MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Mobilní aplikace Kniha jízd

BAKALÁRSKA PRÁCA

Ján Sulin

Brno, podzim 2009

Prehlásenie

Prehlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Vedúci práce: Mgr. Jan Kasprzak

Zhrnutie

Cieľom tejto bakalárskej práce je návrh a následná implementácia mobilnej aplikácie typu kniha jázd, ktorá umožňuje pohodlnú a jednoduchú správu vozidla pre osobnú potrebu. Výsledok práce je aplikácia Karavana pre operačný systém Maemo určený na každodenné používanie.

Kľúčové slová

Maemo, kniha jízd, Nokia, Gtk+, Sqlite

Obsah

Kapitola 1

Úvod

Výrazným prejavom civilizačnej vyspelosti a úrovne vedy a techniky, ktorý si paradoxne veľa ľudí neuvedomí, je možnosť cestovať a presúvať sa na veľké vzdialenosti prakticky s minimálnym úsilím. Autá a ďalšie prostriedky dopravy sa stali nedeliteľnou súčasťou našej každodennej reality, či už pracovnej alebo súkromnej, a nutnosť spravovať ich sa naplno prejavila. Každý dopravný prostriedok, aj keď je výsledkom ľudského umu a tvorivosti a bol vytvorený tak, aby jeho použitie bolo čo najjednoduchšie, potrebuje špeciálny druh starostlivosti, ako napríklad výmenu oleja, povinné prehliadky a podobne. Kedysi sa celá správa automobilu odohrávala na papieri a od majiteľa alebo správcu auta vyžadovala, aby si pamätal množstvo údajov a aby celú správu robil ručne. Rozvoj informačných technológií však priniesol aj do tejto oblasti pokrok a počítače zjednodušili a zjednotili správu dopravného prostriedku a činností popri ňom vykonávaných. V súčastnosti existuje skupina aplikácií označovaných ako kniha jázd pre rôzne počítačové platformy, ktoré túto činnosť vykonávajú.

1.1 Ciele práce

Cieľom tejto bakalárskej práce je návrh a implementácia aplikácie, ktorá bude reprezentovať knihu jázd pre mobilné PDA [1]. Na dosiahnutie tohto cieľa bola vytvorená aplikácia Karavana, ktorá umožňuje pohodlný manažment automobilu z pohľadu používateľa.

1.2 Štruktúra práce

V druhej kapitole sú stručne zhrnuté vlastnosti a charakteristiky dostupných aplikácií pre správu automobilu na rôznych platformách. Analyzovaný je spôsob práce s aplikáciou, návrh užívateľského rozhrania a jej kľúčové vlastnosti, prípadne jej špecifiká. Rozoberá sa hlavne ich typické použitie a vplyv návrhu v kontexte použitej platformy na prácu s ňou.

V tretej kapitole sú stručne zhrnuté teoretické základy technológií použitých pri vývoji aplikácie a obsahuje všeobecný popis platformy Maemo ako aj vývojového prostredia. Vysvetlené sú princípy vývoja aplikácie pre platformu Maemo spolu s jednotlivými použitými a dostupnými API [2]. Kapitola je venovaná aj problému cross-kompilácie. Hlavný dôraz je

1. Personal Digital Assistent <http://en.wikipedia.org/wiki/Personal_digital_assistant>.
2. Application Programming Interface.

kladený na analýzu a vhodnosť rôznych prístupov ukladania užívateľských dát a z toho plynúcich implikácií pre aplikáciu.

Vo štvrtej kapitole je prezentovaná aplikácia Karavana. Táto aplikácia umožňuje užívateľovi spravovať dáta o dopravnom prostriedku a pomáha analyzovať výdavky a bežné úkony, ktoré sú spojené s prevádzkou dopravného prostriedku. Kapitola obsahuje analýzu a implementačné detaily aplikácie Karavana, ako aj popis a zdôvodnenie rozhodnutí urobených počas vývoja.

Záverečná piata kapitola obsahuje zhodnotenie celého projektu.

1.3 Použité technológie

Platforma pre aplikáciu Karavana bola vopred daná, a to Maemo, čo podmienilo aj výber použitých technológií na vývoj. Na správne fungovanie aplikácie je potrebný digitálny osobný asistent s operačným systémom Maemo [3].

1.3.1 Programovací jazyk

Celá bakalárska práca bola napísaná v programovacom jazyku C (štandard c99), pretože to je programovací jazyk priamo podporovaný operačným systémom s vynikajúcou náväznosťou na hardvér zariadení podporovaných OS Maemo a množstvom dostupných API.

1.3.2 Vývojové prostredie Code Blocks

Vývoj aplikácie prebiehal vo vývojovom prostredí Code::Blocks[4] vo verzii 8.02. Je to multi platformné vývojové prostredie vydané pod slobodnou licenciou rozšíriteľné množstvom zásuvných modulov.

1.3.3 Ostatné nástroje

Na kompiláciu programu bol použitý kompiler GCC vo verzii 3. Na debugging a analýzu správy pamäte boli použité nástroje Valgrind a GDB dostupný pod slobodnou licenciou.

3. Od novembra 2009 je na trhu Nokia N900, ostatné modely sa už nevyrábajú.
4. <http://www.codeblocks.org/>

Kapitola 2

Analýza dostupného softvéru

Táto kapitola obsahuje stručnú analýzu vlastností dostupného softvéru na správu vozidiel pre rôzne zariadenia a platformy, či už komerčného alebo nekomerčného. Zameriava sa na rozhodnutia spravené pri tvorbe týchto aplikácií a ich implikácie v kontexte používateľskej prívetivosti. Hlavným cieľom tejto kapitoly je lokalizácia a pochopenie problémových oblastí tohto typu softvéru s ich následnou elimináciou pri tvorbe vlastnej aplikácie.

2.1 AutoBase

Aplikáciu AutoBase [1] vytvoril v roku 1999 pán Herb Otto a jej vývoj pokračoval až do roku 2005, kedy sa zastavil. Bola vytvorená pre platformu Palm OS a je primárne určená na monitorovanie používania osobného alebo iného automobilu.

Užívateľské rozhranie a spôsob práce s aplikáciou je prispôsobený operačnému systému a druhu zariadenia, pre ktorý je aplikácia určená. Keďže Palm OS, pre ktorý je aplikácia určená, nepodporuje farby a pracuje iba v čiernobielom režime, spolu s faktom, že rozlíšenie zariadení je relatívne malé, sa prejavili na skutočnosti, že všetky aplikačné okná sú bez grafických prvkov a počet ovládacích prvkov je minimálny.

Hlavné okno programu obsahuje iba zoznam zadaných udalostí, tlačidlo na pridanie novej udalosti a poskytuje možnosť ich filtrovania spolu s výberom auta. Ďalšie operácie s užívateľskými dátami ako mazanie a editovanie sú prístupné až pri podrobnom zobrazení udalostí alebo áut. Základná funkcionalita aplikácie spočíva v možnosti zadania 6 rôznych druhov udalostí pre špecifikovaný automobil. Užívateľ si vyberá typ udalosti a pre každú z nich je definovaná skupina parametrov, ktoré ju popisujú. Prvky užívateľského rozhrania sú prispôsobené práve na túto činnosť, pričom ďalšia funkcionalita vyžaduje viac interakcie používateľa. Aplikácia je schopná spracovať 15 áut a udalosti s nimi spojené. Hardvérová platforma podmienila aj fakt, že je to mono užívateľský softvér bez možnosti zabezpečovať prístup k dátam heslom, keďže jestvuje predpoklad, že zariadenie a teda aj aplikácia je využívaná práve jedným užívateľom.

Medzi pokročilejšie funkcie patrí export dát do textového súboru, avšak absentuje možnosť vyhľadávania, čo pri väčšom počte zadaných udalostí robí prácu s aplikáciou nepríjemnú. Celková koncepcia platformy však bola dodržaná a AutoBase vyniká svojou jednoduchosťou používania.

---

1. <http://herbo.dyndns.org:1111/autobase.html>

Obrázok 2.1: Hlavné okno programu AutoBase

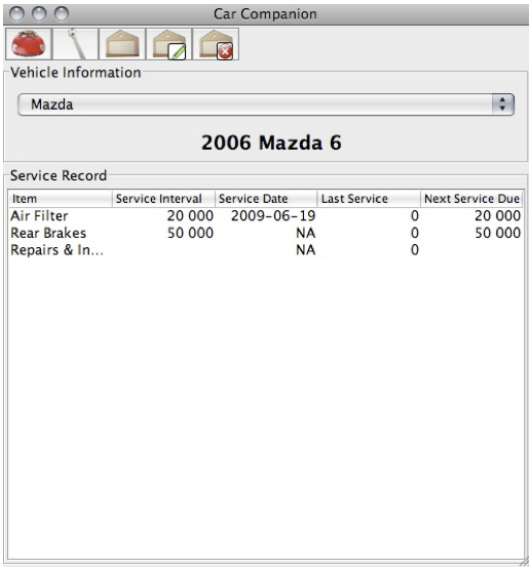
2.2 Car Companion

Aplikácia Car Companion [2] je určená pre osobné počítače a slúži na spravovanie záznamov o opravách vozidla a plánovaných servisných úkonoch týkajúcich sa automobilu. Car Companion je softvér naprogramovaný v jazyku Java s otvorenou licenciou a je použiteľný na majorite dostupných počítačových platforiem.

Užívateľské rozhranie aplikácie je strohé, vizuálne nepríťažlivé a nevyužíva možnosti platformy, pre ktorú je vytvorená. Základné okno aplikácie obsahuje tlačidlá na pridávanie, mazanie a editovanie jednotlivých záznamov, ponúka možnosť výberu automobilu a zobrazuje jednotlivé záznamy v tabuľke. Tlačidlá neobsahujú textové popisky, ale obrázky, ktoré nepripomínajú štandardné symboly, čo spôsobuje, že ovládanie aplikácie je neintuitívne a užívateľ potrebuje premýšľať nad spôsobom práce so softvérom, čo je jeho výrazná slabina. Car Companion je lokalizovaný iba do jedného jazyka a užívateľské prostredie svojou koncepciou neplní dôležitú funkciu a tou je spríjemňovať užívateľovi prácu s aplikáciou. Nevýhodou tejto aplikácie je aj fakt, že pre každý typ záznamu poskytuje rovnaké parametre a týmto nereálne a strnulo simuluje status quo týchto úkonov. Spôsob práce je veľmi priamočiary, užívateľ pre vybrané auto zadá žiadaný servisný úkon a vyplní potrebné údaje. Pokročilejšia funkcionalita v aplikácii prakticky absentuje, vyhľadávanie nie je prítomné, export a import údajov nie je v Car Companion implementovaný. Jediná možnosť, ako vyextrahovať dáta zadané do tejto aplikácie, je vytlačiť ich na papier, čo prakticky odstraňuje možnosť znovu použitia už zadaných dát iným softvérom.

Car Companion je príkladom aplikácie, ktorá aj napriek jednoduchosti riešeného problému nevyužíva potenciál platformy a pri analýze a návrhu softvéru poskytuje cennú ukážku toho, ako by kvalitná aplikácia nemala vyzerať a čo sú podstatné vlastnosti monitorovacej aplikácie.

2. <http://www.macupdate.com/info.php/id/13933>



Obrázok 2.2: Hlavné okno programu Car Companion

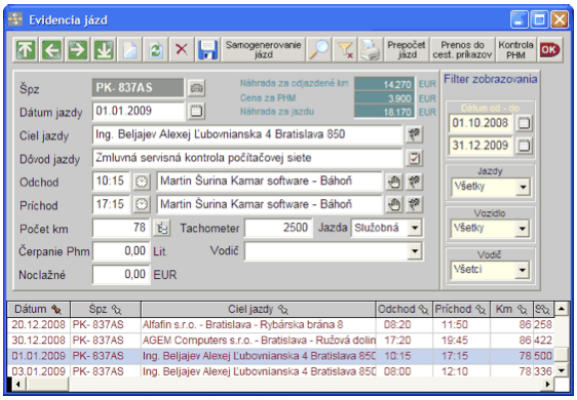
2.3 Kniha jázd Speedy

Aplikácia Kniha jázd Speedy [3] je komerčný softvér vytvorený spoločnosťou Kamar určený pre operačný systém Windows a Linux a poskytuje mohutný nástroj na komplexnú správu firemného vozidla.

