Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Mobilní platforma pro podporu vykonávání aktivit řízená aktuálním kontextem

Bc. Bohuslav Mach

Vedoucí práce: Ing. Zdeněk Míkovec, Ph.D.

17. června 2013

Poděkování

Rád bych poděkoval především vedoucímu práce panu Ing. Zdeňku Míkovcovi, Ph.D. za jeho rady a pomoc při tvorbě tohoto textu, oponentovi Ing. Pavlu Žikovskému, Ph.D. a panu Ing. Janu Balatovi za jeho vstřícnost a umožnění uživatelských rozhovorů. Dále bych rád poděkoval svým blízkým a přátelům za velkou podporu, kterou mi projevovali po celou dobu mého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 17. června 2013 .....

Odkaz na tuto práci

Mach, Bohuslav. *Mobilní platforma pro podporu vykonávání aktivit řízená aktuálním kontextem*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2013.

Abstract

This work deals with the creation of a special platform for the efficient running of applications, depending on regular activities. The aim is to develop a draft application for mobile devices that delivers innovative enhancements to existing solutions with a focus on limited users (especially for visually impaired) and their repetitive activities.

Keywords Launcher, Accessibility, Context, Mobile device, Google Android, Visually impaired

Abstrakt

Tato práce se zabývá problematikou tvorby speciální platformy pro efektivní spouštění aplikací v závislosti na pravidelným aktivitách. Cílem je rozpracovat návrh aplikace pro mobilní zařízení, která přináší inovativní vylepšení stávajících řešení se zaměřením na limitované uživatele (především zrakově postižené) a jejich opakované činnosti.

Klíčová slova Launcher, Accessibility, Kontext, Mobilní zařízení, Google Android, Zrakově postižení

Obsah

Seznam obrázků

Úvod

Mobilní zařízení se v posledních letech stala nedílnou součástí našich životů a není nadsázkou, když řekneme, že si svůj běžný den bez těchto moderních vymožeností nedokážeme představit. Obyčejné aktivity, jako zasílání textových zpráv nebo uskutečňování a přijímání hovorů, si už ani neuvědomujeme. Vyřizování emailů, hledání spojení nebo rezervaci oblíbené restaurace můžeme pohodlně realizovat prostřednictvím mobilních aplikací rychle a jednoduše. Díky tomu mnoho uživatelů již nepotřebuje osobní počítač a všechny požadované úkony vyřídí přes svůj mobilní telefon nebo tablet.

Počet aplikací se každým dnem zvyšuje a my objevujeme stále více takových, které nám usnadní každodenní úkony. Jejich instalace je díky specializovaným internetovým obchodům velmi jednoduchá a proto běžný uživatel nepřemýšlí nad tím, kolik jich vlastně v telefonu má. S rostoucím počtem nainstalovaných aplikací se stává hlavní menu čím dál tím víc nepřehledné a hledání zdlouhavější a obtížnější. Drtivá většina dnešních operačních systémů určitou formu řešení nabízí a to především prostřednictvím sdružení podobných/oblíbených aplikací do složek, čímž časovou ztrátu při hledání zmenšíme (díky rozdělení do sekcí). Méně zkušení uživatelé však nemusí být s těmito možnostmi obeznámeni a jakékoliv složitější operace je mohou odradit.

Je tu však další skupina, které tento způsob nemusí tak úplně vyhovovat. Představme si uživatele, který se z nějakého důvodu nachází v časové tísni. Potřebuje otevřít aplikaci ihned a nemůže mobilnímu zařízení věnovat příliš mnoho času. Do takových situací se pravidelně dostávají například řidiči a ačkoliv se to na první pohled nezdá, rovněž do této skupiny patří určitým způsobem i zrakově postižení. Při práci s mobilním zařízením nepředstavují běžné operace pro vidícího člověka žádný problém. Proto například vybrat položku v menu nezabere díky vizuálnímu vjemu zdaleka tolik času, jako

kdybychom všechny položky museli projít jednu po druhé.

