzi Windows. Žádné dvoje registry nejsou shodné, protože obsahují data o použitém hardware a software, který je nainstalovaný a k dispozici na dané stanici. To znamená, že data jsou závislá na konfiguraci daného počítače (ať už hardwarové nebo softwarové). Navíc obsah některých registrových klíčů je vytvořen znovu, při každém startu stanice, případně při přihlášení uživatele. Vzhledem k tomu je mnoho změn v registrech po restartu stanice, případném odhlášení uživatele, ztraceno.

Tato databáze má hierarchickou strukturu a obsahuje data, která jsou kritická pro běh operačního systému, aplikací a služeb které běží na systému Windows. Data jsou uspořádána do stromu. Každý uzel stromu se nazývá klíč. Každý klíč může obsahovat jak podklíče, tak

datové záznamy nazývané hodnoty. Klíč může mít libovolný počet hodnot, které mohou obsahovat libovolná data. Každý klíč má své jméno složené z jednoho nebo více tisknutelných znaků. Jména klíčů nejsou „case-sensitive“, čili nezáleží na velikosti jednotlivých znaků. Jména klíčů a podklíčů ovšem nemohou obsahovat zpětné lomítko (\). Jména hodnot a jejich data toto omezení nemají. Každý klíč je v dané větvi unikátní. Jména klíčů nejsou lokalizována, ale jména hodnot mohou být.

Registry mobilních operačních systémů se od registrů desktopových operačních systémů příliš neliší. Určité změny zde samozřejmě jsou. V registrech na pracovních stanicích například chybí klíče ve větvi HKEY_CURRENT_USER jako „Control Panel“, „Telephony“, „Comm“ a další. Jak je vidět na obrázku 2.5, registry systému Windows Mobile mají 4 hlavní větve. Jsou jimi HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE a poslední HKEY_USERS. Popis dat, která obsahují se nachazí v tabulce 2.1.



Obrázek 2.5: Ukázka registrů systému Windows Mobile 6 Professional

Na desktopových operačních systémech může mít strom až 512 zanoření, přičemž jediným zavoláním API funkcí obsluhujících registry je možno vytvořit až 32 zanořených klíčů. mobilní operační systémy, musí být úspornější, zde je maximální hloubka zanoření 16. Přehled možných datových typů je uveden v tabulce 2.2.

Registry mají také určité limity pro ukládání dat. Tyto limity jsou znázorněny tabulkou 2.3. Nedoporučuje se ukládat do registrů příliš velká data, ale pouze informace typu rozměry formulářů, jejich stav (minimalizovaný, maximalizovaný). V případě potřeby uložení větších dat je vhodné uložit do registrů pouze cestu k souboru.

Hlavní větev	Popis
HKEY_CLASSES_ROOT	Obsahuje popis datových typů. Například informace o tom, že soubory s koncovkou *.png jsou obrázky.
HKEY_CURRENT_USER	Obsahuje data specifická pro uživatele, který je přihlášen. Odkazuje na relevatní klíč v HKEY_USERS.
HKEY_LOCAL_MACHINE	Obsahuje data specifická pro dané zařízení. Například pro ovladače a aplikace.
HKEY_USERS	Obsahuje data všech uživatelů, včetně implicitního (default) uživatele.

Tabulka 2.1: Popis hlavních větví registru, inspirováno z [9]

2.2.3 Pocket Outlook Object Model

Pocket Outlook Object Model (POOM) je mobilní ekvivalent Microsoft Office Outlook Object Model. Jeho funkcionalita je samozřejmě přizpůsobena možnostem mobilních zařízení. Jedná se o knihovnu založenou na technologii COM, která poskytuje programový přístup do Outlook Mobile Personal Information Management (PIM). Jedná se o objektově orientované API, které umožňuje vytvářet, upravovat, mazat a zobrazovat schůzky, úkoly a položky kontaktního listu. Toto API je dostupné jak v nativní podobě pro jazyk C++, tak v managed pro jazyk C#. V obou verzích poskytuje rozsáhlé možnosti monitorování objektů. Dá se sledovat ať už vytvoření nového objektu, odstranění objektu, až po sledování změn atributů specifického objektu. Tím se dostává programátorům do rukou mocný nástroj pro správu kalendáře, schůzek a kontaktního listu. V managed verzi je toto API pro vývojáře přívětivější, má však jednu zásadní nevýhodu oproti nativní verzi. Pokud chceme v nativní verzi sledovat například vytvoření nové události v kalendáři, vše funguje jak bychom očekávali. Po vytvoření nového objektu se zašle aplikaci příslušná zpráva - a to i v případě že je objekt vytvořen externí aplikací. V managed verzi se však toto nestane, pokud si zaregistrujeme danou událost signalizující vytvoření nového objektu, nic se nestane. Managed API totiž nereaguje na objekty vytvořené externě (například aplikací kalendář), ale pouze na objekty vytvořené v rámci této aplikace. Toto omezení je velice nepříjemné, ale dá se obejít pomocí navázání na nativní API.

2.3 XML

Aplikace dokáží v dnešní době pracovat s elektronickými daty, představujícími například hudbu a obrázky. Stále je zde však značná potřeba ukládání dokumentů založených na textových datech. Skvělou volbou pro uložení textových dat je jazyk XML (Extensible Markup Language). Byl vyvinut konsorciem W3C na základě zkušeností s předchozími formáty dat. Je to volný a zdarma dostupný standard. Proto pro něj existuje řada volně dostupných parserů a nástrojů podporující práci s ním v nejrůznějších jazycích a editorech. Dovoluje šíření informací mezi různými architekturami počítačů a mezi různými aplikacemi, bez nutnosti několikanásobné konverze. Dokument XML obsahuje specifické elementy, entity a značky (tagy). Jednotlivé elementy se mohou vnořovat. Jazyk XML nám umožňuje definovat obsah dokumentu odděleně od jeho formátování.

XML je značkovací jazyk, který je podmnožinou jazyka SGML. SGML (Standard Gene-

Typ	Popis
REG_BINARY	Binární data v jakékoliv formě
REG_DWORD	Celé číslo reprezentované na 32 bitech
REG_DWORD_LITTLE _ENDIAN	Systém Windows je navržen na architekturu little-endian - na platformě Windows kompatibilní s s REG_DWORD
REG_DWORD_BIG_ENDIAN	REG_DWORD pro UNIX-like systémy fungující na architekture big-endian.
REG_EXPAND_SZ	Nulou zakončený řetězec, který obsahuje reference na proměnné prostředí, jako například %TEMP%.
REG_LINK	Nulou zakončený řetězec v kódování Unicode, který obsahuje symbolický odkaz
REG_MULTI_SZ	Sekvence nulou ukončených řetězců, zakončená prázdným řetězcem (\0).
REG_NONE	Nedefinovaný datový typ
REG_QWORD	Celé číslo reprezentované na 64 bitech
REG_QWORD_LITTLE _ENDIAN	Systém Windows je navržen na architekturu little-endian - na platformě Windows kompatibilní s REG_QWORD
REG_SZ	Nulou zakončený řetězec, který bude buď v Unicode nebo ASCII formátu, v závislosti na volaných funkcích

Tabulka 2.2: Datové typy v registrech, tabulka byla převzata z [10]

Element registru	Limit velikosti
Jméno klíče	255 znaků
Jméno hodnoty	255 znaků
Data	4 KB
Strom	Může být maximálně 16 úrovní hluboký.