Funkcionalita aplikácie je rozsiahla a spravovanie vozidla je v nej zachytené z množstva rôznych prístupov. Aplikácia je multi užívateľská a umožňuje autentizáciu heslom. Podpora rôznych automobilov je samozrejmosťou, pre každý z nich je možné nastaviť rad parametrov do detailov a poskytuje aj široké možnosti uloženia dát pre jednotlivé udalosti. Hlavný zámer aplikácie je poskytnutie kompletnej správy automobilu vo firme a obsahuje niekoľko zaujímavých funkcií, ktoré túto činnosť uľahčujú, ako napr. samogenerovanie jednotlivých jázd na základe zadaných kritérií, vyúčtovania pracovných ciest a rôznych nákladov na prevádzku automobilu spojených s možnosťou tlače a exportu do rôznych formátov, napojenie na GPS záznamník a sledovanie podrobnej prevádzky a nákladov, zakomponovanie výstupov do účtovnej agendy spoločnosti.

Rozdiel oproti predchádzajúcim aplikáciám je zrejmý, či už z množstva funkcií, alebo aj z užívateľského prostredia, ktoré je kvôli tomu preplnené ovládacími prvkami a na orientáciu v ňom je potrebná istá doba. Užívateľská neprívetivosť je hlavný nedostatok tejto aplikácie, ktorá ale pri správnom používaní bude vhodným doplnkom softvéru používaného vo firme.

3. <http://www.kamar.sk/s_speedy.html>



Obrázok 2.3: Hlavné okno programu Speedy

2.4 Car Maintenance and Fuel Economy

Aplikácia Car Maintenance and Fuel Economy[4] je komerčný softvér vo verzii 2.3 vyvíjaný spoločnosťou BlueKatana určený pre prístroje s operačným systémom Windows Mobile. Cieľová platforma s jej možnosťami sa podobá platforme Maemo najviac zo všetkých analyzovaných aplikácií, preto poznatky získané z tejto analýzy dobre poslúžia pri návrhu programu Karavana

Aplikácia funguje na prinípe pridávania udalostí pre vybraté vozidlo, ktoré musí byť definované aspoň jedno, inak aplikácia zobrazí chybové hlásenie a poskytne možnosť zadania vozidla. Grafické rozhranie aplikácie je tomuto cieľu prispôsobené a je rozdelené na dve logické časti. Prvá časť, ktorá zaberá väčšinu plochy obrazovky, je určená na zadávanie udalosti a zobrazuje pre túto činnosť potrebné prvky. Druhá časť plochy je určená na zobrazenie už zadaných udalostí v tabuľke. Všetky ďalšie akcie súvisiace s udalosťami alebo autami, ktoré aplikácia poskytuje, sú prístupné z menu.

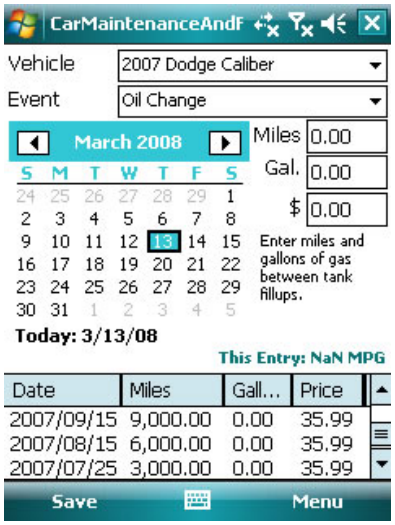
Medzi pokročilú funkcionalitu zaraďujeme možnosť zobrazenia odporúčaných udalostí, kedy aplikácia na základe už zadaných dát odhadne, či nie je potrebá servisná prehliadka a podobne. Aplikácia pracuje spôsobom, že si pamätá posledné vybrané auto a zobrazené udalosti sa vzťahujú pre toto, rovnako ako aj manipulácia s udalosťami.

Celé grafické rozhranie je vytvorené použitím štandardných prvkov, ktoré používa daná platforma a dobre vizuálne zapadá medzi ostatné nainštalované aplikácie. To, že sa autori nerozhodli používať vlastné ovládacie prvky považujeme za výhodu, ktorá uľahčí užívateľovi prácu s aplikáciou. Rozdelenie hlavného okna na spomínané dve časti nepovažumeje za vhodné riešenie, pretože vzhľadom na veľkosť obrazovky sa zámer zobraziť zadané udalosti minul účinku a potreba posúvať sa v tabuľkovom zobrazení pri hľadaní udalosti, keďže tá zobrazí iba 3 udalosti naraz, je pre užívateľa obťažujúca. Rovnako rozhodnutie umiestniť kalendár na hlavné okno považujeme za nevhodné, pretože obsadil cen-

4. <http://www.bluekatana.com/frmStatMobileCarMaintenanceandFuelEconomy.aspx>

né miesto, ktoré by sa dalo využiť napríklad pre tabuľku. Naopak počet rôznych typov udalostí a spomínane odporúčané udalosti poukazujú na robustnosť a vyspelosť aplikácie.

Celkovo v nás tento program aj napriek nevhodnému rozdeleniu hlavného okna zanechal dobrý pocit a reprezentuje kvalitne a používateľsky vhodne vytvorený softvér.



Obrázok 2.4: Hlavné okno programu Car Maintenence and Fuel Economy

Kapitola 3

Technológie

Ak sú ciele modernej aplikácie konkurencieschopnosť, rozšíriteľnosť do budúcna a užívateľská prístupnosť, mala by byť založená na technológiách poskytujúcich pevné a overené základy.

Začiatok kapitoly obsahuje obecný popis mobilných platforiem a špecifiká Maema. Táto kapitola predstavuje technológie a koncepty použité pri programovaní mobilnej aplikácie Karavana a stručne opisuje výhody plynúce z ich použitia, respektíve dôvody pre a proti ich použitiu.

3.1 Mobilné platformy

Svet mobilných platforiem by sa dal charakterizovať ako podmnožina sveta osobných počítačov. Konkrétne sú na trhu rôzne prístroje s Windows Mobile OS, s Android OS, s Blackberry OS a iPhone[1]. Smartphony[2] alebo PDA prístroje sú vo svojej podstate počítače a väčšinu komponentov majú spoločných. Avšak vzhľadom na zameranie prístrojov a na fakt, že sú to príručné zariadenia, sú hardvérove komponenty prístrojov špeciálne zvolené. Pri návrhu sa dizajnéri týchto prístrojov zameriavajú na viacero oblastí. Výber komponentov je podriadený hlavne spotrebe energie, často sú menej výkonné ako ich počítačové ekvivalenty. Pracujú v režimoch obmedzenejších ako podobné komponenty v iných systémoch[3], svoju prácu prispôsobujú špecifikám platformy[4]. Tieto ich charakteristiky majú na programovanie pre mobilnú platformu významný vplyv.

Z hľadiska programovania ako manuálnej činnosti nie je reálne na žiadnej mobilnej platforme programovať priamo[5] a vývojári sú nútení používať rôzne špecifické emulátory a vývojové prostredia. Z pohľadu programovania ako duševnej činnosti musia vývojári prispôsobovať jadrá programov konkrétnym podmienkam danej platformy. Aby bol program užívateľsky prívetivý, nemôže napr. vykonávať intenzívne výpočty, ktoré zabránia použitiu prístroja na niekoľko minút alebo výrazne spotrebúvajú energiu batérie[6]. Grafické prostredie jednotlivých aplikácií musí byť prispôsobené faktu, že vstup užívateľských dát je

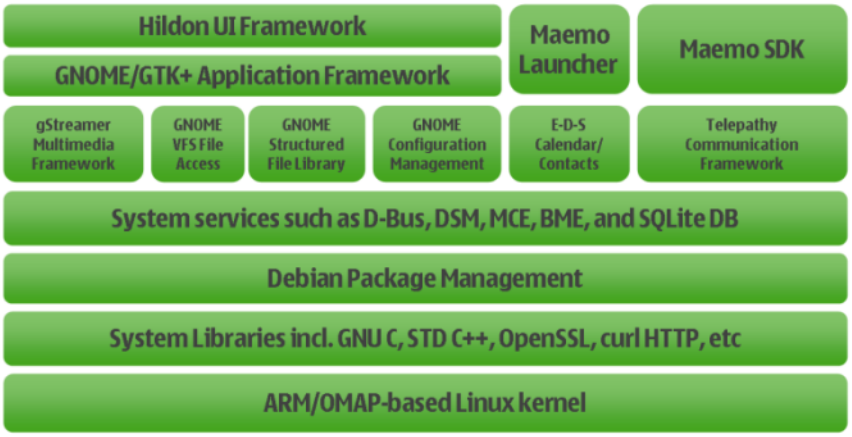
1. Tieto prístroje majú majoritný podiel na trhu, existuju aj ďalšie.
2. http://en.wikipedia.org/wiki/Smartphone
3. Cpu nemá taku obsiahlu inštrukčnú sadu napr. pre podporu starších verzií kódu i686.
4. Napríklad majú nútený úsporný režim.
5. Dôvody sú napríklad dlhý čas kompilácie, nevhodné vstupno-výstupné zariadenie, nedostupnosť príslušných aplikácií.
6. Na osobnom počítači môže vývojár využiť vlákna alebo nechať aplikáciu počítať.

sťažený, rozlíšenie displayov je výrazne zmenšené, užívateľ ocení jednoduchú navigáciu v aplikácii a pod. Všetky tieto obmedzenia sa odrážajú na návrhu a implementácii aplikácie.

3.2 Maemo

Maemo[2] je softvérová platforma vyvíjaná spoločnosťou Nokia pre smartphony a internetové tablety, ktoré zároveň produkuje. Operačný systém Maemo je založený na Linuxe pre procesory typu ARM. Veľká väčšina kódu[7]Maema je otvorená, a touto stratégiou sa líši od ďalších telefónov alebo zariadení založených na Linuxe, pretože sú často uzavreté a vyžadujú špeciálne vývojové nástroje. Filozofiou Maema je umožniť používateľom ľahkú modifikovateľnosť a rozšíriteľnosť samotnej platformy a Nokia na tomto procese aktívne spolupracuje napr. poskytovaním priestoru pre rôzne projekty.

Architektúra Maema vyššej úrovne z pohľadu vývojára pozostáva z niekoľkých komponentov[2]. Jadro systému, založené na ARM odnoži, je vo verzii 2.6 a poskytuje možnosť dynamického zavádzania modulov za behu. Základná sada programov obsiahnutých v štandardej inštalácii sa zhoduje s operačným systémom Debian Linux, avšak niektoré programy boli optimalizované a rozšírené, aby zodpovedali napr. napájacím špecifikám a vstupným operáciám pomocou dotykovej obrazovky. Ako nástroje príkazového riadku bol použitý BusyBox[8], keďže bol vytvorený s dôrazom na malú veľkosť. Správu užívateľského prostredia zabezpečuje GTK+ a ako balíčkovací systém je použitý nástroj dpkg[9].

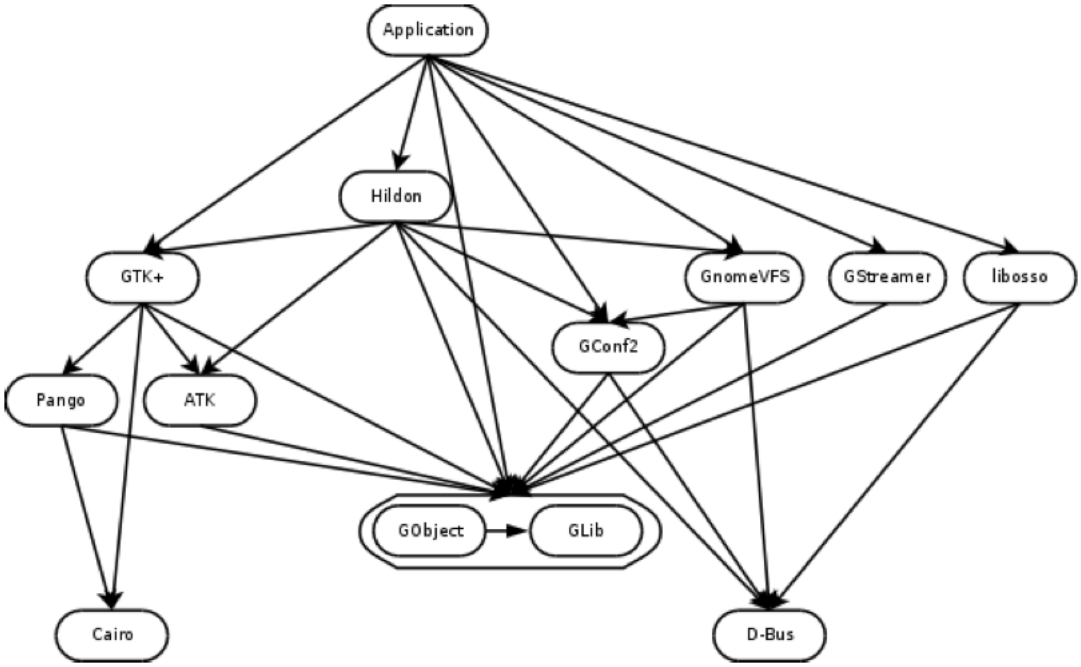


Obrázok 3.1: Vizualizácia softvérovej architektúry Maema, zdroj: <http://maemo4beginners.garage.maemo.org/maemo-quick-start-guide.pdf>

7. Približne 90% kódu.
8. <http://www.busybox.net>
9. <http://www.debian.org/doc/FAQ/ch-pkgtools.en.html>

3.3 API

Aplikačný framework pre platformu je rozsiahly a mal by pokryť bežné nároky programátora pri tvorbe aplikácií a ich jednotlivých vlastností. Na obrázku je naznačené prepojenie jednotlivých komponentov aplikačného frameworku z pohľadu úrovne abstrakcie.