Tímto způsobem však musí se seznamy pracovat každý zrakově postižený uživatel. Díky tomu se může dostat do časové tísně daleko snáze, protože operace, která vidícím uživatelům zabere pět minut, trvá této skupině zpravidla několikrát déle. Proč bychom tedy nemohli pravidelně vyhledávané položky řadit na začátek seznamu a nejlépe ty, které by mohl uživatel potřebovat právě v tuto chvíli (v aktuálním kontextu)? Převedeme-li tento požadavek na problém vyhledávání a spouštění aplikací, tak by uživateli v ideálním případě stačilo prozkoumat prvních několik položek, ve kterých by našel požadovanou volbu. Časová úspora by tak mohla být velmi znatelná. Samozřejmostí je, že by tento způsob byl efektivní i pro ostatní uživatele, kteří by tak měli vhodné aplikace ihned na očích a nemuseli by zdlouhavě projíždět menu nebo sdružovat aplikace do složek.

Jak vyplývá z předchozích odstavců, tak aplikace, která by nahradila klasickou domácí obrazovku její chytřejší verzí, by usnadnila práci a ušetřila čas širokému spektru uživatelů, především pak těm, kteří se potýkají s některou formou celoživotní nebo dočasné limitace. V následujícím textu se tedy budeme zabývat návrhem a implementací popisovaného systému, který by měl vynikat jednoduchostí a pro uživatele být maximálně komfortní a přívětivý.

Popis problému, specifikace cíle

V této kapitole se pokusíme nastínit rozsah aplikace a dané problematiky. Nejdříve zmíníme samotné oficiální zadání této práce, poté si přiblížíme pojmy, které se budou nejčastěji objevovat v souvislosti s vývojem aplikace. Jako poslední podkapitola nás čeká vymezení cílů práce, které nám budou sloužit pro představu, jaké úkoly nás budou při implementaci čekat.

Popis problému

Cílem práce je navrhnout a implementovat software pro uživatele „chytrých“ telefonů, který jim umožní efektivnější spouštění aplikací díky sledování jejich pravidelného režimu. V první řadě je třeba analyzovat potřeby a požadavky specifických cílových skupin s omezenou možností interakce (zrakově a pohybově postižení, řidiči atp.). Dále se seznámíme s dostupnými vývojovými platformami, které mohou být reálně využity vzhledem k cílovým skupinám a provedeme analýzu kontextu v závislosti například na poloze, časových údajích, událostech v kalendáři, spouštěných aplikacích, probíhajících činnostech aj.

Na základě uživatelské analýzy a analýzy platforem navrhneme platformu využívající aktuální kontext pro generování nabídky a spouštění uživatelských aktivit v podobě dostupných aplikací na koncovém zařízení. Zaměříme se na přívětivost a srozumitelnost grafického rozhraní a chování aplikace pro cílového uživatele.

Definice pojmů

Vytvoření seznamu definic pojmů, který bude zahrnovat ty nejčastěji zmiňované ve spojení s vývojem aplikace, je velmi důležité z důvodu lepší a snadnější orientace čtenáře v celém textu. Jedná se vlastně o formu výkladového slovníku, jenž pomůže především těm, kteří jsou méně znalí dané problematiky. Pro tuto skupinu by se totiž výsledná práce mohla stát matoucí a těžko uchopitelnou.

Uživatel je entita, která nějakým způsobem interaguje s naší aplikací a využívá ji.

Launcher je domácí (hlavní) obrazovka, kterou uživatel vidí ihned po plnohodnotném spuštění operačního systému.[25]

Aplikace je programové vybavení mobilního zařízení. Jedná se například o emailového klienta, internetový prohlížeč nebo přehrávač hudby.

Senzor je zdroj informací o stavu mobilního zařízení nebo jeho okolí. Například se může jednat o teploměr, GPS nebo Wi-Fi modul.

Data jsou informace získané ze senzorů. Například okolní teplota, poloha mobilního zařízení nebo seznam dostupných Wi-Fi sítí.

Platforma určuje použitý operační systém, knihovny, ale i použité programovací jazyky či kompletní framework (vývojová a běhová platforma).[68]

Accessibility definuje míru dostupnosti určitého produktu definované skupině uživatelů. V našem případě se s tímto pojmem setkáváme především ve spojení se zrakově postiženými uživateli.