Tabulka 2.3: Omezení velikostí dat v registrech WM a Win CE, převzato z [12]

ralized Markup Language) je ještě obecnější než XML. XML je ve skutečnosti meta-jazyk, čili jazyk, který se používá k popisu jiných jazyků. Nemá žádný pevně daný seznam jmen elementů. Poskytuje uživateli naprostou volnost při vytváření nových elementů. Doporučuje se samozřejmě používání jmen, která mají pro danou aplikaci smysl. Přílišná volnost ovšem může být i na škodu. Abychom se v záplavě značek neztratili a dodrželi korektnost dokumentu, je zde jazyk DTD (Document Type Definition). DTD definuje povolené prvky. DTD definici je poté možno použít pro validující parser. Tento softwarový analyzátor pak kontroluje validitu XML dokumentu vůči DTD definici. Kromě DTD jsou i novější technologie pro definici struktury dokumentu. Je jím například XML Schema. XML Schema je nástupcem DTD a má mnohem větší vyjadřovací schopnosti při popisování XML dokumentů. Příklad XML dokumentu je zobrazen v tabulce 2.4

```
<?xml version="1.0" encoding="UTF-8"?>
<clanek name="moudro">
    <veta>
        <slovo>XML</slovo>
        <slovo>je</slovo>
        <slovo>standard</slovo>
    </veta>
</clanek>
```

Tabulka 2.4: Příklad XML dokumentu

2.4 Databáze

Relační databáze jsou výkonným a široce rozšířeným úložištěm dat. Tyto databáze využívají pro dotazy jazyk SQL, který je relativně jednoduchý a výrazy v něm zapsané se velmi podobají přirozenému jazyku.

Platforma Windows Mobile obsahuje Microsoft SQL Server Compact Edition. Tento databázový systém je určený pro použití v mobilních zařízeních a jednodušších aplikacích pro Windows, kde se nepředpokládají záznamy v řádech miliónu řádků. Tento SQL server sdílí API s jinými edicemi Microsoft SQL serveru. Je proto velmi jednoduché v něm navrhovat, spravovat a využívat služeb databáze. Samotná databáze je uložena v souborech s příponou `.sdf`. Tyto soubory mohou být šifrovány. Samozřejmostí je ochrana databáze heslem. Oproti ostatním edicím Microsoft SQL serveru neběží jako služba, ale jako proces kvůli úspoře paměti. Instalační soubor má necelé 2 MB, z tohoto nám vyplývá, že nemůžeme očekávat plnou funkčnost klasických databázových systémů. Nechybí možnost definovat pohledy, či používat vnořené klauzule SELECT. Chybí uložené procedury a triggery. Z datových typů jsme ochuzeni o ne-unicodové CHAR a VARCHAR. Přesto je toto databázové řešení (vzhledem ke své velikosti) velice přívětivé.

Kapitola 3

Specifikace

Mojí cílovou platformou je Windows Mobile 6. Ve verzi Standard profily existují, je jich pět a nedají se vytvářet nové. Ve verzi Professional profily zcela chybí, proto se ve své práci zaměřím na tuto verzi (Windows Mobile 6 Professional).

Aplikace si klade za cíl mít jednoduché a intuitivní ovládání. Jelikož je pro verzi Professional, bude podporovat vytváření a kompletní správu profilů, protože zde nejsou vůbec obsaženy. Přepínání profilů bude realizováno třemi způsoby:

Standardně - na základě uživatelovy interakce. To znamená, že uživatel vezme do ruky zařízení a přepne profil manuálně.

Na základě události v kalendáři - předpokládejme, že uživatel má v kalendáři poznamenanou schůzku. Naše aplikace toto zjistí, vyhledá potřebné pravidlo v nastavení a přepne na požadovaný profil. Aby aplikace nereagovala na všechny události v kalendáři, využijeme příznaku schůzek „Free, Tentative, Busy, Out Of Office“. Na jaký příznak reagovat, bude možno změnit v nastavení.

Změna BTS - tzn. změna vysílače na který je telefon přihlášen. Uživatel si uloží id vysílací věže, když je například v práci nebo ve škole. Následně v aplikaci vytvoří pravidlo, určující že pokud je přihlášen na danou věž, má se aktivovat například profil práce. Po přihlášení na jiný vysílač se profil přepne automaticky na původní.

Aby se předešlo kolizím, budou zavedeny priority. Mohlo by se totiž stát, že uživatel bude přihlášen na BTS, pro kterou má vytvořené pravidlo. Zároveň by mu však nastala v kalendáři událost, pro kterou by měl vytvořené jiné pravidlo. Nejvyšší prioritu bude mít uživatelská změna profilu, druhou nejvyšší událost v kalendáři a nejnižší prioritu bude mít BTS. Aplikace by měla mít přehledné ovládání, umožňující provádět úkony rychle a interaktivně.

Kapitola 4

Analýza a návrh

V této části budu rozebírat použité technologie. Zaměřím se na jejich rozdílnost a budu diskutovat klady a zápory použití dané technologie v mé aplikaci.

4.1 Změna vyzvánění

Hlavní funkce celé aplikace je změna vyzvánění. Zjištění, jak změnit typ vyzvánění pro danou událost, je klíčové pro celou aplikaci. Nalezl jsem funkci `SndSetSound` [11], která umí nastavit hlavní vyzvánění, ale neposkytuje dostatečnou variabilitu. Neumožňuje nastavit vyzvánění pro notifikace. Vyzvánění na zařízeních s Windows Mobile se dělí do dvou kategorií

- Zvuky
- Notifikace

U zvuků je konfigurovatelné zda vibrovat, zda přehrát melodii, případně kombinace obou. Samozřejmě se dá nastavit přehrávaná melodie. Notifikace mají mnohem širší možnosti nastavení. Kromě nastavení vibrací a přehrávaného zvuku, má uživatel možnost zvolit zda zobrazit zprávu na úvodní obrazovce. Je zde také volba, jak dlouho má mobilní zařízení blikat v minutách.

Jak jsem se již zmínil, hledal jsem způsob, jakým nastavovat notifikace a zvuky, čili celé vyzvánění. Prohledával jsem proto registry systému, kde jsem nalezl řešení.

4.1.1 Notifikace

V klíčích `HKEY_CURRENT_USER\ControlPanel\Notifications\{xxx}`, kde místo hodnoty xxx je v registrech Class ID. Jedná se o jedinečný identifikátor dané notifikace. To je poněkud nepřehledné, strukturu těchto klíčů si ukážeme. Tabulka 4.1 znázorňuje příklad struktury registrového klíče pro některou z notifikací.