Obrázok 3.2: Prepojenie jednotlivých prvkov API, zdroj: <http://maemo4beginners.garage.maemo.org/maemo-quick-start-guide.pdf>

3.3.1 Hildon

Základnou odlišnosťou Maema od bežnej aplikácie pre platformu Linux je komplexný aplikačný framework Hildon[1] vo verzii 2.0.4, ktorý uľahčuje programátorom vývoj ich aplikácií pre mobilný telefón tým, že poskytuje štadardnú štruktúru aplikácie. Hildon[2] je čiastočne postavený na Gnome frameworku s dôrazom na komponenty typu GTK+ a preto sa aj jeho programovacie API podobá tomu použitému v Gnome frameworku. Hildon oproti GTK+ obsahuje niekoľko úprav a vylepšení s ohľadom na fyzické vlastnosti zariadení.

Hildon obsahuje:

- sada Hildon widgetov pre užívateľské prostredie
- server obrázkov a tém Sapwood
- správca nastavení a úloh, stavový panel

- podpora mobility pre aplikácie (napr. automatické uchovávanie stavu, správca okien) a iné

Štandardom pri programovaní aplikácie pre Maemo je použitie HildonProgram[8] objektu[10] na reprezentáciu bežiacej aplikácie. HildonWindow[8] je štandardný objekt, ktorý predstavuje okno v GUI na najvyššej úrovni. tieto dva objekty majú svoje ekvivalentné objekty v GTK+ frameworku. Keďže mobilný telefón je špecifické zariadenie, boli HildonProgram aj HildonWindow modifikované tak, aby vyhovovali jeho požiadavkám a preto by každá bežiaca Maemo aplikácia mala obsahovať práve jednu inštanciu týchto objektov. Špecifickou časťou knižnice objektov Hildon je Hildon file management knižnica[7], ktorá obsahuje objekty na prácu so súbormi, na ich otváranie a zatváranie, prípadne na zobrazenie informácií o nich. Framework Hildon obsahuje aj ďalšie objekty, ktoré majú svoj GTK+ ekvivalent, ale museli byť modifikované, ako napríklad HildonColorChooser alebo HildonVolumeBar. Všetky modifikácie boli prevedené za účelom lepšej integrácie so zariadením alebo jeho unifikovaným vzhľadom a mali by byť použité namiesto GTK+ objektov.

3.3.2 GObject a GLib

Základnými nízkoúrovňovými knižnicami sú GObject[4] (verzia 2.12.12) a GLib[4] (2.12.12) poskytujúce systém manažmentu objektov. GObject obsahuje typový systém, systém predávania správ a systém tried a rozhraní. GLib je systémová knižnica nástrojov obsahujúca na elementárnej úrovni náhradu dátových typov štandardnej C knižnice a rozširujúca ich o spojené zoznamy, hashovacie tabuľky, rôzne typy n-árnych stromov, dynamické polia, stringy a pod. Poskytuje základnú podporu aplikáciám na prácu s vláknami a ich komunikáciou, pamäťovými poliami, výstupnými, chybovými správami a prácu s pamäťou.

3.3.3 GConf2

Knižnica GConf2[4] (verzia 2.16) obsahuje objekty, ktoré sa využívajú pri ukladaní a načítavaní užívateľských preferencií a nastavení aplikácie. GConf2 funguje ako špecifický druh databázy, kde je na pozadí bežiaci gconf daemon. Tento daemon poskytuje prepojenie medzi databázou, kde sú uložené samotné nastavenia aplikácie, a aplikáciou, ktorá ich potrebuje načítať alebo uložiť. Objekt GConfClient je navrhnutý pre využitie v aplikácii a pomocou neho sa pracuje s uloženými dátami. Samotné dáta sú uložené v pároch kľúč a hodnota, pričom hodnotou môže byť reťazec znakov, celé alebo desatiné číslo, boolean hodnota alebo schéma, čo je zložitejší kľúč využitý na popis hodnoty. Na prístup k hodnote sa využíva stromová štruktúra podobná súborovému systému používanému v systémoch Unix a to napr. /apps/karavana/filename. Použitie GConf2 objektov je výhodné preto, lebo poskytuje programátorovi jednoduchý a unifikovaný mechanizmus ukladania nastavení a väčšinu problémov s tým súvisiacich rieši priamo svojou architektúrou (napríklad zamykanie nasta-

10. V následujúcom texte používame slovo objekt ako označenie častí API, nie je však ekvivalentný významu objektu v kontexte objektovo orientovaného programovania.

vení, aby ich nemohla zmeniť iná aplikácia kým s nimi pracujete, mechanizmus notifikácií zmien alebo počiatočné predspracovanie hodnôt nastavení).

3.3.4 GTK+

Aplikačný framework GTK+[5] (verzia 2.10.12) zabezpečuje prezentáciu aplikácie užívateľovi a jeho pomocou konštruujeme grafické užívateľské rozhranie. GTK+ poskytuje objekty reprezentujúce rôzne grafické elementy, ktoré sú využívané v GUI a mechanizmy na jeho ovládanie, obecne nazývané GtkWidget. Patria tu napríklad rôzne typy tlačidiel, výberových boxov, plôch na zadávanie textov, objektov na tvorbu menu. Podstatnou časťou GTK+ sú objekty slúžiace na zobrazenie užívateľských dát, vytvorené pomocou MVC konceptu[11]. GtkTreeView[6] reprezentuje View a slúžii na renderovanie dát, GtkTreeStore alebo GtkListStore[12] prestavuje model kde sú uložené dáta a kontroler je vnútorne ukrytý v týchto dvoch častiach. Výhoda tohto konceptu je možnosť viacerých typov zobrazenia pre jednu inštanciu dát, čo šetrí využívanú pamäť ako aj zmenšuje logiku nutnú na implementovanie programátorom.

Špecifickou časťou GTK+ sú tzv. layout kontainery, ktoré kontrolujú umiestnenie všetkých ostatných GtkWidget objektov na výstupnom zariadení a vytvárajú celkový look-and-feel[13] aplikácie. Rozdeľujú sa na fixné, ktoré umiestňujú elementy na presne zadanú pozíciu a dynamické, ktoré sa prispôsobujú dostupnej veľkosti. Vhodnejšie pre aplikáciu sú dynamické, pretože je ťažké odhadnúť presné pozície a veľkosti jednotlivých elementov a jednotlivé zariadenia využívajúce Maemo majú odlišné veľkosti displayov a rôzne rozlíšenia.

Každý objekt po užívateľovej interakcii emituje príslušný signál (signály jednotlivých objektov sú popísané v referenčnej príručke napr. tlačidlo po kliku myšou vytvorí "clicked" signál). Na zachytenie týchto signálov slúžia tzv. callback funkcie, ktoré sú priradené jednotlivým elementom v GUI podľa typu emitovaného signálu a telo funkcie obsahuje logiku reagujúcu na signál. Týmto spôsobom sa vytvára aplikačná logika, ktorá prepája jednotlivé elementy GUI.

3.3.5 Ďalšie časti api

Aplikačný framework Maemo[1] obsahuje aj nasledujúce časti:

- Cairo, GDK, Pango sú frameworky zabezpečujúce nízkoúrovňové vykresľovanie aplikačných okien, prácu s vektorovou a rastrovou grafikou a prácu s písmami
- GDK pixbuf umožňuje načítavanie a prácu s rôznymi formátmi obrázkov
- GStreamer poskytuje komponenty na prácu s multimédiami ako prehrávanie, editovanie a nahrávanie audia a videa

11. Model-View-Controler, model, view a kontroler sú vzájomne oddelené a každý zabezpečuje inú časť celkového výsledku <http://en.wikipedia.org/wiki/Model--view--controller>.
12. Ďalšie sú GtkTreeModelFilter, GtkTreeModelSort, ale menované sú najpoužívaniejšie.
13. <http://en.wikipedia.org/wiki/Look_and_feel>

- GnomeVFS vytvára abstraktnú vrstvu pre prácu so súbormi medzi aplikáciou a úložiskom dát
- libosso a D-Bus poskytujú možnosti komunikácie medzi jednotlivými bežiacimi procesmi a informačný kanál medzi zariadením a aplikáciou

3.4 Vývojové prostredie a cross-kompilácia

Fakt, že mnohé použité komponenty v zariadeniach sú výkonovo slabšie ako ich ekvivalenty v osobných počítačoch, spolu so skutočnosťou, že časť softvérového vybavenia je rovnaká ako pre bežné počítače, postavili vývojárov pred závažný problém, a to dĺžku kompilácie. Neustále čakanie, kým kompilátor zostaví natívny kód pripravený na spustenie, predlžovalo vývoj a testovanie softvéru. Spôsob akým sa tento problém rieši sa nazýva cross-kompilácia.

Cross-kompilácia je veľmi jednoduché a elegantné riešenie tohto problému. Jej základom je, že sa vývoj a testovanie softvéru prenesie na hostiteľský počítač, ktorého výkon je rádovo väčší ako výkon mobilného prístroja[2]. Na hostiteľskom počítači pri vývoji beží prostredie emulujúce prostredie mobilnej platformy, pre ktorú vývoj prebieha. Toto zabezpečuje dostupnosť práve tých knižníc a prostriedkov, ktoré sú v reálnom prístroji. Hlavná výhoda cross-kompilácie však je, že samotná kompilácia, čo je proces časovo najnáročnejší počas vývoja, sa výrazne skracuje. Na hostiteľskom počítači sú tak isto dostupné mnohé aplikácie rozširujúce možnosti odstraňovania chýb a ladenia aplikačného kódu. Najvýraznejšie sa táto výhoda prejavuje pri kompilácií rozsiahlych programov ako napríklad okenný manažér, internetový prehliadač.

3.4.1 Scratchbox

Cross-kompilačné a vývojové prostredie pre platformu Maemo[2] je zložené z dvoch oddelených programov. Prvým z nich je Scratchbox[14], čo je vlastne program pre operačný systém Linux zabezpečujúci cross-kompilačné prostredie a slúži ako deliaci element medzi hostiteľským systémom a systémom pre ktorý vyvíjame. Poskytuje nástroje pre vývoj a ladenie programov, poskytuje možnosti na prepojenie vývoja s rôznymi vývojovými prostrediami ako napríklad Eclipse.

Súčasťou Scratchbox-u je aj kompilátor GCC a nastavenia kompilátora pre rôzne špecifické platformy. V prípade Maemo je to nastavenie umožňujúce kompilovanie aplikácie do natívneho kódu procesoru ARM, ktorý je využívaný v zariadeniach obsahujúcich Maemo, ako aj nastavenie pre kompiláciu do natívneho kódu hostiteľskej platformy. Tieto rôzne cieľové kompilačné architektúry sa nazývajú TARGETS a prostredníctvom Scratchboxu nemá programátor problémy s nastavovaním kompilátora pre danú architektúru.

14. <http://www.scratchbox.org/>

3.4.2 SDK

Druhou časťou cross-kompilačného prostredia je SDK[15], čo sú samotné knižnice, ktoré obsahuje operačný systém Maemo a ktoré môže programátor využiť pri tvorbe svojej aplikácie. Tieto sa do Scratchbox-u integrujú a pomocou nástroja dpkg je možné doinštalovať ďalšie programy alebo aktualizovať už nainštalované. SDK osahuje aj uzavreté binárne súbory poskytnuté firmou Nokia.