Aktivity jsou uživatelská činnosti, které budeme monitorovat a ve kterých se budeme snažit hledat pravidelnosti a šablony.

Kontext vyjadřuje aktuální stav, ve kterém se nachází mobilní zařízení a prostředí okolo něj.

Log/Logovací záznam obsahuje informace o aktuálním stavu mobilního zařízení a jeho okolí. V podstatě se jedná o snímek kontextu, který je uložen v databázi.

Takto zadefinované pojmy nám pomohou lépe se orientovat v probírané problematice a předejít případným nejasnostem a omylům.

Vymezení cílů práce

Vzhledem k povaze řešeného problému je nepraktické vytyčit si cíle prostřednictvím funkčních nebo nefunkčních požadavků, které by dle našeho názoru nedostatečně pokrývaly výsledný záměr. Především také díky tomu, že klasická aplikace, se kterou bude uživatel interagovat, již existuje a my se na ni chceme podívat z trochu jiného úhlu. V zásadě se budeme potýkat se známým problémem, který bychom ale rádi řešili inovativním způsobem a nabídli tak uživateli větší užitek a efektivitu při používání výsledného produktu.

V oficiálním zadání je stanovena cílová skupina, na kterou se budeme zaměřovat. Jelikož chceme, aby tento projekt nebyl slepou uličkou, je nutné vybrat vhodný mobilní operační systém, který bude nejlépe splňovat námi stanovená kriteria a zároveň bude zohledňovat právě potřeby těchto koncových uživatelů. Na jeho základě pak budeme schopni vybrat Launcher, který nám bude poskytovat potřebnou funkcionalitu, s důrazem na chování, jenž je pro uživatele dotykových zařízení a vybraného operačního systému dobře známé. Ve chvíli, kdy budeme mít specifikovanou cílovou skupinu, je nutné zaměřit se na vytvoření prototypu jejího pravidelného režimu, který nejlépe vypozorujeme z uživatelských rozhovorů provedených na základě jednoduché osnovy.

Stanovením rámcové architektury systému a z výše získaných poznatků pak budeme schopni definovat oblasti, jenž bude třeba detailně rozpracovat před samotnou realizací. Ze specifikace problému můžeme uvažovat, že bude třeba pokrýt správu datových zdrojů (aktuální stav mobilního zařízení a okolí) a sjednotit přístup k nim stejně jako umožnit registraci k získávání notifikací o změnách. S takto vytvořeným rozhraním bude snadnější navrhnout systém ukládání záznamů o sledovaných aktivitách uživatele (logovací záznamy), ze kterých pak bude potřeba získat jednotlivé pravidelné aktivity, na jejichž základě budeme provádět filtrování aplikací a jejich zobrazení v uživatelském rozhraní (hlavní obrazovka Launcheru).

Uvážíme-li celkový rozsah výsledného řešení, budeme během vývoje klást důraz především na důkladnou analýzu, která by měla být založena na požadavcích obsahujících pojmy jako nízká provázanost, vysoká flexibilita

a jednoduchá rozšiřitelnost jednotlivých částí systému vzhledem k předpokládané další práci.

Kapitola 1

Analýza a návrh

Správná a dostatečně detailní analýza projektu bude stěžejní součástí tohoto textu a výsledného produktu. Nejprve se budeme věnovat uživatelskému výzkumu, díky kterému budeme schopni modelovat pravidelný cyklus naší cílové skupiny. Provedeme rešerši mobilních platforem a popíšeme si jak by měla vypadat architektura systému, na jejímž základě pak provedeme návrh jednotlivých součástí.

1.1 Uživatelský výzkum

Stěžejní částí naší práce je uživatelský výzkum uskutečněný ve spolupráci s ČVUT FEL, katedrou počítačové grafiky a interakce. Informace z provedených rozhovorů nám pomohou nahlédnout do světa uživatelů, pro které naši aplikaci vyvíjíme, a pochopit jejich potřeby a požadavky. Výstupním produktem pak bude sestavení pravidelného týdenního cyklu, který nám pomůže odhalit s jakým druhem problému se budeme reálně potýkat.