Default hodnota nám určuje název v dané jazykové mutaci. Hodnota Options nám určuje zvolené parametry dané notifikace. Pole Wave je cesta k nastavenému souboru s melodií. Duration je délka blikání a AvailableOptions určují, které z voleb jsou aktivní.

Nejzajímavější jsou pole Options a AvailableOptions. Obě fungují na stejném principu. Hodnoty se nastavují logickým OR na hodnoty znázorněné v tabulce 4.2

Chování je tedy následující: Pokud má pole Options hodnotu 0, na notifikaci se nereaguje. Hodnota 7 znamená, že je zapnutá vyzváněcí melodie (melodie dostupná v poli Wave),

Hodnota	Data
(Default)	ActiveSync: Begin sync
Options	1
Wave	\Windows\Infbeg.wav
AvailableOptions	3

Tabulka 4.1: Příklad hodnot v registrech v sekci notifikací

Hodnota	Činnost
0	Žádná činnost
1	Zapnuto vyzvánění melodie na adrese Wave
2	Zapnuté vibrace
4	Zapnuté blikání
8	Zobrazit notifikaci na hlavní obrazovce

Tabulka 4.2: Podporované hodnoty polí Options a AvailableOptions

také jsou hodnotou 7 značeny zapnuté vibrace a také blikání po dobu Duration. Protože $7 = 1 \mid 2 \mid 4$. Délka blikání uvedená v poli Duration je v minutách.

4.1.2 Zvuky

V klíčích `HKEY_CURRENT_USER\ControlPanel\Sounds\RingTone0` jsou uloženy všechny potřebné informace pro změnu vyzvánění. Hodnota, která nás zajímá, se skrývá v klíči `RingTone0`. Jedná se o vyzvánění pro příchozí hovory. Struktura tohoto klíče je zobrazena na tabulce 4.3.

Hodnota	Data
(Default)	
Sound	*vibrate*
SavedSound	\Windows\Ring-WindowsMobile.wma
Category	Ring
Script	apw3r

Tabulka 4.3: Příklad hodnot v registrech v klíči RingTone0

Pole (Default) je prázdné a pole Category určuje kategorii vyzvánění. Nejdůležitější pro ovládání vyzvánění jsou pole Sound a Script. Pole Sound může nabývat tří hodnot:

none znamená, že na příchozí hovor se nebude reagovat ani vibracemi, ani vyzváněním.

vibrate značí, že na příchozí hovor se bude reagovat pouze vibracemi.

cesta_k_souboru - pokud je v tomto poli cesta k souboru s melodií, pak se v reakci na příchozí hovor přehraje daná melodie.

Jak nastavit, aby telefon pouze vibroval, je z předchozího zcela jasné. Jak ovšem docílit, aby telefon přehrával melodii a zároveň vibroval? Zde nastupuje význam pole Script, které

nám poskytuje obrovskou variabilitu. Toto pole je v podstatě sekvence příkazů pro průběh vyzvánění. Tato sekvence je složena z písmen a číslic, jejichž význam teď blíže objasním tabulkou 4.4 a praktickým příkladem.

Hodnota	Data
a	Aktivuj zařízení
cN	Nastav hlasitost vyzvánění na N, které je uváděno v procentech
fN	Blikat po N sekund
p	Přehraj celou melodii vyzvánění, ta bude přehráta celá, před dalším pokračováním skriptu.
r	Opakuj celou předchozí sekvenci - pokud bude použit, měl by to být poslední příkaz.
vN	Vibrovat po N sekund
wN	Čekat po N sekund

Tabulka 4.4: Význam hodnot ve skriptu

Příklad:

Skript `apw5r` značí aktivuj zařízení, přehraj vyzváněcí melodii, počkej 5 sekund a následně opakuj.

Skript `ac33pc66pc100p` zaktivuje zařízení, nastaví hlasitost vyzvánění na 33 procent, přehraje vyzváněcí melodii, nastaví hlasitost vyzvánění na 66 procent, přehraje vyzváněcí melodii, nastaví hlasitost vyzvánění na maximum a přehraje naposledy vyzváněcí melodii.

4.2 Uložení dat

Naše aplikace musí umět efektivně ukládat data. Je potřeba ukládat především konfigurační data, to jsou data obsahující údaje o vysílači a adekvátním profilu (konfigurační data). Dále definice příslušných profilů. Jedná se o data v textové formě. Proto budeme zvažovat tato 3 úložiště dat:

- Databáze
- Registry
- XML

Každé Windows Mobile 6 zařízení je vybaveno a na rozměry zařízení výkonnou databází Microsoft SQL Server 2005 Compact Edition. Ukládání dat do databáze by však vzhledem k relativně malému množství dat a nevelkým požadavkům na výkon bylo zbytečně robustní řešení. I s ohledem na potřeby naší aplikace.

Uložení do registrů je lákavé. Naše aplikace musí s registry pracovat, ať už zvolíme jakékoliv úložiště dat. Je tomu tak z důvodu potřeby měnit vyzváněcí profily. Bylo by jednoduché data uložit do nově vytvořené větve registru, protože potřebné funkce je potřeba naimplementovat kvůli zajištění hlavní funkčnosti aplikace. Toto řešení jsem však nakonec nezvolil kvůli špatné přenositelnosti dat z hlediska uživatele.

Pro uložení dat jsem zvolil velmi rozšířený značkovací jazyk XML. Tento jazyk je pro uložení textových dat ideální. Každý profil je uložen ve svém vlastním souboru `Nazev.xml`.

Tyto soubory jsou uloženy v podadresáři, kde běží program. Tento podadresář se jmenuje `Profiles`. Při tomto uložení dat bude export profilů jednoduchý, jelikož si uživatel pouze zkopíruje obsah složky Profiles na jiné zařízení. Na tabulce 4.5 je vidět mnou navržená struktura XML dokumentu s příklady dat.

```
<?xml version="1.0"encoding="utf-16"?>
<ICRP.RingProfile ProfileName="Doma">
     <RingElement>
          <Key type="RingTypes">ASBegin</Key>
          <Value type="ICRP.RingNotifications">
               <Duration>0</Duration>
               <Options>1</Options>
               <AvailableOptions>3</AvailableOptions>
               <Wave>\Windows\Infbeg.wav</Wave>
               <Default>ActiveSync: Begin sync</Default>
               <RegPath>A877D658-239C-47a7-9304-0D347F580408</RegPath>
          </Value>
     </RingElement>

<!--Další notifikace uzavřené v tagu RingElement-->

     <RingElement>
          <Key type="RingTypes">RingTone0</Key>
          <Value type="ICRP.RingSounds»
               <RegPath>RingTone0</RegPath>
               <Default />
               <Sound>Windows\Alarm3.wma</Sound>
               <Script>apw3r</Script>
               <SavedSound>\Windows\Ring-Standard .wma</SavedSound>
               <Category>Ring</Category>
          </Value>
     </RingElement>
</ICRP.RingProfile>
```

Tabulka 4.5: Struktura XML dokumentu uchovávájícího profil

Nyní popíši strukturu tohoto XML dokumentu. Hlavička tohoto XML dokumentu nám poskytuje dvě informace. První informací je, že se jedná o verzi 1.0 standardu XML. Použité kódování je UTF-16. Jedná se o kódování s proměnnou délkou znaku. Každý Unicode znak je uložen sekvencí po 16-ti bitech.