Pre vývoj je taktisto potrebný špeciálny XServer Xephyr, ktorý zobrazuje grafické užívateľské prostredie emulované Scratchboxom. Z obmedzení programov Xephyr a Qemu vyplýva, že TARGET musí byť nastavený na x86 a až po skončení vývoja môžeme skompilovať výsledný program pre ARM procesor a vytvoriť napr. binárny balíček.

3.4.3 Programovací jazyk

Implicitne podporuje vývojové prostredie pre Maemo programovací jazyk C[10] a obsahuje potrebné balíčky pre kompiláciu zdrojového kódu v Scratchbox-e. Ďalšie podporované programovacie jazyky sú Python a C++, avšak doinštalovanie podporných balíčkov je potrebné pre vývoj a debuggovanie aplikácií napísaných v týchto jazykoch. Konkrétne ide o meta balíky maemo-cplusplus-env a maemo-python-env avšak vytvorené aplikácie nemôžu na nich byť priamo závislé[16].

3.5 Libxml2

Libxml2[17] je sada knižníc pôvodne vytvorených pre používateľské prostredie Gnome využívaných na prácu so súbormi vo formáte Xml. Libxml2 je naprogramovaný v jazyku C, ale postupom času bol rozšírený o možnosti napojenia aj na odlišné programovacie jazyky a momentálne podporuje množstvo programovacích jazykov ako Python, C++ a iné. Knižnica Libxml2 je prenosná medzi rôznymi dostupnými platformami a je použiteľná v operačných systémoch Windows, Linux a Mac OSX. Knižnice obsiahnuté v Libxml2 sú rozdelené do viacerých častí.

3.5.1 Tree a parser

Najstaršia časť tohto súboru nástrojov sa nazýva tree a parser[9]. Parser obsahuje funkcie a štruktúry reprezentujúce Xml súbor v operačnej pamäti. Funguje rovnako ako konkurenčné Xml parsery a umožňuje načítanie Xml súboru a pohyb po jednotlivých uzloch. Xml súbor po načítaní je implementovaný ako strom obsahujúci práve jeden koreňový uzol a skupinu uzlov, ktoré sú jeho potomkami. Uzly na rovnakej úrovni sa aj v xml súbore nachádzajú na rovnakej úrovni a existuje možnosť ľubovoľne ich vnárať.

15. Software development kit.
16. <http://maemo.org/development/documentation/programming_languages/>
17. <http://xmlsoft.org/>

Každý uzol (premenná typu xmlNode) obsahuje informácie o svojom otcovi, o svojom následníkovi, čo je ďalší uzol, o svojom poslednom následníkovi. Tieto dáta sú implementované ako ukazatele do pamäti. Uzol obsahuje aj informácie o svojom mene a svojom obsahu, čo sú potrebné dáta načítane zo súboru a využívané programom a radu ďalších pomocných dát využívaných xml parserom na korektné fungovanie. Načítanie dát zo súboru vyžaduje presné prechádzanie jednotlivých uzlov pomocou ukazateľov v nich uložených a extrahovanie samotných dát, ktoré sú uložené ako premenné typu char *. V aplikácii sa s Xml súborom pracuje pomerne jednoducho. Aplikácia má v pamäti uloženú premennú typu xmlDoc, čo je ukazateľ na xml dokument. Pri načítaní xml súboru zavolá program funkciu xmlReadFile (), ktorej argumenty sú názov súboru, použité kódovanie a doplňujúce možnosti a týmto krokom načíta obsah Xml súboru do pamäte. Funkcia xmlDocGetRootElement () získa koreňový uzol a samotný pohyb po xml štruktúre je prezentovaný kopírovaním ukazateľov. Pri uložení súboru sa používajú funkcie typu xmlSaveFile.

3.5.2 Xmlreader a Xmlwriter

Novšie časti knižnice sú xmlreader a xmlwriter[9]. Tieto sady funkcií umožňujú pohodlnejšiu prácu s xml súbormi a slúžia na načítavanie resp. ukladanie súborov z a na súborový systém. Práca s nimi je podstatne jednoduchšia ako s tree a parserom, keďže potrebné úkony na prácu s Xml súborom zjednotili do skupiny funkcií, ktoré sú jasne definované a nevyžadujú manuálne prechádzanie štruktúr. Na pohyb po načítanom súbore využíva programátor preddefinované funkcie z rodiny xmlTextReaderRead, ktoré poskytujú pohodlný prístup k jednotlivým uzlom a ich hodnotám bez nutnosti využívať ukazatele. Táto vlastnosť výrazne minimalizuje možnosť chyby pri spracúvaní Xml dát, keďže o jednotlivé prepojenia a umiestnenia uzlov sa stará vnútorná logika knižnice a nie programátor. Na vkladanie uzlov do stromu sa používajú funkcie typu xmlTextWriterWrite, ktoré efektívne zapúzdrujú vnútornú reprezentáciu xml stromu a poskytujú rovnaké výhody ako funkcie xmlTextReaderRead.

Knižnica Libxml2 obsahuje taktiež funkcie umožňujúce pokročilú prácu s xml súbormi[9]. Podpora Xml Scheme je nekompletná, Libxml2 avšak podporuje validáciu Xml súborov podľa štandardu DTD, podpora XPath je takisto prítomná, rovnako možnosť stavby Xml stromov pomocou SAX je implementovaná.

3.6 Ukladanie dát

Najpodstatnejšou charakteristikou aplikácie pracujúcej nad užívateľskými dátami, ktorá výrazne ovplyvňuje spôsob, akým je aplikácia implementovaná, je typ a spôsob ukladania dát. Programovací jazyk C bez akýchkoľvek rozšírení podporuje ukladanie do textového súboru. S použitím rôznych rozšírení v podobe knižníc a externých programov s príslušnými API môžeme použiť ako zdroj užívateľských dát Xml súbory alebo databázy a iné.

3.6.1 Textový súbor

Ukladanie dát do textových súborov má dlhú tradíciu a je pevne zviazané s históriou operačného systému Unix, rovnako ako aj s programovacím jazykom C, v ktorom bol Unix implementovaný. Súčasťou takmer každého moderného operačného systém je štandardná knižnica funkcií programovacieho jazyka C. Tento fakt poskytuje možnosť písať multiplatformné aplikácie, ktoré sa správajú rovnako na rôznych systémoch. Textové súbory nie sú výnimka a preto nezáleží, na akom operačnom systéme bude aplikácia bežať, aký systém súborov a aký procesor je dostupný na danej platforme, jej správanie by malo byť konzistentné, pretože štandardná knižnica má definované vonkajšie správanie funkcií v nej obsiahnutých.

Najväčšou výhodou použitia textového súboru ako zdroja dát pre aplikáciu je multiplatformnosť a jednoduchosť návrhu. Na pochopenie princípu ukladania dát postačuje skúmanie dátového súboru, ktorý je prehliadateľný každým textovým editorom a spravidla je takýto súbor jednoduchý. Výraznou výhodou je aj možnosť naprogramovania externej aplikácie, ktorá môže tieto dáta modifikovať, aj pre platformy, ktoré nemajú rôzne rozšíriteľné knižnice, ktorými spracúvavame dáta pri ďalších spôsoboch ukladania alebo možnosť editácie dát pomocou bežného textového editora.

Medzi hlavné nevýhody patrí neobratný spôsob práce s dátami, pretože samotná štandardná knižnica obsahuje iba funkcie na čítanie celého riadku alebo čítanie po znakoch. Implementácia programu musí byť upravená, aby počítala s týmto obmedzením a musí obsahovať netriviálnu logiku, slúžiacu na extrahovanie dát. Výraznou nevýhodou je nemožnosť editácie častí súboru. Táto nevýhoda sa najviac prejaví pri malých zmenách obsiahlych dát, keď je potrebné celý súbor nanovo uložiť do súborového systému, pričom táto operácia je náročná na spracovanie a neefektívna pri častej modifikácii dát.

3.6.2 Xml

Xml súbory majú svoju štruktúru definovanú podľa štandardu vytvoreného organizáciou W3C[18]. Štandard Xml bol vytvorený ako univerzálny štandard na tvorbu a ukladanie dát vo forme prístupnej človeku, pretože sémantika definuje obsah dát, a teoreticky je veľmi vhodný na ukladanie dát typu záznamov o používaní vozidla. Programovací jazyk C neobsahuje žiadne knižnice na prácu s xml dátami, a táto funkcionalita musí byť sprostredkovaná externými knižnicami. Multiplatformnosť programu je tak priamo závislá na dostupnosti xml knižníc na rôznych platformách a pri nutnosti použitia odlišnej knižnice sú vyžadované značné modifikácie aplikačného kódu.

V porovnaní s ukladaním textových súborov nám xml knižnice ponúkajú sadu funkcií, ktoré zjednodušujú načítavanie a ukladanie dát, preto je spracovanie xml dát spravidla jednoduchšie. Ani najdokonalejší xml parser však neodstráni základnú nevýhodu textových súborov, ktorá je prítomná aj pri xml súboroch, a to nutnosť ukladania celého súboru aj pri najmenšej modifikácii dát a z toho plynúcu nutnosť načítania celého xml súboru do operač-

18. <http://www.w3.org/XML/>

nej pamäte, čo je pri mobilných aplikáciach nežiadúca vlastnosť.

Xml súbory však predstavujú veľmi dobrú voľbu pri potrebe exportu a importu dát do a z vonkajšieho sveta. Prakticky pre každý operačný systém sú už vytvorené a v praxi používané Xml parsre, čo nám poskytuje možnosť napísania natívnej aplikácie, ktorá bude schopná korektnej editácie našich dát.

3.6.3 Databázy

Databázy sú od svojho vzniku univerzálnym dátovým skladiskom a sú na ukladanie dát projektované. Z povahy aplikácie a hardvéru, na ktorom aplikácia bude funkčná, vyplýva aj voľba databázového systému. Keďže aplikácia beží na personálnom digitálnom asistentovi, permamentné spojenie s databázovým serverom, kde sú uložené užívateľské dáta, je nerealizovateľná požiadavka a udržiavanie takéhoto spojenia nezanedbateľne vybíja batérie. Užívateľ mnoho krát potrebuje pracovať s aplikáciou aj v situácii, keď je sieť nedostupná, a preto je voľba databázy založenej na systéme klient server zlá.

Existujú však aj databázy, ktoré nepotrebujú na svoj beh dedikovaný server alebo špeciálnu službu systému, ale pracujú nad súborom v súborovom systéme. Nazývajú sa embedded databázy a aplikáciám využívajúcim ich služby poskytujú transparentnú vrstvu emulujúcu reálny databázový server. Ich využitie je ideálne v prípadoch, keď potrebujeme ukladať dáta, ktoré majú byť jednoducho prístupné, potrebujeme aplikovať rôzne kritériá na triedenie týchto dát, modifikácia dát prebieha v malých častiach a nechceme alebo nemôžeme použiť dedikovaný databázový server. Medzi najpoužívanejšie embedded databázy patria SQLite3 a BerkeleyDB1[19]. Obe sú distribuované pod slobodnou licenciou a sú často používané v rade projektov ako napr. Firefox, OpenOffice.

19. <http://en.wikipedia.org/wiki/Embedded_database>

Kapitola 4

Implementácia

V predchádzajúcich dvoch kapitolách sme analyzovali existujúce aplikácie a technológie, zamerali sme sa na ich vlastnosti a z toho plynúce výhody a nevýhody pri ich použití a dopady na celkový výsledok programátorskej práce.

Táto kapitola popisuje proces rozhodovania a vývoja mobilnej aplikácie Karavana a vysvetľuje rozhodnutia použité pri implementácii.

4.1 Základné požiadavky a prvotná analýza

Pred začiatkom vývoja bol jasne daný cieľ, a to vytvorenie aplikácie na evidenciu prevádzkových nákladov vozidla, ktorá bude fungovať na osobnom digitálnom asistente s operačným systémom Maemo. Základné mantinely, v ktorých sa následný vývoj musel udržiavať, sú týmto naznačené, rovnako aj, aké vlastnosti je reálne očakávať od takejto aplikácie. Základom bola jednoduchá manipulácia s evidovanými dátami, či už pri ich vytváraní, mazaní alebo editovaní. Toto je evidentne činnosť, pri ktorej užívateľ strávi maximum času a typicky bude aplikácia používaná iba so zreteľom na tento typ úkonov, takže hlavný dôraz bol, aby manipulácia s dátami bola čo najefektívnejšia, či už z pohľadu časového alebo z hľadiska užívateľskej interakcie.