1.1.1 Interview

Seznam následujících několik otázek, které by nám měly pomoci získat od uživatelů potřebné informace pro další vývoj, sloužil jako volnější osnova v závislosti na směru, kterým se rozhovor ubíral. Výsledný počet položek byl během několika iterací (před započetím rozhovorů) redukován a jejich znění mírně upravováno vzhledem k tomu, že na jednotlivé rozhovory bylo vymezeno časové rozhraní cca 10-15 minut.

Dotazy nebyly omezeny pouze na mobilní zařízení, ale brali jsme v potaz i osobní počítač, jelikož mnoho dotazovaných používá různé počítačové programy, jejichž funkcionalitu ale dokáží plnohodnotně zastat i aplikace pro

mobilní telefony/tablety. Celkově bylo uskutečněno sedm rozhovorů s participanty z různých věkových kategorií.

1. Jakého druhu je Vaše limitace?
2. Jak dlouho se s touto limitací potýkáte a kdy k ní došlo? (od narození, v dětství, později)
3. Jaký typ mobilního zařízení využíváte?
    a) Uvažoval/a jste o přechodu na dotykové mobilní zařízení?
4. Jaký je Váš pravidelný režim? (denní, týdenní, dvoutýdenní atp.)
5. Jaké jsou Vaše typické (opakované) činnosti? (cesty, venčení, pořady, práce atp.)
6. Jaké aplikace nejvíce využíváte a kdy? (volání, SMS, kalendář, internet, spoje atp.)
7. Jaké plusy/mínusy jste při jejich používání zaznamenal/a?
8. Které z nich se Vám nejlépe používají a proč?
9. Na jaké problémy při jejich používání narážíte?
10. Existují aplikace, které jste z nějakého (jakého) důvodu přestal/a používat?

1.1.2 Vzorové scénáře

Na základě uživatelského výzkumu si v následujících podkapitolách budeme schopni popsat pravidelné režimy naší cílové skupiny. Jedná se o uměle vytvořený ideální scénář (vytvoření prototypu), který je složen ze všech participantů dané skupiny s přihlédnutím k logickému uspořádání aktivit. Z takto popsaného scénáře a jeho aktivit odehrávajících se v rámci dne/týdne můžeme vypozorovat, jak bude vypadat reálný problém, který se snažíme pokrýt, v praxi a jak se bude naše aplikace v průběhu času schopna učit.

Rozčlenění aktivit

Abychom mohli lépe odhadnout horizont učení, stanovili jsme si několik kritérií pro určitou formu kategorizace jednotlivých aktivit. Jmenovitě to jsou četnost, pravidelnost a variabilita. Na základě takto zvolených údajů můžeme přibližně stanovit, jak dlouho bude aplikaci trvat vysledování jednotlivých pravidelností v rámci našeho scénáře.

Četnost a pravidelnost

Zcela záměrně uvádíme tato dvě kritéria ve společné kapitole, protože spolu velice úzce souvisí, jak si následně vysvětlíme. Zjednodušeně bychom mohli říci, že čím četnější a pravidelnější aktivita, tím lépe a rychleji se ji bude aplikace schopna naučit.

Sledováním četnosti výskytu dané aktivity ve scénáři získáme přehled o cyklech, ve kterých se jednotlivé činnosti objevují, a také o tom, jaký má uživatel režim. Ty, které se vyskytují denně se bude aplikace schopna učit podstatně rychleji, než například ty, které se vyskytují jednou týdně nebo dokonce měsíčně. Díky stanovení pravidelnosti budeme mít přehled o tom, jak obtížné bude pro aplikaci najít tyto pravidelné cykly v záplavě sledovaných dat. Ideálním případem je pro nás absolutně pravidelný výskyt bez mezer (časových úseků vybočujících z pravidelného řádu).