Hlavním elementem je `ICPR.RingProfile`. Tento element má atribut `ProfileName`. Hodnota tohoto atributu je jméno profilu přidělené uživatelem. Element `ICPR.RingProfile` má v sobě vnořeno několik elementů `RingElement`. Tyto reprezentují jednotlivé registrové hodnoty notifikací, případně zvuků. `RingElement` obsahuje 2 elementy a to `Key` a `Value`. Element `Key` má atribut `type`, který nám říká jaký datový typ je reprezentován. V našem případě se jedná o enumeraci `RingTypes` a její hodnota je AsBegin. Jednoznačně určuje

o jakou notifikaci nebo zvuk se jedná. Druhým elementem je `Value`. Má opět atribut `type` určující zda jde o notifikaci nebo zvuk.

V případě, že se jedná o notifikaci, obsahuje element `Value` tyto synovské elementy. `Duration` označuje délku blikání. Element `Options` je ekvivalentem k registrové hodnotě Options. Stejně tak element `AvailableOptions`, vztahující se ke stejnojmenné registrové hodnotě, která reprezentuje dostupné možnosti dané notifikace. Element `Wave` obsahuje úplnou cestu k souboru s melodií určenou pro danou notifikaci. `Default` označuje název notifikace uvedené v jazykové mutaci systému. Posledním elementem je `RegPath`, označující daný registrový klíč, kde se nachází naše notifikace.

Pokud `Value` reprezentuje zvuk, obsahuje tyto synovské elementy. `RegPath` určující cestu v dané větvi registru. Element `Default`, který nemusí být vždy využit. Element `Sound`, který určuje, jak se bude reagovat na příchozí hovor. Hodnoty tohoto pole byly popsány dříve. Element `Script`, který je ekvivalent k stejnojmenné registrové hodnotě. `SavedSound` obsahuje uloženou úplnou cestu k souboru s vyzváněcí melodií. Poslední element `Category` určuje do jaké kategorie spadá daný zvuk.

Jak vyplývá z předchozího, návrh struktury tohoto XML dokumentu vychází z velké části z architektury uložení notifikací a zvuků v registrech. Tato struktura je velmi přehledná.

Pro ukládání konfigurace byla zvolena následující struktura XML dokumentu 4.6. Stejně jako v předešlém XML se jedná o standard XML ve verzi 1.0. Použité kódování je opět UTF-16. Hlavním elementem je Configuration. Má v sobě vnořeny následující elementy. `ActiveRILMon` určující zda-li bylo při posledním použití aktivováno monitorování pomocí RIL. Element `ActiveCalMon` nám určuje zda bylo zapnuto monitorování kalendáře. Element `CurrentProfileName` nese informaci o naposledy aktivním profilu. V konfiguraci je také uloženo, který profil se má aktivovat v případě reakce na událost v kalendáři. Tuto informaci nese element `CalMonProfileName`. Abychom věděli, na který příznak schůzek reagovat, je v nastavení element `ReactingAppStatus`. Element `bts` obsahuje subelementy `rule`. Každý z těchto elementů má 3 subelementy obsahující potřebné informace o vysílači. Jmenují se `cellId, name` a `profile`. V elementu `cellId` je uloženo identifikační číslo vysílače, přičemž `name` je uživatelem definované jméno tohoto vysílače. Element `profile` určuje, který profil přísluší k danému vysílači.

4.3 Získání ID vysílače

Nejedná se o triviální záležitost. Na platformě Windows Mobile, lze získat ID vysílače třemi rozdílnými způsoby, bohužel ani jeden z nich není 100% spolehlivý.

- AT příkazy
- Něoprávněné přístupy (tzv. hacky) do paměti
- Radio Interface Layer

První možností se nebudeme příliš zabývat. Pro zjištění ID vysílače lze využít AT příkazů GSM modulu. Toto řešení je ovšem závislé na daném GSM modulu a vyžaduje podrobné znalosti dokumentace. Je natolik specifické, že se jím nemá cenu dále zabývat.

Druhou možností je využít toho, že identifikační číslo buňky(Cell ID) vysílače se nachází na určité adrese paměti. Najít tuto adresu ovšem není triviální a je nutno se podrobně orientovat v oblasti uložení čísel v paměti daného mobilního zařízení. Nejdříve zjistíme od operátora, případně za pomoci zařízení speciálně určeného k získávání identifikačního čísla

```
<?xml version="1.0"encoding="utf-16"?>
<Configuration>
     <ActiveRILMon>false</ActiveRILMon>
     <ActiveCalMon>false</ActiveCalMon>
     <CurrentProfileName>Default</CurrentProfileName>
     <CalMonProfileName>Default</CalMonProfileName>
     <ReactingAppStatus>Busy</ReactingAppStatus>
     <bts>
          <rule>
               <cellId>444-66-4</cellId>
               <name>Jmeno</name>
               <profile>PROFIL</profile>
          </rule>
          <rule>
               <cellId>444-66-60454</cellId>
               <name>Jmeno</name>
               <profile>PROFIL</profile>
          </rule>
     </bts>
</Configuration>
```

Tabulka 4.6: Struktura konfiguračního XML dokumentu

buňky, toto ID. Lze použít i MCC(Mobile Country Code), ovšem hledání je pak delší. Když víme toto číslo, můžeme tuto sekvenci bajtů lokalizovat v paměti zařízení. Z těchto adres pak stačí pouze číst. Jelikož poloha v paměti se může velmi lehce změnit a není nikde garantovaná, je tato metoda velmi nepraktická. Může se lišit mezi různými sestaveními operačního systému. Není výjimkou, že si výrobce upraví operační systém pro vlastní potřebu, čímž může taktéž změnit polohu identifikačního čísla buňky v paměti. Programátor by musel zjišťovat adresu v paměti pro všechny tyto případy. Pro bližší porozumění je možno se podívat do literatury [14].