Z analýzy vyplynul jeden podstatný fakt, ktorý ovplyvnil návrhové rozhodnutia na každej úrovni, a tým bolo, či bude aplikácia koncipovaná pre jedného používateľa alebo bude obsahovať možnosť práce pre rôznych užívateľov. Jednotlivé aplikácie prezentovali oba prístupy, avšak na iných úrovniach zamerania. Tá časť z nich, ktorá by sa dala nazvať jednoduchšou, obsahovala podporu iba pre jedného užívateľa, často aj to nešpecifikovaného. Zámer tohto rozhodnutia je jasný, implementácia možnosti multiužívateľského prostredia je značne netriviálna, aplikácia musí riešiť oddelenie jednotlivých užívateľov, ochranu ich osobných dát, ktoré zadali do systému, znásobenie jednotlivých údajov o ukladaných udalostiach. Z toho plynú problémy, ktoré sa musia riešiť na každej vrstve aplikácie, užívateľské rozhranie je prirodzene komplikovanejšie kvôli nutnosti autentizácie a autorizácie, jednotlivých užívateľov je potrebné škálovať. Na funkčnej úrovni aplikácie to znamená, že je nutné vyriešiť otázky bezpečnosti a navrhnúť model prístupu k údajom jednotlivých užívateľov, napríklad ak používajú jeden automobil, či je vhodné aby navzájom videli udalosti toho druhého spojené s daným vozidlom. Na úrovni uloženia dát sa rozrastá réžia jednotlivých udalostí kvôli nutnosti pamätať si doplnkové informácie a tým sa komplikuje dátový model. Rovnako takúto funkcionalitu poskytujú aplikácie zamerané na komerčný sektor,

ktoré je možné považovať za mohutnejšie a silnejšie nástroje. Významnú rolu pri tomto rozhodnutí zohral aj fakt, že zariadenie, pre ktoré je Karavana tvorená, je príručné, a teda je veľmi pravdepodobné, že bude využívaná iba jedným užívateľom. Záver z tejto stránky analýzy bol, že Karavana bude mono užívateľská[1].

4.2 Používateľské rozhranie

Analýza dostupných programov ďalej ukázala, aké podstatné je, aby bolo užívateľské prostredie jednoduché, ale aj napriek tomu elegantné, ovládanie a práca s aplikáciou konzistentná, pretože na elegancii grafického prostredia si používateľ vytvorí dojem o aplikácii a konzistentnosť spolu s efektívnym spôsobom práce v programe bude ovplyvňovať názor používateľa na aplikáciu. Jednoduchosť aplikácie a jej užívateľského rozhrania závisí od funkcionality, ktorú ponúka a od možností platformy, na ktorej je používaná, ale myslíme si, že aplikácia pre PDA by mala byť čo najjednoduchšia a najtransparentnejšia pre užívateľa.

Dôraz pri návrhu a implementácii bol kladený najmä na jednoduchosť a prívetivosť používania. Počas vývoja sme dodržiavali empiricky overený poznatok inižiniermi spoločnosti Palm, ktorý tvrdí, že ak je nejaká funkcia prístupná po viac ako štyroch kliknutiach stylusom, je zložito implementovaná[2] a je potrebné prerobiť ju. Dôležitý princíp uplatňovaný počas celého životného cyklu vývoja bol K.I.S.S[3], a spolu s predchádzajúcim zaručuje, že aplikácia bude jednoduchá a príjemná na použitie.

4.3 Základná funkcionalita

Hlavný poslaním aplikácie je evidencia údajov o udalostiach, ktoré sú bežné pri používaní vozidla. Aplikácia poskytuje používateľovi možnosť definovať si jeden alebo viac automobilov a pre každý z nich zadať príslušné udalosti, ktoré sú rozdelené do viacerých kategórií. Po analýze existujúceho softvéru a typického používania automobilu sme sa rozhodli predefinovať tieto typy udalostí:

- tankovanie (fuel)
- výlet (trip)
- servis (service)
- výmena oleja (oil change)
- výmena pneumatík (rotate tyre)
- ostatné (misc)

1. Aplikácia si pamätá meno a adresu užívateľa, ale tieto dáta nie sú nijako využívané.
2. <http://www.ted.com/talks/david_pogue_says_simplicity_sells.html>
3. <http://en.wikipedia.org/wiki/K.I.S.S>

Myslíme si, že tieto preddefinované udalosti by mali obsiahnuť typické činnosti spojené s prevádzkou automobilu, a teda nie je potrebné rozširovať ich o ďalšie typy. Ak by užívateľ často zadával udalosť, ktorá nie je medzi preddefinovanými, a teda by bol nútený používať kategóriu ostatné, po jednoduchých úpravách jednotlivých funkcií je možné pridanie novej kategórie, ale vzhľadom na druh aplikácie a spôsob jej využívania je takýto scenár málo pravdepodobný.

Cieľ aplikácie je prostredníctvom evidovaných udalostí poskytnutie vyčerpávajúcich informácií o každodennej prevádzke vozidla, vytvorenie štatistík o spotrebe paliva za jednotlivé obdobia, evidencia jednotlivých ciest vozidlom, spravovanie pridružených nákladov na prevádzku automobilu ako sú poplatky za servisné úkony a pomocou výstupov aplikácie umožniť užívateľovi automobilu optimalizovať svoje návyky, prípadne znižovať náklady na prevádzku vozidla.

4.4 Štruktúra aplikačného kódu

Programovací jazyk použitý na tvorbu programu je jazyk C a z tohto faktu plynie rozdelenie zdrojového kódu do viacerých súborov. Samotný zdrojový kód sa skladá z viacerých hlavičkových a zdrojových súborov, pričom sa toto rozdelenie snaží zachytiť a svojím spôsobom zapúzdriť reálne oddelené bloky funkcionality programu. Ďalším dôvodom pre takéto rozdelenie zdrojového kódu je snaha o jeho sprehľadnenie a ľahšiu udržovateľnosť do budúcna a tendencia vyhýbať sa nevhodným programátorským praktikám ako napríklad špagetový kód[4].

Prvý hlavičkový súbor obsahuje deklarácie funkcií, ktoré majú na starosti ukladanie a manipuláciu užívateľských dát v databáze. Druhý hlavičkový súbor obsahuje funkcie, ktoré považujeme za pokročilejšie a samotné na správu dát nemajú vplyv, rozširujú však funkcionalitu programu. Základné požiadavky a ciele špecifikované vyššie by boli dosiahnuteľné aj bez funkcií v tomto hlavičkovom súbore a preto sú tieto obsiahnuté samostatne v špeciálnom hlavičkovom súbore. Konkrétne ide o funkcie súvisiace s exportom a importom dát. Tretí hlavičkový súbor obsahuje funkcie využívané grafickým užívateľským rozhraním. Štvrtý hlavičkový súbor obsahuje deklarácie štruktúr používaných naprieč aplikáciu. V súbore karavana.c sa nachádza iba funkcia main.

Toto rozdelenie sa snaží určitým spôsobom napodobňovať funkciu rozhrania z jazyka Java, pretože zmena spôsobu ukladania užívateľských dát (napríklad zmena databázy z Sqlite3 na BerkleyDB a pod.) neovplyvní funkcionalitu ďalších častí aplikácie, ktoré ju využívajú, pokiaľ budú novoimplementované funkcie prijímať rovnaké argumenty a návratová hodnota bude identická. Vzájomné prepojenie jednotlivých súborov je realizované prostredníctvom konštrukcie include a má hierarchickú štruktúru.

4. <http://en.wikipedia.org/wiki/Spaghetti_code>

4.5 Reprezentácia udalostí a áut

Centrom aplikácie Kniha jázd sú dáta, ktoré užívateľ zadá do programu a následne s nimi pracuje. Preto je pre prehľadnosť nevyhnuté a výhodné, aby sme dáta prenášali medzi jednotlivými časťami programu v abstraktnej forme a implemenované funkcie pre prácu s dátami nemali 13 argumentov.

Keďže aplikácia pracuje s udalosťami a autami, ktoré užívateľ pridáva, modifikuje a odstraňuje, v hlavičkovom súbore karavana_struct.h sú definované dve štruktúry Car a Event, ktoré ich reprezentujú.

```
typedef struct _Car Car;
struct _Car
{
   gchar *maker, *model, *plate;
   glong periodicService, oilChange, mileAtStart;
};
```

Štruktúrou Car je reprezentované auto a obsahuje výrobcu, model vozidla, poznávaciu značku, ktorá je pre každé auto unikátna a technické informácie o aute ako periódu výmeny oleja, servisnej prehliadky a najazdené kilometre v dobe pridania auta do aplikácie.

```
typedef struct _Event Event;
struct _Event
{
   gchar *type, *location, *comment, *plate;
   glong startDate, startTime;
   glong endDate, endTime;
   glong mileage, destination;
   gdouble price, volume;
   gint scheduled, event_id;
};
```

Štruktúrou Event je reprezentovaná udalosť súvisiaca s automobilom a má nasledujúce položky: typ, cieľ, komentár, poznávaciu značku auta, ktorému je štruktúra priradená, dátum a čas udalosti (začiatok a koniec, ktorý je nepovinný), najazdené kilometre na začiatku a kilometre na konci, cenu a objem, príznak či udalosť bola plánovaná a unikátny identifikátor udalosti.

Nie každá preddefinovaná udalosť obsahuje všetky položky, napríklad výlet nemusí obsahovať objem a podobne, ale položky typ, dátum a čas (začiatočný), cieľ, počty kilometrov a poznávacia značka sú povinné.

4.6 Ukladanie dát

V 3. kapitole sme analyzovali možné metódy ukladania dát a výhody z nich plynúce. Keďže užívateľ môže zadať a používať viac automobilov (napríklad jeden služobný a jeden súkromný), tak bude v tomto prípade aplikácia obsahovať záznamy súvisiace s oboma autami.

Na základe používania nevieme predpokladať, ktoré auto bude využívané pri nasledujúcom zadávaní a pri štarte aplikácie chceme mať načítané aktuálne dáta. Preto je textový súbor s pomiešanými záznamami jednotlivých áut nevhodný spôsob ukladania. Riešenie tohto problému je napríklad v rozdelení záznamov pre jednotlivé autá do samostatných súborov, avšak musíme vyriešiť problém jedinečnosti mien súborov a tieto si uchovávať v konfiguračnom súbore, aby sme ich vedeli načítať. Nerieši to však problém vyhľadávania, kedy musíme ručne prechádzať súbor a hľadať príslušné záznamy.

4.6.1 Embedded databáza

Problém vyhľadávania podľa zadaných parametrov, ukladania viacerých áut do jedného súboru, udržiavania konzistencie dát veľmi elegantne rieši embedded databáza. Samotné tabuľky sú uložené v jednom súbore a môžeme jednoducho rozširovať rozsah dát, pretože databáza poskytuje prostredníctvom jazyka SQL abstraktnú vrstvu nad týmito dátami.

Ako konkrétnu databázu sme zvolili Sqlite3, pretože je to predvolená databáza na platforme maemo (je nainštalovaná na každom zariadení), je vydaná pod slobodnou licenciou, je minimalistická[5] a jej použitie je jednoduché.

4.6.2 Práca s Sqlite3

V jednotlivých funkciách pre ukladanie dát potrebujeme premennú typu sqlite3, ktorá reprezentuje spojenie na databázu. Toto spojenie je vytvorené prostredníctvom funkcie sqlite3_open[12] a po jeho úspešnom vykonaní môžeme pracovať s dátami.

Samotná práca s spočíva na vytvorení tzv. statement premennej sqlite3_stmt, ktorá reprezentuje jednu požiadavku. Následne sú zavolané tri funkcie, ktorých argumentom je pripravený statement a to v poradí sqlite3_prepare_v2, sqlite3_step a sqlite3_finalize. Po úspešnom vykonaní sqlite3_step, ktorá má argument sql príkaz, môžeme prostredníctvom rodiny funkcií sqlite3_collumn pristupovať k jednotlivým načítaným dátam a statement ukončíme volaním funkcie sqlite3_finalize.

Titeto tri funkcie sú zapúzdrené vo funkcii sqlite3_exec[12], ktorá potrebuje ako argument aj callback funkciu, ktorá spracuje načítané riadky. Použitie sqlite3_exec alebo predchádzajucej trojice funkcií vedie k tomu istému výsledku a je iba na ľubovôli programátora, pre ktorú sa rozhodne.