V takovém případě bude aplikace schopna rozpoznat denní aktivitu během 2-3 dní, podobně týdenní během 2-3 týdnů. Můžeme tento odhad zobecnit tím, že si určíme časovou jednotku (den, týden, měsíc atp.). Následně pak bude platit, že se danou aktivitu aplikace naučí během dvojnásobku až trojnásobku této časové jednotky (příslušné její četnosti). Samozřejmostí je, že pokud například uživatel spouští aplikaci s pravidelností

- první den ano,
- druhý a třetí den ne,
- čtvrtý a pátý den opět ano,

tak se v takovém případě prodlouží doba učení na 4-5 dní, ačkoliv se jedná o aktivitu s denní četností. Navíc se může jednat buď o aktivitu, která má být spouštěna pouze ve všední dny, nikoliv o víkendu, nebo ji jen uživatel „zapomněl“ druhý a třetí den spustit.

Četnost výskytu budeme hodnotit slovně podle příslušného časového úseku, např. denní, týdenní, měsíční atp. Pravidelnost výskytu aktivity budeme hodnotit číselně na stupnici od jedné do pěti, přičemž jedna znamená pravidelná a pět znamená nepravidelná.

Variabilita

Výskyt aktivity může a nemusí být striktně vázán na časové období (poslech zpravodajské relace versus čtení knihy). Zde proto přichází ke slovu variabilita, která bude popisovat tento fakt striktně z pohledu časového ukotvení.

Opět platí, že aplikaci se budou lépe rozpoznávat aktivity s menší variabilitou v čase, jako je například zmíněné sledování/poslech hlavní zpravodajské relace nebo pravidelného pořadu (pevný čas).

Variabilitu výskytu budeme popisovat na stupnici od jedné do pěti (podobně jako u pravidelnosti), přičemž jedna udává pevný čas a pět naproti tomu velké časové rozmezí.

Nevidomí

V tomto scénáři se pokusíme namodelovat prototyp pravidelného rytmu nevidomého člověka tak, jak jsme ho zaznamenali během našich rozhovorů.

Všední den

- **6:00 - 7:00** - vstávání
  - četnost: denní | pravidelnost: 1 | variabilita: 2
- vypravení dítěte do školy, poslech rádia, kontrola emailů, snídaně, venčení psa
  - nabídnutí aplikace pro poslech rádia, emailového klienta, aplikace s počasím
  - četnost: denní | pravidelnost: 1 | variabilita: 1-2
- **8:30** - poslech pravidelného pořadu „Jak to vidí“ na ČRo 2
  - nabídnutí aplikace pro poslech rádia (vstupní parametr - frekvence ČRo 2)
  - četnost: denní | pravidelnost: 2 | variabilita: 1
- práce/aktivita
  - nabídnutí aplikace kalendář (denní přehled klientů)
  - po skončení aktivity s klientem nabídnout aplikaci pro vystavení účtu
  - preferování aplikací využívaných při práci
  - četnost: denní | pravidelnost: 1 | variabilita: 2
- oběd, venčení psa
  - četnost: denní | pravidelnost: 1 | variabilita: 3

- práce/aktivita
  - preferování aplikací využívaných při práci
  - četnost: denní | pravidelnost: 1 | variabilita: 2
- **19:00** - poslech zpráv na ČT24
  - nabídnutí aplikace pro sledování ČT24 / internetového prohlížeče (vstupní parametr - URL živého vysílání)
  - četnost: denní | pravidelnost: 1-2 | variabilita: 1
- večeře, venčení psa
  - četnost: denní | pravidelnost: 1 | variabilita: 3
- četba, poslech televize/rádia, internet
  - nabídnutí aplikace pro poslech rádia / sledování televize, čtečky knih / aplikace pro přehrávání namluvených knih, internetového prohlížeče
  - preference aplikací využívaných při relaxaci
  - četnost: denní | pravidelnost: 1-2 | variabilita: 3

Úterý a čtvrtek

- **18:00** - vaření večeře
  - nabídnutí aplikace s recepty (ve spojení s chytrou lednicí by mohl být vstupním parametrem seznam potravin)
  - četnost: týdenní | pravidelnost: 1-2 | variabilita: 2

Pátek

- **14:00** - španělština po Skype
  - nabídnutí aplikace Skype (alternativně se vstupním parametrem pro přímé volání kontaktu)
  - četnost: týdenní | pravidelnost: 2-3 | variabilita: 1-2