Třetím způsobem je využití již zmiňovaného Radio Interface Layer, jehož teoretické pozadí je popsáno v sekci 2.2.1. Tato metoda je nesporně lepší z hlediska přenositelnosti a kompatibility. Není ovšem taky ideální, protože RIL není oficiálně podporován na Windows Mobile, ale pouze na Windows Embedded CE. Není proto dostupný v běžně dodávaném SDK a vývoj v nativním kódu je značně ztížen. Na internetu ovšem lze sehnat hlavičkový soubor `ril.h` společně s knihovnou `ril.lib`. V managed kódu je nutno se spolehnout na definice struktur, funkcí z dokumentace a funkce používat za pomoci rozhraní „P/Invoke“ (Platform Invocation Services). P/Invoke je Common Language Infrastructure - na Windows Mobile pod názvem Microsoft's Common Language Runtime. Jedná se o schopnost interpretovaného kódu volat nativní funkce systému, v tomto případě z dynamické knihovny „ril.dll“. Většina výrobců dodržuje implementaci podle dokumentace, jsou zde ovšem výjimky. Na velké části přístrojů s Windows Mobile 6 Radio Interface Layer funguje, zbylá část zařízení je bohužel o tuto funkci ochuzena.

Samotné id buňky není dostatečně specifické pro globální potřebu. Tím se myslí, že vysílač v jiné lokalitě nebo jiném státě by mohl mít stejné ID. Proto funkce zjišťující informace o vysílací věži vrací nejenom Cell id, ale i například: „Mobile Country Code“ reprezentující kód dané země, „Mobile Network Code“ reprezentující kód příslušící danému telefonnímu operátorovi, „Location Area Code“ určující kód dané oblasti, „GPRS Cell ID“ kód buňky poskytující GPRS připojení a samotné „Cell ID“. Dostatečně specifická identifikace vysílače pro naše potřeby je kombinace tří hodnot a to: Location Area Code, Cell ID a Mobile Country Code.

Pro zjištění identifikačního čísla buňky vysílače jsem zvolil třetí způsob, to znamená využití Radio Interface Layer. Tento způsob byl zvolen, protože je nejvíce univerzální a funguje na největší škále mobilních zařízení.

4.4 Přístup do kalendáře

Přístup k aktuální události v kalendáři je možno zrealizovat těmito způsoby:

- Pocket Outlook Object Model (Microsoft.WindowsMobile.PocketOutlook - .NET)
- State and Notifications Broker (Microsoft.WindowsMobile.Status - .NET)

Naše aplikace potřebuje mít přístup k datům z kalendáře. K tomuto účelu je zde API POOM. Existuje jak v nativní, tak v managed verzi, kde je umístěno v namespace Microsoft.WindowsMobile.PocketOutlook. U každé události se dají sledovat rozličná data. Lze monitorovat ať již změny celé události, tak změny jejích součástí (změna předmětu události, názvu události, časového rozvržení, ...). Jak již bylo zmíněno, Pocket Outlook Object Model má v managed verzi jednu zásadní nevýhodu. Tou je neschopnost reagovat na události, které vzniknou mimo tuto aplikaci. Toto je velká limitace pro naši aplikaci a celé API je prakticky nepoužitelné. Naštěstí lze nalézt řešení. Pan Jim Wilson napsal jednoduchou třídu, která zaobaluje nativní API (wrapper) a poskytuje tak jeho možnosti pro interpretovaný kód v prostředí .NET. Odkaz na toto řešení je možno nalézt v literatuře [13]. Pokud bychom potřebovali takto ucelenou správu, je použití jeho tříd skvělým řešením pro programátory managed aplikací.

Naše aplikace ovšem takto ucelenou správu nepotřebuje. Pokud se zamyslíme, jediné co potřebujeme vědět je informace, zda-li je v tuto chvíli aktivní jakákoliv událost. A následně potřebujeme zjistit její příznak. Tzn. zda-li je její příznak ten, který monitorujeme podle nastavení. K tomuto účelu se výborně hodí API State and Notifications Broker, které je umístěno v namespace Microsoft.WindowsMobile.Status v managed verzi API. Poskytuje aktuální informace o mobilním zařízení. Mezi těmito informacemi jsou například údaje o aktuální schůzce, kde se lze lehce zaregistrovat na odběr notifikací o nové schůzce. Toto API používá definice tříd, enumerací a struktur z POOM. Důvod pro použití tohoto API je zřejmý. A to sice, že jak v managed, tak v nativní verzi poskytuje informace o změně údajů nejenom z naší aplikace, ale i při změně aplikacemi externími.

Z předchozího představení technologií vyplývá, že nejvhodnějším způsobem pro zjišťování aktuální události v kalendáři je využití State and Notification Broker. Poskytuje nám nejjednodušší přístup k aktuální schůzce v kalendáři.

4.5 Programovací jazyk

Výběr programovacího jazyka je velmi důležitá část návrhu naší aplikace. Pro vývoj na platformě .NET v managed kódu přichází do úvahy jazyk C#. Pro nativní vývoj pak jazyk C++. Jak jsme zjistili z předchozích kapitol, jsou oba jazyky vyrovnanými soupeři. API pro jazyk C++ má přece jenom širší paletu možností a netrpí některými nepříjemnými omezeními popsanými výše. Navzdory těmto omezením je programovací jazyk C# plnohodnotným soupeřem. Jak jsme však zjistili, pro potřeby naší aplikace lze všechna tato omezení vyřešit, případně obejít. Jazyk C# nám nabízí oproti C++ mnohem přívětivější rozhraní. Na základě těchto podkladů jsem se rozhodl pro implementaci zvolit právě C#.

Kapitola 5

Implementace

Celá aplikace je naiplementována v jazyce C# na platformě .NET Compact Framework, verze 2.0. Jeden z hlavních požadavků vycházející ze specifikace [3] byla uživatelská přívětivost. Rozhodl jsem se proto, že moje aplikace bude pracovat jako takzvaný Today Screen plugin. Tyto pluginy se zobrazí přímo na hlavní obrazovce telefonu, proto je k nim jednoduchý a rychlý přístup. Pro napsání tohoto pluginu je potřeba použít Today Screen API. Toto API ale bohužel není dostupné na platformě .NET. Tím je vývoj Today Screen pluginů značně ztížen. Protože se nejedná o stěžejní část mé práce, rozhodl jsem se pro řešení tohoto problému využít volně dostupný software třetích stran. ChrisTec Managed Today Screen Item Framework [6] poskytuje interpretovanému kódu jednoduchou tvorbu Today Screen pluginu. Tento framework po nainstalování a restartu zařízení použije dostupné `dll` soubory, které se nacházejí ve stejném umístění a zaktivuje je.

5.1 ChrisTec Managed Today Screen Item Framework

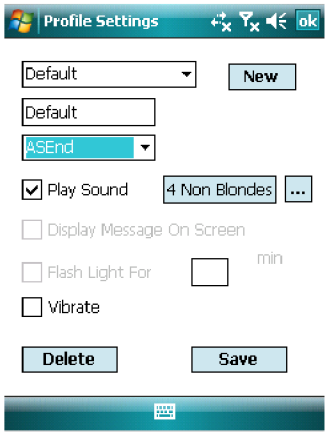
Je volně dostupný framework umožňující vývoj Today Screen pluginu v managed kódu. Plugin zaobaluje nativní API, jedná se o takzvaný wrapper. Jeho použití je velmi jednoduché. Stačí přidat referenci na dynamickou knihovnu implementující tento framework a plugin používat jako takzvaný User Control ve Visual Studiu. Na tento User Control se již notoricky známým způsobem přidávají klasické .NET ovládací prvky. Celý náš projekt se zkompiluje jako Class Library `dll` a distribuuje společně s tímto frameworkem. Celý framework je poskytován pod „New BSD Licence“, která poskytuje dostatečnou volnost.