Každá sqlite3 funkcia vracia definovaný návratový kód, ktorý môžeme použiť na overenie správnosti priebehu funkcií.[6] Po skončení práce s databázou musíme uzavrieť spojenie volaním sqlite3_close ().

Sqlite3 podporuje "napojenie" parametrov na sql príkaz, čo zabraňuje použitie sql injection útoku a odstraňuje chyby, ktoré vzniknú, ak požadovaný výraz obsahuje špeciálne znaky[11]. Túto funkionalitu využívajú všetky naprogramované funkcie prostredníctvom stmt_prepare_event_search, stmt_prepare_event_manipulate a stmt_prepare_car, kto-

5. <http://www.sqlite.org/different.html>
6. <http://www.sqlite.org/c3ref/c_abort.html>

ré podľa typu požiadavky pripravia statement na vykonanie.

Niektoré typy modifikácie dát vyžadujú, aby sa vykonalo viac požiadaviek (napríklad zmena evidenčného čísla auta vedie aj k zmene evidenčného čísla udalostí) a ak niektorá bude neúspešná, celá modifikácia musí skončiť. Sqlite3 obsahuje podporu transakcií, ktorými to dosahujeme.

4.6.3 Databázová vrstva aplikácie

Databáza programu Kniha jízd obsahuje dve tabuľky a to tabuľku áut, kde je primárnym kľúčom evidenčné číslo a tabuľku udalostí, kde je primárnym kľúčom unikátny identifikátor. Tabuľky sú prepojené prostredníctvom evidenčného čísla, takže k udalosti si vieme dohľadať príslušný automobil a vice versa.

Pre prácu s databázou sme vytvorili skupinu funkcií, ktoré obsahujú spomenuté napájanie parametrov a rôzne kontroly konzistenice dát (napríklad vymazávanie neexistujúcej udalosti a pod.) a poskytujeme ich iným častiam aplikácie.

- event_create, event_modify, event_delete - slúžia na manipuláciu dát týkajúcich sa udalostí
- car_create, car_modify, car_delete - slúžia na manipuláciu dát týkajúcich sa vozidiel
- con_open, con_close - na správu spojenia s databázou
- tables_drop, tables_create - na vytvorenie a vymazanie tabuliek
- export_cars, export_events, event_data_for_model, event_get_id - slúžia na načítavanie súboru dát z databázy

4.6.4 Export a import dát

Možnosť importovania a exportovania dát často v aplikáciách nie je implementovaná a pritom sú to funkcionality využiteľné a cenné. Napríklad možnosť exportu a následného importu dát poskytuje spôsob na ich zálohovanie ak táto funkcionalita nie je osobitne implementovaná a ak nechceme užívateľa nútiť vstupovať do systému súborov a manuálne kopírovať potrebné súbory. Rovnako pri zvolení vhodného formátu dát poskytuje možnosť prezerať a prípadne editovať užívateľské dáta aj iným softvérom na rôznych platformách. Toto využitie importu a exportu sa výrazne prejaví pri potrebe editovania väčšieho množstva dát na zariadení ako je Maemo, ktoré nie je dokonale prispôsobené na takúto činnosť takže umožní užívateľovi manipuláciu s dátami iným softvérom a spätné vloženie dát do aplikácie. Pre tieto dôvody sme sa rozhodli naimplementovať import a export dát.

Zvolili sme xml formát pre import a export pretože ciele jeho dizajnu boli celková jednoduchosť, všeobecnosť, ľahká čitateľnosť človekom, keďže ide o textový formát, a jednoduchá tvorba aplikácií spracujúcich xml súbory.[7]

7. <http://www.w3.org/TR/REC-xml/#sec-origin-goals>

V 3. kapitole sme popísali knižnicu Libxml2, ktorá poskytuje nástroje na prácu s xml súbormi a ktorú používame na import a export. Pre tieto účeli sme vytvorili dve funkcie xml_import a xml_export.

Funkcia xml_export zabezpečuje export dát uložených v databáze do xml súboru, pričom samotný proces prebieha následovne:

- načítanie príslušných dát z databázy
- vytvorenie premennej xmlDoc reprezentujúcej vytváraný xml súbor v operačnej pamäti
- postupné spracovanie načítaných dát z databázy a tvorba príslušných xml uzlov
- uloženie xml súboru na príslušné miesto v súborovom systéme

Dáta pre export sú načítané funkciami export_cars a export_events, ktoré majú ako jeden zo svojich argumentov jednocestne spojený zoznam[8], kde uložia dáta načítané z databázy. Použitie tohto typu zoznamu je dostatočné, pretože potrebujeme iba jednu iteráciu zoznamom práve jedným smerom. V prípade, že sú príslušné dáta načítané, v cykle volaním funkcie create_car_xml vytvoríme xml uzol obsahujúci informácie o jednom aute a pripojíme ho k tvorenému xml súboru. Rovnaký úkon prevedieme aj s dátami o udalostiach. Po úspešnom vytvorení celého xml súboru uložíme výsledok do súborového systému.

Úlohou funkcie xml_import je import predpripravených dát (pomocou iného softvéru alebo pomocou aplikácie Karavana) presne zadanej formy. Argumenty funkcie sú premenná sqlite3, meno súboru obsahujúceho dáta a príznak, ktorým upresníme, či chceme zmazať celú databázu a pridať záznamy zo súboru alebo doplniť už obsiahnuté dáta.

Súbor načítame pomocou funkcie xmlReadFile[9] z knižnice Libxml2 a následne otvorený súbor predáme funkcii data_retrieve, ktorá pracuje následovne:

- načítanie súboru do pamäte, prechádzanie xml uzlov spolu s kontrolou ich mien
- v prípade, že sa nachádzame v časti súboru obsahujúceho záznamy áut, vytvoríme jednocestne spojený zoznam áut
- v prípade, že sa nachádzame v časti súboru obsahujúceho záznamy udalostí, vytvoríme jednocestne spojený zoznam udalostí

Jednotlivé záznamy áut a udalostí načítame v cykle while, kedže máme xml súbor načítaný v operačnej pamäti a uchovávame si odkazy uzly obsahujúce udalosti a autá. Zápis jednotlivých záznamov do databazy prebieha v rámci jednej transakcie, aby sme zachovali konzistenciu dát a využíva metódy poskytované databázovou vrstvou aplikácie.

8. <http://en.wikipedia.org/wiki/Singly_linked_list>

4.7 Užívateľské rozhranie

Z analýzy existujúcich programov vyplynulo, aké dôležité je kvalitne navrhnuté grafické užívateľské rozhranie, keďže toto zabezpečuje kontakt užívateľa s aplikáciou a jeho prostredníctvom užívateľ posudzuje kvalitu aplikácie.

Pri návrhu GUI sme vychádzali z koncepcie vytvorenej spoločnosťou Apple pre iPhone aplikácie[9]. Apple rozdelil aplikácie do 3 kategórií podľa spôsobu práce s nimi a ich zamerania:

- "productivity applications" - do tejto kategórie patria aplikácie organizujúce a manipulujúce detailné informácie (príklad je aplikácia pre mailovú poštu).
- "utility applications" - aplikácie v tejto kategórii sú využívané na jednoduchý cieľ s minimom užívateľskej interakcie (príklad je aplikácia na zobrazenie informácií o počasí)
- "immersive applications" - aplikácie využívajúce celú obrazovku so zameraním na samotnú prácu s aplikáciou a dátami prostredníctvom netradičného GUI (príkladom sú rôzne hry)

Jednotlivé kategórie sa líšia zameraním aplikácií a z toho vyplýva aj odlišný prístup pri tvrobe GUI. Karavana zjavne patrí do kategórie "productivity application", pretože nezobrazuje iba jednu informáciu ale celý súbor (tj. nepatrí medzi "utility applications"), pracuje s detailnými informáciami o udalostiach a autách, a užívateľ potrebuje jednoduchý a účinný spôsob manipulácie s dátami namiesto vizuálne oslnivého prehliadania dát a GUI (tj. nepatrí medzi "immersive applications").

Aplikácie patriace do kategórie "productivity applications" sú určené na rýchle a jednoduché spracovanie užívateľských dát, práca s nimi nemá byť hrou, ale má byť intuitívna a bez zbytočných rušiacich elementov. Preto je vhodné využitie štandardných GUI elementov poskytovaných aplikačným frameworkom GTK+ namiesto modifikovaných elementov, aby sme docielili unifikované GUI v rámci celej platformy, takže Karavana neobsahuje žiadne vlastné GUI elementy a pracuje iba s už poskytnutými.

4.7.1 Hlavné okno

Z povahy aplikácie vyplýva, že pri štandardnom spustení aplikácie bude užívateľ vyžadovať zobrazenie udalostí, ktoré do aplikácie vložil, aby s nimi mohol následne manipulovať. Zároveň však potrebuje vidieť aj vložené automobily. Vzhľadom na fakt, že rozlíšenie a veľkosť obrazovky sú relatívne malé, zobrazenie všetkých udalostí a áut spolu s ovládacími prvkami v jednom celku by bolo neprehľadné a takýto prístup je nevhodný. Jednotlivé

9. Pretože iPhone je atraktívna a úspešná mobilná platforma a teoretické koncepty sú znovupoužiteľné aj pre iné platformy <http://developer.apple.com/iphone/library/documentation/UserExperience/Conceptual/MobileHIG/DevelopingSoftware/DevelopingSoftware.html>.

skupiny zobrazovaných údajov a ovládacie prvky s nimi spojené potrebujeme rozdeliť do logických celkov a graficky oddeliť. Možné riešenie je zobrazenie udalostí v hlavnom okne aplikácie spolu s ovládacími prvkami a ďalšie časti (napríklad zobrazenie dostupných vozidiel) sprístupňovať vo forme dialógových okien. Jednou nevýhodou tohto riešenia je obmädzenie užívateľa práve na jednu činnosť, napríklad počas pridávania udalosti si nemôže pozrieť, či už udalosť neexistuje. Preto sme sa rozhodli ako základý prvok GUI použiť koncept záložiek, ktorý je často používaný v internetových prehliadačoch. Následne sme mohli dáta a ovládanie aplikácie rozdeliť do súvisiacich celkov a tieto umiestniť na samostatné záložky:

- rýchle zobrazenie udalostí a ich manipulácia (pridávanie, mazanie a modifikácia)
- zobrazenie vozidiel a ich manipulácia (rovnako ako pri udalostiach)
- zobrazenie informácií o užívateľovi a výber aktívneho vozidla
- zobrazenie štatistík o udalostiach
- vyhľadávanie

Objekt typu GtkNotebook[5] predstavuje kontajner najvyššej úrovne v našej aplikácii a obsluhuje zobrazovanie a správu záložiek. Panel na prepínanie záložiek je umiestnený v dolnej časti pomocou funkcie gtk_notebook_set_tab_pos[6], pretože počet žáložiek je 5 a potrebujeme efektívne[10] využiť priestor na obrazovke.

Jednotlivé elementy na záložkách ako tlačidlá, zobrazenia dát, polia na vkladanie a zobrazenie dát a pod. sú umiestnené v tabuľkách s dostatočným počtom riadkov a stĺpcov, aby bolo ovládanie bezproblémové.

Prvá záložka, ktorá je zobrazená po štarte programu, je záložka obsahujúca tzv. rýchle zobrazenie udalostí, pričom maximálny počet udalostí je obmädzený, aby mal užívateľ informácie o posledných pridaných udalostiach ľahko dostupné. V pravej časti sú tri tlačidlá umožňujúce pridávanie, editovanie a mazanie vybraných udalostí.

Druhá záložka obsahujúca vozidlá má rovnakú formu ako prvá a umožňuje užívateľovi prezeranie a modifikáciu všetkých vozidiel, keďže predpokladáme, že ich nebude veľké množstvo, takže práca s nimi bude prehľadná.

Tretia a štvrtá záložka obsahuje dáta o užívateľovi, resp. jednoduché zobrazenie štatistiky pomocou GtkLabel elementov.

Piata záložka slúži na vyhľadávanie a obsahuje niekoľko GtkEntry elementov, kde užívateľ zadá hľadaný výraz a následne prezerá a modifikuje výsledok.