Sobota

- **10:00 - 11:00** - odjezd na plavání a 12:00 - 13:00 - odjezd z plavání
  - nabídnutí aplikace pro vyhledání spoje MHD (vstupní parametry: odkud, kam), aplikace s aktuálním počasím, aplikace s mapami, volání do navigačního centra, aplikace pro navigaci (vstupním parametrem může být cílový bod, pokud je znám)
  - četnost: týdenní | pravidelnost: 2-3 | variabilita: 1-2

Neděle

- **13:00** nedělní pohádka na ČRo 2
  - nabídnutí aplikace pro poslech rádia (vstupní parametr - frekvence ČRo 2)
  - četnost: týdenní | pravidelnost: 1-2 | variabilita: 1

1.2 Rešerše platforem operačních systémů

Předtím, než začneme s implementací naší aplikace, je nutné zvolit vhodnou platformu operačního systému. V následujících kapitolách si rozvedeme jednotlivá kriteria, se kterými konfrontujeme předem vytipované platformy a na základě takto získaných poznatků se pak rozhodneme, která z nich bude pro naše účely nejvhodnější.

Na základě předběžného prozkoumání aktuálního trhu a statistik prodejů jednotlivých zařízení, jsme dospěli k závěru, že se budeme primárně zabývat platformami

- Apple iOS,
- Google Android a
- Windows Phone.

V rámci rešerše se tedy zaměříme především na požadavky, které vznášíme na finální platformu my jako vývojáři, a uživatelé s různou formou postižení zraku, protože naše aplikace cílí primárně na ně. Naším úkolem je ale zároveň vzít v potaz atraktivitu vybraného operačního systému pro ostatní uživatele, protože chceme pokrýt co největší oblast trhu a zpřístupnit tak aplikaci (i do budoucna) co nejširšímu okruhu zájemců.

1.2.1 Podíl na trhu

Vzhledem k budoucnosti a následnému vývoji naší aplikace by nás mělo zajímat, jaký trend v zastoupení na trhu zaznamenávaly jednotlivé platformy v předchozích letech. Z těchto informací se následně rámcově můžeme domnívat, jak si platforma, kterou budeme favorizovat, povede do budoucna.

Její rozšířenost je jedním z klíčových atributů výběru, protože i sebelepší platforma bez podpory známých výrobců mobilních zařízení nemůže z dlouhodobého hlediska na trhu obstát. Důvodů, proč tomu tak je, je hned několik. Čím více výrobců bude nabízet daný operační systém, tím více potencionálních uživatelů si k němu najde cestu. S tím úzce souvisí zájem vývojářské komunity, která se bude primárně upínat tam, kde cítí finanční potenciál a růst uživatelské základny.

V současné době si můžeme povšimnout dvou větví, kterými se operační systémy, potažmo jejich tvůrci, ubírají. Google Android a Windows Phone si zákazník může zakoupit od vícero výrobců mobilních zařízení a naproti tomu Apple iOS je pevně svázán se svými zařízeními (tvůrce operačního systému je zároveň výrobcem mobilních zařízení).