5.2 Rozhraní aplikace

`TodayScreen` je třída, která je načtena a zobrazena jako Today Screen Plugin. Tato třída je odvozena od System.Windows.Forms.UserControl. Její rozhraní je velmi jednoduché. Z hlavní obrazovky hned víme, který profil je právě aktivní a jednoduchým kliknutím na něj je možno provést změnu profilu. Po kliknutí na jméno aktuálního profilu se totiž ihned zobrazí ContextMenu s nabídkou všech uložených. ContextMenu bylo zvoleno vzhledem k největší jednoduchosti pro uživatele, nezobrazují se totiž žádné zbytečné dialogy a změna je opravdu rychlá. Dále jsou zde dva CheckBox, které umožňují změnu aktivaci a deaktivaci monitorování položek kalendáře, případně Radio Interface Layer. Poslední ovládací prvek je tlačítko Settings. Po zmáčknutí se zobrazí ContextMenu, kde má uživatel na výběr ze tří možností. Může nastavit Profily, RIL nebo monitorování kalendáře. Každá

z těchto možností otevře samostatný dialog. Všechny tyto dialogy jsou odvozeny od třídy System.Windows.Forms.Form.

Dialog Profile settings je implementován ve třídě FormProfileSettings. Můžeme jej vidět na obrázku 5.1. Skládá se z mnoha ovládacích prvků zajišťujících pohodlnou správu profilů. Možnosti editace jednotlivých možností jsou aktivovány na základě vybrané notifikace nebo zvuku, protože každá notifikace má jinou hodnotu pole `AvailableOptions` v registrech. Uživatel může jednoduše vybrat vhodnou melodii pomocí dialogu na otevření souboru s melodií(`OpenFileDialog`). Změna se provede až po kliknutí na tlačítko Save.



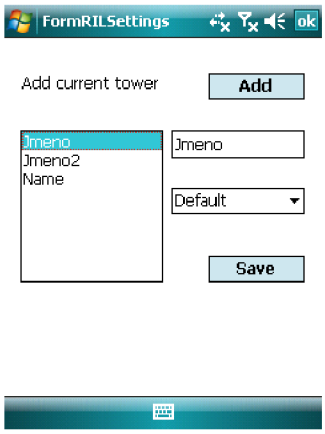
Obrázek 5.1: Formulář umožňující správu profilů

Dialog Calendar Monitor Settings, zobrazený na obrázku 5.2 je ve v podstatě strohý. Obsahuje volbu, na který příznak schůzky reagovat a také jaký profil má být aktivován. samozřejmě je zde tlačítko pro uložení zvolených voleb.

Obrázek 5.2: Formulář pro editaci nastavení monitorování kalendáře

Dialog RIL settings, který je k vidění na obrázku 5.3, se stará o správu vysílačů a adekvátních profilů k nim příslušícím. Je zde tlačítko pro přidání nového vysílače. Nejdříve se vysílač vytvoří s novým výchozím jménem. Uživatel si ho může následně přejmenovat. Samozřejmostí je kontrola, zda se uživatel nesnaží přidat stejný vysílač dvakrát. Ke každému vysílači lze nastavit profil. Tento profil je aktivován, pokud se zařízení přihlásí na daný

vysílač. Tento dialog nelze aktivovat, pokud je GSM modul vypnutý, případně pokud není podporován a rozhraní RIL není dostupné.



Obrázek 5.3: Formulář pro správu vysílačů

5.3 Aplikační části

Zde popíši hlavní třídy využívané v naší aplikaci, jejich použití a zajímavé implementační detaily.

5.3.1 Konfigurace

Třída uchovávající nastavení celé aplikace se jmenuje `Configuration`. Implementace této třídy je založena na návrhovém vzoru singleton (jedináček). Je definována jako `sealed`, to znamená, že se od ní nedá dědit a nehrozí tedy poškození návrhového vzoru. Hlavní funkce jedináčka je ta, že v celém programu se vyskytuje pouze jedna instance. Protože naše implementace obsahuje pouze privátní konstruktor, není možné vytvořit instanci třídy mimo třídu samotnou. Statický atribut třídy `_instance` obsahuje referenci na jedinou instanci třídy v programu. Tento atribut je `readonly`, což znamená, že do něj může být přiřazeno pouze v průběhu statické inicializace. Je přístupný pomocí statické třídní vlastnosti `Instance`. Veškeré atributy třídy uchovávající nastavení, jsou zpřístupněny pomocí vlastností. Je mezi nimi například jméno profilu, na který se má telefon přepnout při události v kalendáři, informace o vysílačích, atd. Metody `Load a Save` nahrají, případně uloží konfigurační data. Příklad užití můžete nalézt v 5.1.

5.3.2 Vyzváněcí profil

Jednotlivé vyzváněcí profily jsou reprezentovány třídou `RingProfile`. Nejdůležitější metoda této třídy je `Activate`, jak napovídá její název, po jejím zavolání se aktivuje daný profil. Každý profil má své jméno a to je zpřístupněno pomocí vlastnosti `ProfileName`. Jak již bylo zmíněno v předchozích kapitolách, vyzvánění se skládá z notifikací a zvuků. Proto třída `RingProfile` obsahuje datovou strukturu `Dictionary`, která obsahuje reference na Notifikace a zvuky. Tato datová struktura, a tím i jednotlivé notifikace a zvuky, je přístupná přes vlastnost `Rings`. Pro uložení profilu jsou k dispozici dvě třídy a to `XMLPersistance` a

```
//vytvoření instance
Configuration conf = Configuration.Instance;

//načtení konfigurace
conf.Load();

//změna konfigurace
conf.BusyProfileName = "Default";

//uložení nastavení
conf.Save();
```