4.7.2 Práca so záložkami

V prípade, že užívateľ chce modifikovať udalosť (auto) alebo pridať novú, potrebujeme vytvoriť a umiestniť elementy, kde zadá príslušné dáta. Použitie dialógu je nevhodné, pretože

10. Aby zaberal čo najmänej miesta a bol ľahko ovládateľný dotykom.

tento je modálny, teda všetky prvky GUI sú neaktívne okrem neho samotného. Ak by sme použili nemodálny dialóg, musíme na každú záložku pridať tlačidlo na vyvolanie dialógu do popredia, čím by sa narušilo rozloženie jednotlivých elementov a je potrebné kontrolovať, či je práve jeden dialóg otvorený. Vytváranie novej záložky je rovnako nevhodné, pretože týmto spôsobom by užívateľ mohol vytvoriť príliš mnoho záložiek a práca s aplikáciou by sa stala neobratnou kvôli veľkosti obrazovky. Vhodným a implementovaným riešením je vo všetkých prípadoch odobratie (ale nie zničenie) záložky, ku ktorej sa vzťahuje akcia a umiestnenie novej záložky s prvkami, ktoré požadujeme na jej miesto. Takto môže užívateľ napr. modifikovať jednu udalosť a zároveň hľadaním na inej záložke kontrolovať, či už daná udalosť neexistuje.

Výmena záložky prebieha následovne:

1 zvýšime si tzv. "floating reference"[6] na menenú záložku prostredníctvom funkcie g_object_ref, aby systém správy pamäte GTK+ automaticky nezničil pôvodný objekt, keďže ho neskôr použijeme

2 vytvoríme si novú tabuľku alebo iný kontajner, ktorý umiestníme na príslušné miesto medzi záložkami

3 po skončení príslušnej práce vykonáme žiadanú akciu a pôvodnú záložku umiestnime na príslušné miesto prostredníctvom gtk_notebook_append, čím sa automaticky zníži "floating reference", dočasnú záložku môžeme odstrániť prostredníctvom gtk_widget_destroy

Tento prístup umožňuje jeden typ práce s udalosťou (autom) v jednom momente, ale nezabraňuje užívateľovi používať iné funkcie aplikácie ako napríklad vyhľadávanie na inej záložke.

4.7.3 Vyhľadávanie

Aby bola práca s udalosťami úplná, keďže prvá záložka poskytuje iba rýchly prístup k najnovším udalostiam, obsahuje aplikácia vyhľadávanie podľa rôznych kritérií. Po analyzovaní údajov, ktoré obsahuje individuálna udalosť sme sa rozhodli, že implementujeme vyhľadávanie podľa miesta, komentára, dátumu, času a typu udalosti. Vyhľadávanie podľa ostatných údajov je zanedbateľné, pretože ide o umelé údaje z pohľadu človeka (napríklad počet kilometrov na začiatku udalosti slúži iba na jej špecifikáciu a bežný užívateľ nie je schopný zapamätať si a rozlišovať udalosti podľa tohto údaju).

Vyhľadávanie poskytuje možnosť špecifikovať jednotlivé hľadané výrazy, a to či sa budú hľadať zároveň, osobitne alebo či sa daný výraz nemá hľadať. "Alebo" hľadanie nijako neindikujeme, iba napíšeme výraz. Ak chceme, aby hľadaný výsledok obsahoval výraz, vyhľadávame vo forme +výraz a ak chceme, aby neobsahoval, použijeme -výraz. V prípade, že je hľadaný výraz viac slovný, nahradíme medzery znakom -, čím naznačíme, že sa má vyhľadávať celý výraz. Táto forma platí pre hľadanie lokácie a komentára.

Vyhľadávanie dátumu a času umožňuje hľadanie špecifickej hodnoty tvaru YYYYMMDD pre dátum a HHMM pre čas alebo rozsahu YYYYMMDD-YYYYMMDD resp. HHMM-HHMM.

Samotnému hľadaniu predchádza analyzovanie výrazu a extrahovanie jednotlivých podvýrazov a ich spracovanie, kedy sa vytvára SQL príkaz so znakom ? na príslušných miestach, aby bolo možné napojenie parametrov. Pri dátume a čase analýza kontroluje, či hľadáme rozsah alebo presnú hodnotu a následne skontroluje, či podvýrazy obsahujú iba číslice. Pri analýze lokácie a komentára najprv rozdelíme výraz na podvýrazy podľa medzery a potom kontrolujeme prítomnosť prefixu + alebo - a nakoniec skontrolujeme, či je podvýraz viac slovný a nahradíme každý znak - v ňom za medzeru. Na tento účel sme vytvorili funkcie validate_and_parse_text a validate_and_parse_datetime, ktoré popísaným spôsobom zanalyzujú hľadaný výraz a podvýrazy rozdelia do jednocestne spojených zoznamov, ktoré vo funkcii search_cb spracujeme a vytvoríme SQL príkaz.

4.7.3.1 GRegex a PCRE

Keďže GLib používaný v Maeme je vo verzii 2.12, nemôžeme na analýzu využívať regulárne výrazy[11], pretože táto funkcionalita bola do knižnice GLib pridaná až vo verzii 2.14[3], ale musíme ručne prechádzať hľadaný výraz. V prípade, že by aplikácia bola projektovaná pre Maemo verzie 5, mohli by sme regulárne výrazy použiť, čo by vyhľadávanie značne zjednodušilo a umožnilo aj kontrolu logickej konzistentnosti dátumu a času, čo v tejto implementácii nerobíme. Rovnako by sme mohli vyhľadávanie dátumu a času prispôsobiť na tvar yXXXX, mXX, dXX (resp. hXX, mXX), kde X naznačuje číslicu. V tomto prípade by užívateľ napríklad pre udalosti v roku 2009 hľadal výraz y2009 namiesto momentálneho 20090000-20100000, čo by prispelo v prívetivosti aplikácie.

Príklad regulárneho výrazu, ktorým zanalyzujeme výraz pre lokáciu alebo komentár, s miernou zmenou vo formáte, kedy viac slovné výrazy naznačíme tým, že ich dáme do úvodzoviek:

```
[+-]?"[\w\s+-]+"|[+-]?[\w+-]+
```

Pre aplikácie využívajúce kódovanie ASCII je možné použitie knižnice PCRE[12], ktorá je dostupná v Maeme a sprostredkuje využitie regulárnych výrazov, avšak nepodarilo sa nám ju použiť pre výrazy obsahujúce iné znaky ako znaky anglického jazyka (kódované štandardne v UTF-8).

4.7.4 View a model

Udalosti, vozidlá a výsledok vyhľadávania je užívateľovy prezentovaný prostredníctvom objektu GtkTreeView a GtkListStore, ktoré sme teoreticky popísali v 3. kapitole. Aplikácia si pri štarte alebo pri špecifických udalostiach načíta z databázy prostredníctvom funkcií export_cars, export_events alebo export_data_for_model pri vyhľadávaní potrebné dáta a v cykle while ich pridá do modelu. Následne vytvorí view, ktorému nastaví model a špecifikuje, ktoré stĺpce zobrazujeme. Pri zmene dát stačí príslušné dáta modifikovať v modely

11. <http://www.pcre.org/>
12. <http://en.wikipedia.org/wiki/Regular_expression>

a zmena sa automaticky prejavý v GUI. Pri výbere riadku poskytuje view funkcie na výber dát z modelu. Na prácu so samotnými dátami v modely používame objekt GtkTreeIter, ktorý odkazuje na jednotlivé riadky.

4.7.5 Aktívne vozidlo

Aplikácia si v nastaveniach uchováva poznávaciu značku aktívneho vozidla a zobrazené udalosti sú viazané s týmto vozidlom. Rovnako pridanie novej udalosti nastaví túto pre aktuálne aktívne auto. Ak chce užívateľ pracovať s udalosťami iného vozidla, musí na 3. záložke nastaviť príslušné auto ako aktívne.

4.8 Lokalizácia, binárny balíček

Pôvodná architektúra aplikácie obsahovala skupinu zdrojových súborov, ktoré sa kompilovali do natívneho kódu pomocou programu make a ručne vytvoreného súboru Makefile. Avšak pri implementácii lokalizácie sme narazili na problém, ktorý sa nám nepodarilo vyriešiť.

Lokalizácia aplikácií sa vytvára pomocou nástroja gettext a funguje tak, že z vytvorených .po súborov obsahujúcich jednotlivé preložené výrazy a ich identifikátory sa pri kompilácii vytvoria príslušné .mo súbory vo formáte, aký potrebuje aplikácia. V Makefile súbore je potrebné špecifikovať cestu k priečinku, kde sú vo filesystéme uložené .mo súbory aplikácií, aby si mohla aplikácia pri behu načítať lokalizované časti slov pomocou premmennej LC_ALL, ktorá je aktuálne nastavená v systéme. Kompilácia programu a zároveň použitie nástroja gettext je však zdokumentované iba pre možnosť, ked sú súbory Makefile vytvárané dynamicky podľa preddefinovaných pravidiel použitím nástrojov autotools[13]. Už vytvorený Makefile sa nám podarilo upraviť a vytvoriť súbory configure.ac a Makefile.in, ktoré sú používané pre tomto automatizovanom procese, avšak následné prepojenie s nástrojom gettext sme nedokázali zrealizovať. Preto aplikácia Karavana nemá naimplementovanéú lokalizáciu, sú však vytvorené príslušné .po súbory pre prípadné doplnenie lokalizácie niekedy v budúcnosti.

Formy distribúcie aplikácie pre platformu Maemo sú rôzne, môžeme užívateľom umožniť inštalovanie zo zdrojových súborov alebo im môžeme poskytnúť binárny balík vo formáte deb, ktorý prostredníctvom nástroja dpkg užívateľ nainštaluje. Druhá možnosť je užívateľsky prístupnejšia, kedže na inštaláciu aplikácie nepotrebuje užívateľ nijaké špecifické znalosti a celý proces za neho spraví spomínaný nástroj. Pre tento dôvod sme vytvorili binárny balík našej aplikácie a umiestnili sme ho na server <http://garage.maemo.org>, kde je k dispozícii verejnosti.

13. <http://www.gnu.org/software/autoconf/manual/autoconf.html> <http://www.lugod.org/presentations/autotools/presentation/autotools.pdf>

Kapitola 5

Záver

Cieľom tejto bakalárskej práce bolo vytvorenie aplikácie pre správu prevádzky automobilu pre mobilné zariadenie Maemo. Od aplikácie sa očakávalo, že bude prispôsobená špecifikám mobilnej platformy, bude obsahovať štandardnú funkcionalitu tohto typu softvéru a že práca s ňou bude užívateľsky príjemná.

V druhej kapitole tejto bakalárskej práce bol stručne analyzovaný dostupný softvér podobného zamerania. Dôraz bol kladený na hľadanie problémových oblastí takejto aplikácie, aby tvorená aplikácia tieto neobsahovala. Čitateľ sa z nej dozvedel, na ktoré aspekty vývoja treba klásť najväčší dôraz, aby vytvorená aplikácia bola čo najlepšia.

Tretia kapitola bola venovaná teoretickej stránke programovania pre mobilnú platformu a čitateľovi stručne predstavila jednotlivé API, ktoré môže počas vývoja použiť. Časť kapitoly pojednávala o špecifických problémoch pri vývoji a možnostiach ich riešenia.

Štvrtá kapitola využila poznatky získané a popísané v predchádzajúcich kapitolách a predstavila praktickú stránku programovania. Výsledkami analýzy softvéru a vlastnosťami jednotlivých technológií zdôvodnila rozhodnutia ovplyvňujúce tvorbu aplikácie. Časť kapitoly bola venovaná reálnym problémom vzniknutým pri tvorbe aplikácie a ich riešeniu.

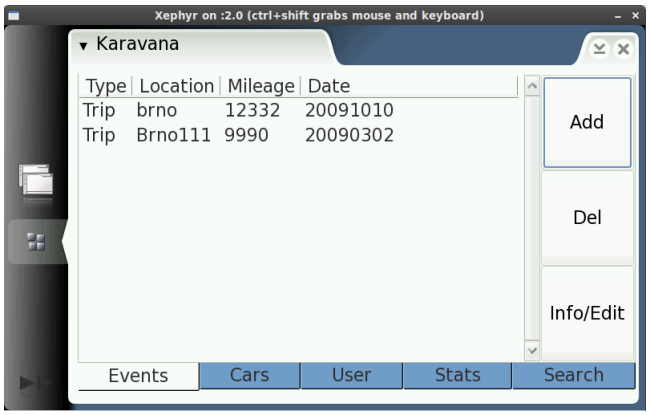
Výsledkom práce je aplikácia Karavana umožňujúca spravovanie vozidla. Počas testovania sa nevyskytli závažné problémy, pridávanie a manipulácia jednotlivých dát bola plynulá a testovanie naznačilo, že množina implementovaných funkcií funguje dobre. Prípadné chyby, ktoré sa počas testovania a vývoja nepodarilo odhaliť budú opravené v nasledujúcich verziách programu.