Tabulka 5.1: Příklad použití třídy `Configuration`

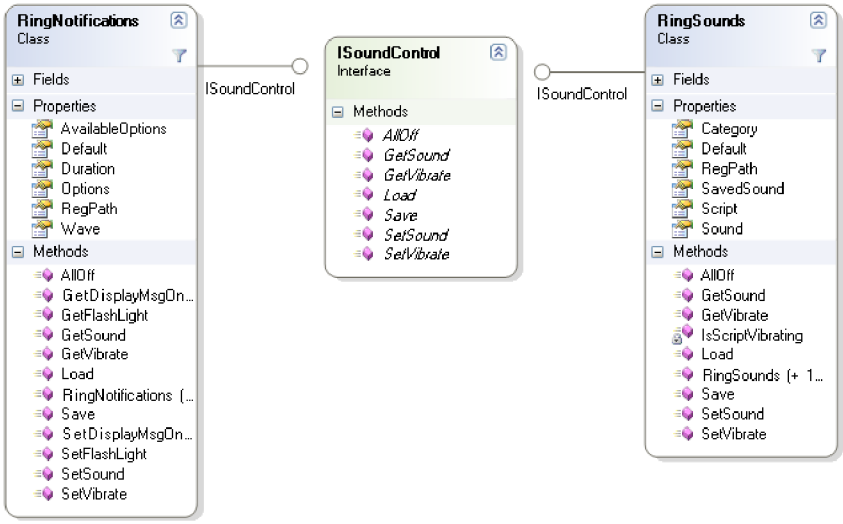
`RegistryPersistance`. Jak napovídají jejich názvy, starají se o uložení profilu do XML souboru a do registru. Obě třídy implementují společné rozhraní `IRingPersistanceAdapter`. Toto rozhraní definuje pouze dvě metody, jejichž definice jsou `RingProfile Load()` a `bool Save(RingProfile)`.

Třída `XMLPersistance` zajišťuje pohodlné uložení dat do XML souboru. Požadované jméno souboru lze dodat přes vlastnost `FileName`. Mezi další důležité metody patří `GetSavedProfiles()`, která získá jména všech uložených profilů pomocí této třídy. Pro ukládání do XML souboru byl použit `XmlTextWriter`. Práce s ním je relativně pohodlná, ovšem nedosahuje kvalit a jednoduchosti práce, které se dá dosáhnout s třídou `XmlSerializer`. Při použití této třídy nemusí programátor dokonce ani volit jména elementů, protože objekt si je zvolí sám a automaticky ukládá všechny vlastnosti objektu. Tyto vlastnosti zpřístupňuje pomocí reflexe. Naneštěstí si tato třída nerozumí se mnou použitou datovou strukturou `Dictionary`. Z toho důvodu jsem musel použít `XmlTextWriter` a uložit vlastnosti třídy manuálně.

`RegistryPersistance` je třída starající se o uložení souborů do registru zařízení. Uložením profilu s využitím této třídy dosáhneme jeho aktivace.

Zvuky jsou reprezentovány třídou `RingSounds` a notifikace třídou `RingNotifications`. Obě třídy implementují rozhraní `ISoundControl`, jak je znázorněno na obrázku 5.4.

Rozhraní `ISoundControl` definuje společné možnosti vyzvánění zvuků a notifikací. Obsahuje metody `string GetSound()` a `bool GetVibrate()`, které nám dovolují jednoduše zjistit, zda třída implementující toto rozhraní má nastavenu vyzváněcí melodii, případně zda vibruje. Pro nastavení vyzvánění jsou zde metody `void AllOff()`, `void SetSound(bool, string)` a `void SetVibrate(bool)`. Dají se aplikovat postupně, čili samozřejmě jde nastavit, aby telefon na danou událost přehrával danou melodii a zároveň vibroval. Docílíme toho zavoláním metod `SetSound` a `SetVibrate` v libovolném pořadí. Metoda `AllOff` samozřejmě zruší jak vyzváněcí melodii, tak vibrace. Jednotlivé třídy implementují kromě tohoto společného rozhraní i metody specifické pouze pro ně. Proto například v třídě `RingNotifications` jsou navíc metody jako `bool SetDisplayMsgOnScreen(bool)`, případně `int GetFlashLight()`.



Obrázek 5.4: Rozhraní `ISoundControl` a třídy jej implementující

5.3.3 Monitorování potřebných událostí

Sledování kalendáře zajišťuje třída `CalendarMonitor`. Využívá třídy z namespace Microsoft.WindowsMobile.PocketOutlook a Microsoft.WindowsMobile.Status. Pomocí nich monitoruje aktuální událost v kalendáři a její příznak. Příklad použití je k vidění zde 5.2. Třída `CalendarMonitor` vystavuje veřejnou událost `AppointmentChanged`, na kterou se můžeme zaregistrovat za použití delegáta `AppointmentChangeDelegate`. Po zavolání metody `Start()` se zahájí odběr zpráv a uživatel začne být informován o každé změně, která v kalendáři na aktuální události nastala. Třída implementuje také metodu `Stop()`, která zastaví odebírání zpráv a uživatel již není dále notifikován o změnách.

```
//vytvoření instance
CalendarMonitor calMon = new CalendarMonitor();

//zaregistrování k odběru událostí
calMon.AppointmentChanged +=
new CalendarMonitor.AppointmentChangedDelegate(calMon_AppointmentChanged);

calMon.Start()
void calMon_AppointmentChanged(object sender, Appointment app)
{
     //reakce na danou událost
}
```

Tabulka 5.2: Příklad použití třídy `Configuration`

Pro zjišťování identifikačního čísla vysílače obsahuje naše aplikace třídu `RadioInter-`

faceLayer. Tato třída využívá stejnojmenné API, které, jak již bylo zmíněno dříve, nemá svůj ekvivalent na platformě .NET. Je proto nutno použít invokaci funkcí z dynamických knihoven (ril.dll). Celá třída RadioInterfaceLayer má podobně navržené rozhraní jako třída CalendarMonitor. Stačí se opět pouze zaregistrovat na odběr událostí CellTowerChanged za použití delegáta CelltowerChangedDelegate. Začátek a konec odběru zpráv je opět pomocí metod Start() a Stop(). Po zavolání metody Start() je ihned zjištěno identifikační číslo vysílače pomocí metody GetCellTowerInfo() a uživatel je o něm informován. Také je spuštěn časovač, který periodicky volá tuto metodu a v případě změny vysílače vyvolá příslušnou událost. Abychom skryli, že volání funkcí z modulu RIL je asynchronní, je pro synchronizaci použit objekt AutoResetEvent. Po invokaci funkce RIL_GetCellTowerInfo je zavolána metoda objektu AutoResetEvent a to WaitOne(). Vykonávání kódu je zastaveno až do okamžiku zavolání metody Set(). Tato metoda je volána z callback funkce, která je předána funkci RIL_Initialize. Bližší ukázka implementace je zobrazena v tabulce 5.3.

```
...
// zinicializujeme rozhraní RIL
result = RIL_Initialize(1, new RILRESULTCALLBACK(rilResultCallback),
                        null, 0, 0, out handleRIL);
...
// zavoláme funkci pro zjištění id vysílače
result = RIL_GetCellTowerInfo(handleRIL);

// čekáme na asynchronní odpověď (rilResultCallback)
waitForGCTI.WaitOne();

// skončili jsme, můžeme uvolnit handle
RIL_Deinitialize(handleRIL);
```

Tabulka 5.3: Ukázka implementace třídy RadioInterfaceLayer

Třídy CalendarMonitor a RadioInterfaceLayer jsou využívány třídou icrp, která má hlavní kontrolu nad vyzváněním. Za pomoci těchto tříd monitoruje uživatelem zvolené události a vykonává případnou změnou profilu.

5.3.4 Udržení notifikací ve funkčním stavu

Jeden z problémů, který se vyskytl, byly nefungující notifikace, pokud má telefon vypnutý zvuk ikonou na hlavním panelu. Tento problém byl odhalen až po naimplementování značné části aplikace. Bylo jej proto nutno urychleně řešit. V průběhu hledání jsem objevil v registrech klíč HKCU\ControlPanel\Notifications\ShellOverrides\Mode. Experimentálně jsem zjistil, že pokud jsou zvuky zapnuté, je hodnota Mode nastavena na 0. Další možné hodnoty jsou 1, pokud je telefon přepnut na vibrace a 2 pokud jsou vypnuty všechny zvuky. Ihned po přepnutí nastavení z hlavního panelu je změna reflektována do registrové hodnoty. Nejdříve jsem zkusil editovat hodnotu ručně přes registrový editor, bohužel tato zpětná změna nefunguje a nastavení systému se nepřepne zcela. Provedl sem několik ne-

úspěšných pokusů, zahrnujících například rozesílání zpráv po systému signalizujících, že se změnilo nastavení. Všechny skončily neúspěšně, kromě jedné. Funkce `SndSetSound` [11] umožňuje nastavit vyzvánění, ovšem její možnosti na Windows Mobile jsou značně omezené. Umožňuje nastavit vyzváněcí melodii pro příchozí hovor, případně vibrace. Při zavolání této funkce s parametry `SND_EVENT_ALL` a `SND_SOUNDTYPE_ON` jsou zapnuty všechny zvuky a notifikace jsou opět funkční. z tohoto důvodu je tato funkce volána okamžitě po spuštění aplikace. Abychom byli informování, zda-li uživatel nevypnul zvuky a hlavním panelu, monitoruje naše aplikace hodnotu `Mode` v již dříve zmíněném registrovém klíči. V případě změny adekvátně reaguje přepnutím aplikace do neaktivního stavu.

Implementace nebyla jednoduchá, protože funkce `SndSetSound` se nachází pouze v nativním API. Použil jsem proto P/Invoke, které mi dovoluje po naimplementování patřičných struktur volat funkce přímo z dynamických knihoven. V tomto případě z knihovny `aygshell.dll`. Ukázka jedné u implementovaných struktur je zobrazena v tabulce 5.4

```
struct SNDFILEINFO
{
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = MAX_PATH)]
    public string szPathNameNative;

    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = MAX_PATH)]
    public string szDisplayNameNative;

    public SND_SOUNDTYPE sstType;
}
```

Tabulka 5.4: Příklad implementace struktury pro invokaci `SndSetSound`

Kapitola 6

Závěr

Bakalářská práce se zabývá inteligentní správou vyzvánění. Během vytváření aplikace jsem musel nastudovat různé materiály. Vzhledem k charakteru práce, která má poměrně netradiční téma, byla většina zdrojů internetového původu. Výsledná aplikace poskytuje uživatelům jednoduchou kontrolu nad vyzváněcími profily a dovoluje jim vytvářet různá pravidla pro automatizovanou změnu vyzváněcího profilu. Tyto změny jsou realizovány na základě existence události v kalendáři, která splňuje uživatelem zadaná kritéria. Případně na základě pravidla obsahujícího specifický vysílač a k němu korespondující profil.

Celá tato práce pro mne byla velmi přínosná. Pronikl jsem díky ní mnohem hlouběji do operačních systémů v mobilních zařízeních a také si osvojil programovací jazyk C# na platformě .NET Compact Framework. Tyto získané zkušenosti jsou pro mne do budoucna velkým přínosem. Segment mobilních zařízení je velmi energicky expandující a je zde proto velký potenciál do budoucna. Stejně tak jazyk C#, ve kterém jsem se během tvorby této práce značně zdokonalil, je na trhu práce velmi ceněný. Jeho znalost bude při řešení dalších prací jistě přínosem.

V rámci budoucího vývoje aplikace by bylo vhodné zlepšit podporu pro získávání identifikačního čísla vysílače. Mnou použitá metoda bohužel nefunguje na všech mobilních zařízeních. Aplikace by také mohla být rozšířena o další verze, které by podporovaly nová agresivní grafická prostředí jako například TouchFlo 3D od firmy HTC.

Literatura

[1] Google Android Reference. [online], poslední modifikace 1. března 2009. [cit. 2009-04-30].
URL http://developer.android.com/reference/packages.html

[2] Google Architecture. [online], poslední modifikace 1. března 2009. [cit. 2009-04-30].
URL http://developer.android.com/guide/basics/what-is-android.html

[3] iPhone OS. [online], poslední modifikace 8.července 2008 . [cit. 2009-04-30].
URL http://developer.apple.com/iphone/i

[4] Symbian OS Reference. [online], poslední verze 2008-02-00. [cit. 2009-04-30].
URL http://developer.symbian.com/main/documentation/sdl/symbian94/sdk/doc_source/index.html

[5] Windows Mobile Developer Center. [online], poslední modifikace 2009 . [cit. 2009-04-30].
URL http://msdn.microsoft.com/en-us/windowsmobile/

[6] Fairbairn, C.: ChrisTec Managed Today Screen Item Framework. [online], poslední modifikace 21.února 2008 . [cit. 2009-04-30].
URL http://managedtodayscreen.codeplex.com/

[7] Lacko, L.: *Programujeme mobilní aplikace ve Visual Studiu .NET*. Computer Press, 2004, iSBN 80-251-0176-2.

[8] Microsoft: Phone Features. [online], poslední modifikace 17.září 2008 . [cit. 2009-04-30].
URL http://msdn.microsoft.com/en-us/library/bb416430.aspx

[9] Microsoft: Registry Value Types. [online], poslední modifikace 12.března 2008 . [cit. 2009-04-30].
URL http://support.microsoft.com/kb/256986

[10] Microsoft: Registry Value Types. [online], [cit. 2009-04-30].
URL http://msdn.microsoft.com/en-us/library/ms724884.aspx

[11] Microsoft: SndSetSound. [online], poslední modifikace 28.srpna 2008 . [cit. 2009-04-30].
URL http://msdn.microsoft.com/en-us/library/bb416592.aspx

[12] Microsoft: Using the Registry. [online], [cit. 2009-04-30].
URL http://msdn.microsoft.com/en-us/library/ms900347.aspx

[13] Wilson, J.: Monitoring Pocket Outlook Changes that Occur Outside of your .NET CF Application. [online], poslední modifikace 18. října 2007. [cit. 2009-04-30].
URL
www.pluralsight.com/community/blogs/jimw/archive/2007/10/18/48800.aspx

[14] YUAN: How to use C# in Windows mobile devices to capture CellID. [online], poslední modifikace 2. února 2007. [cit. 2009-04-30].
URL http://mail.cs.nchu.edu.tw/~s9356003/CellID/GetCellid_CHS.htm