Proces tvorby aplikácie Karavana ukázal, že aj jednoduchá aplikácia je podložená svedomitou prácou pri analýze a vývoji a aj pri jednoduchých aplikáciach sa často vyskytnú chyby a problémy, ktoré sú omnoho zložitejšie ako samotný softvér. Práca zároveň ukázala, aký dôležitý je vo vývoji softvéru aspekt skúseností a ich rozširovanie bude prebiehať následovným doplňaním programu o ďalšiu funkcionalitu.

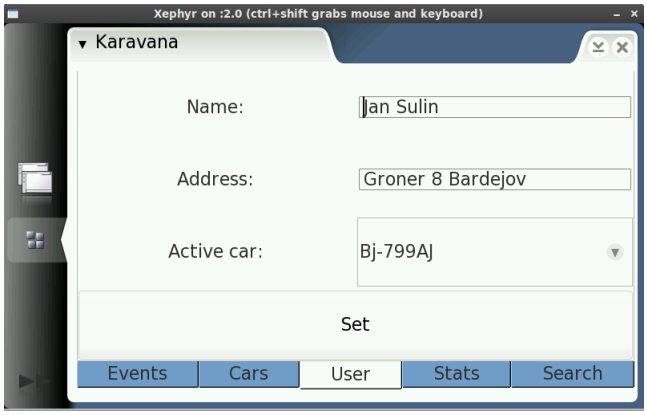
Nedostatkom práce je absencia funkčného prekladu, pričom tento problém môže byť jednoducho vyriešený po získaní skúseností s používaním nástrojov autotools. Aplikácia samotná má veľký potenciál stať sa nápomocnou a v budúcnosti je priestor na jej rozširovanie (napríklad užívateľské definovanie klávesových skratiek, podpora GPS modulu na získanie aktuálnej pozície).

Dodatok A

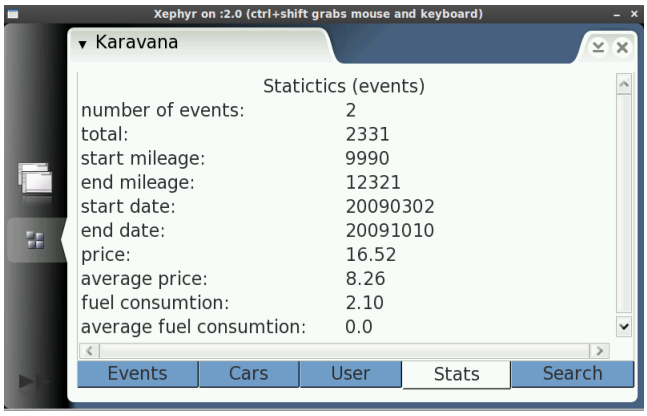
Užívateľské rozhranie aplikácie Karavana



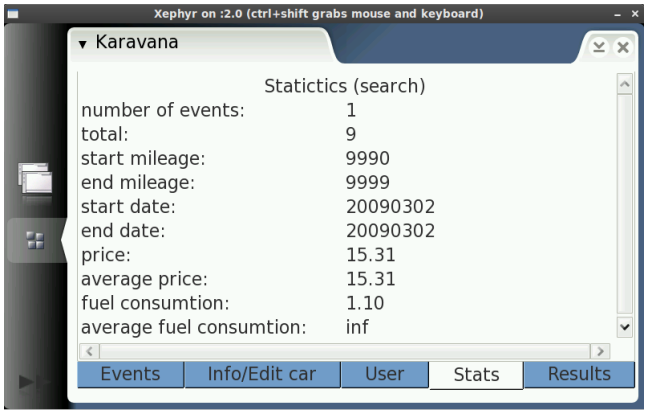
Obrázok A.1: Hlavné okno programu Karavana so zobrazeným rýchlym prehľadom udalostí



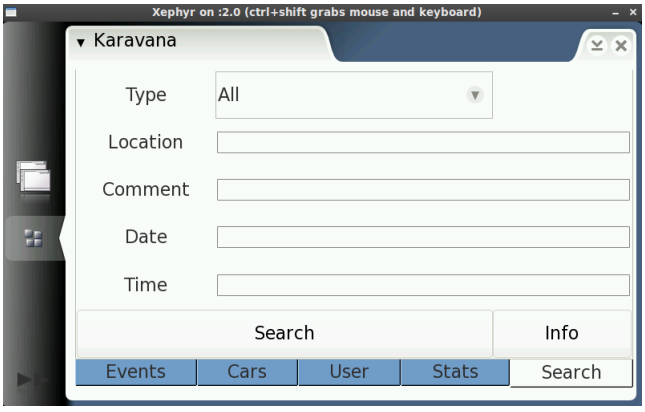
Obrázok A.2: Záložka užívateľ s načítanými dátami



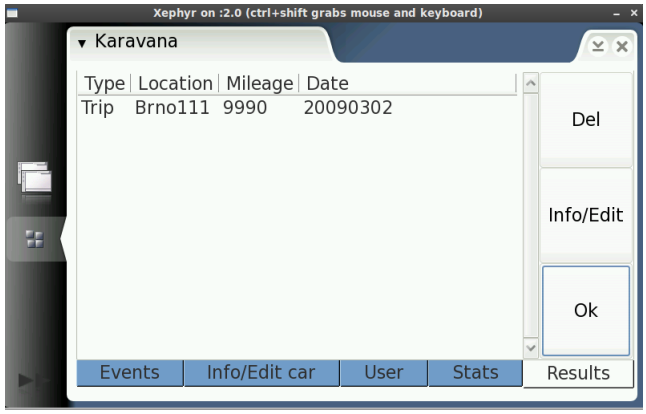
Obrázok A.3: Zobrazenie štatistík pri neaktívnom vyhľadávaní



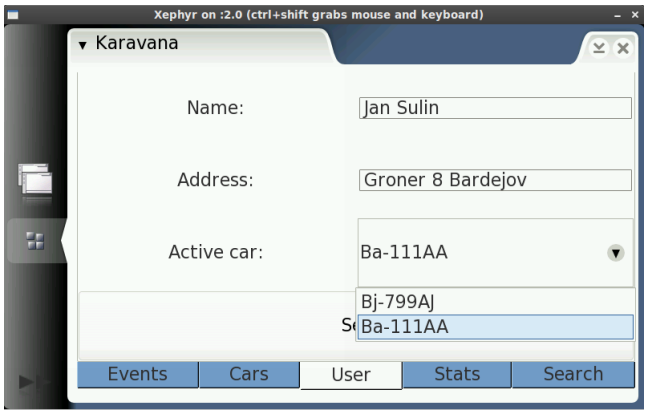
Obrázok A.4: Zobrazenie štatistík pri aktívnom vyhľadávaní



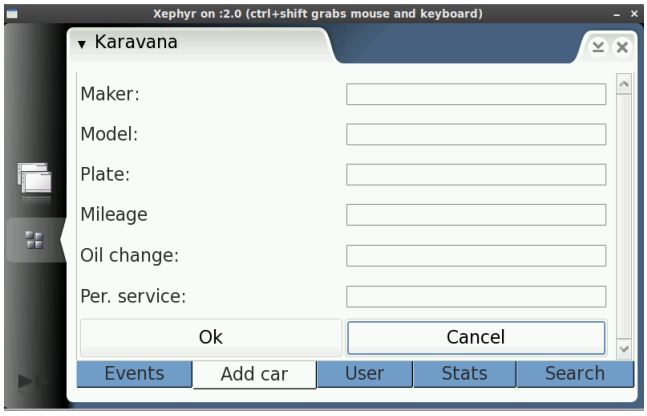
Obrázok A.5: Záložka vyhľadávania



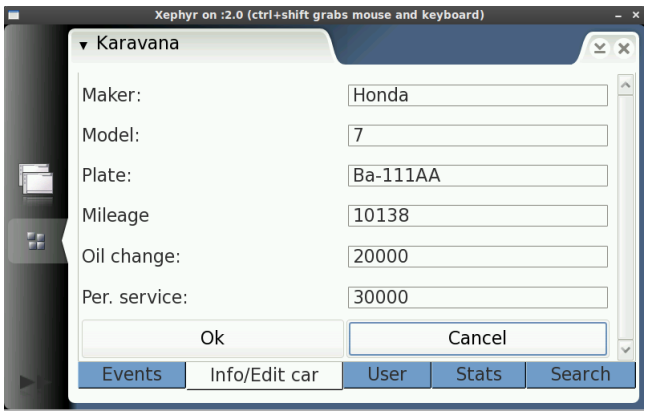
Obrázok A.6: Záložka aktívneho vyhľadávania



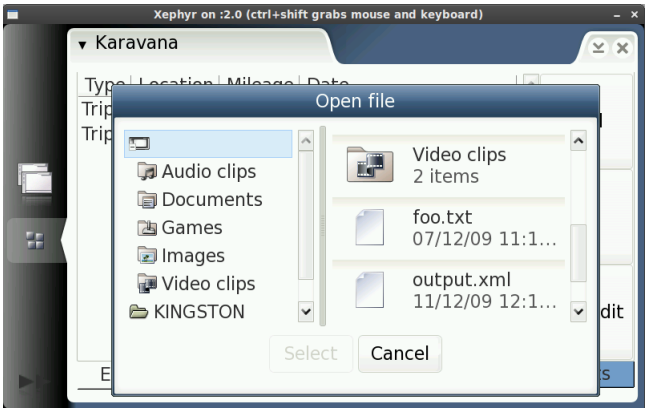
Obrázok A.7: Príklad výberu aktívneho auta



Obrázok A.8: Okno pridávania auta



Obrázok A.9: Okno zobrazenia informácií o aute a ich editovania



Obrázok A.10: Výber xml súboru na import dát

Dodatok B

Obsah priloženého CD

Súčasťou tejto práce je aj CD. Jeho obsah:

- Zdrojové kódy tejto práce vo formáte XML a samotná práca vo formáte PDF.
- Zdrojové kódy a binárna verzia aplikácie Karavana

Literatúra

[1] *Maemo Diablo Reference Manual*, [cit. 30. decembra 2009], URL: <http://maemo.org/maemo_release_documentation/maemo4.1.x/Maemo_Diablo_Reference_Manual_for_maemo_4.1.pdf>. 3.3.1, 3.3.5

[2] *Maemo Diablo Quick Start Guide*, [cit. 30. decembra 2009], URL: <http://maemo.org/development/documentation/maemo-quick-start-guide.pdf>. 3.2, 3.3.1, 3.4, 3.4.1

[3] *GLib Reference Manual*, [cit. 30. decembra 2009], URL: <http://maemo.org/api_refs/4.1/glib2.0-2.12.12/libglib2.0/>. 4.7.3.1

[4] Warkus, M.: *The Official GNOME 2 Developer's Guide*, No Starch Press, 2004, ISBN: 1-59327-030-5. 3.3.2, 3.3.3

[5] Krause, A.: *Foundations of GTK+ Development*, Apress, 2007, ISBN: 1-59059-793-1. 3.3.4, 4.7.1

[6] *Gtk+ Reference Manual*, [cit. 30. decembra 2009], URL: <http://maemo.org/api_refs/4.1/gtk+2.0-2.10.12/libgtk2.0/>. 3.3.4, 4.7.1, 1

[7] *Hildon FM Reference Manual*, [cit. 30. decembra 2009], URL: <http://maemo.org/api_refs/4.1/libhildonfm-2.0.5/>. 3.3.1

[8] *Hildon 2.0.4 Reference Manual*, [cit. 30. decembra 2009], URL: <http://maemo.org/api_refs/4.1/libhildon-2.0.4/>. 3.3.1

[9] *Reference Manual for libxml2*, [cit. 30. decembra 2009], URL: <http://xmlsoft.org/html/index.html>. 3.5.1, 3.5.2, 4.6.4

[10] *Maemo Programming Languages*, [cit. 30. decembra 2009], URL: <http://maemo.org/development/documentation/programming_languages/>. 3.4.3

[11] Owens, M.: *The Definitive Guide to SQLite*, Apress, 2006, ISBN: 1-59059-673-9. 4.6.2

[12] *SQLite C Interface*, [cit. 30. decembra 2009], URL: <http://www.sqlite.org/c3ref/funclist.html>. 4.6.2