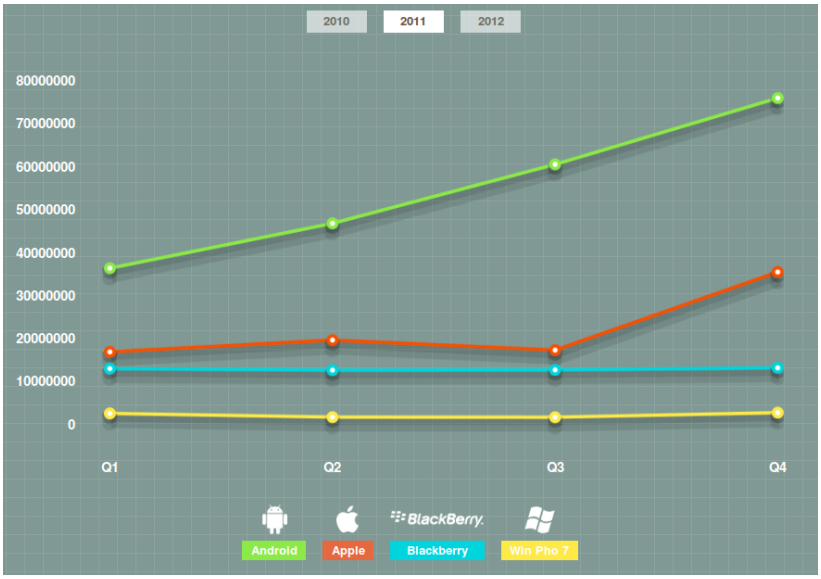
Z uvedených grafů 1.1 a 1.2 na straně 14 je zřejmé, že hlavními hráči na trhu jsou Apple iOS a Google Android, u kterého je pak rostoucí trend jasně patrný, stejně jako u Windows Phone. Jeho podíl na trhu je sice zatím malý, ale do budoucna se zdá být slibnou platformou. Otázkou je, zda (a případně jak rychle) se mu podaří výrazněji konkurovat ostatním platformám.

Současný trend navíc odhaluje nastávající zlom pro zrakově postižené. Dříve byl pro tuto část trhu jasnou jedničkou operační systém Symbian OS, u kterého ale zřetelně vidíme propadající se tendenci. Tito uživatelé budou v budoucnu nuceni přecházet (pokud již tak neučinili) na jiná zařízení s jinými operačními systémy. Z toho vyplývá že namísto jedné cílové platformy jich tu budeme mít hned několik - nejpravděpodobněji právě ty, které jsou předmětem naší rešerše.

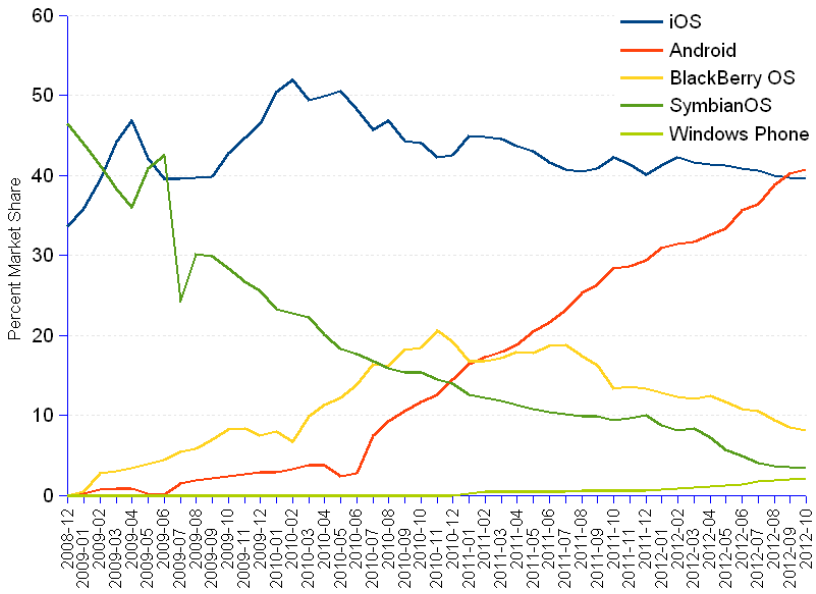
1.2.2 Spokojenost uživatelů

Cílem každé aplikace by měla být maximální spokojenost uživatelů při jejím používání a nejinak je tomu u samotných operačních systémů. Pokud nebude uživatel spokojen s jeho celkovým používáním, bude mít tendence poohlížet se po jiném řešení, kde jinde než u konkurence. Navíc jeho nespokojenost z operačního systému může paušálně ovlivňovat i dojmy ze samotné aplikace.

V rámci každoroční čtenářské ankety „Readers' Choice Awards“ zjišťuje webová stránka pcmag.com celkovou spokojenost uživatelů s používáním



Obrázek 1.1: Čtvrtletní prodeje zařízení s vybranými operačními systémy v roce 2011 (celosvětově).[51]



Obrázek 1.2: Vývoj zastoupení na trhu pro jednotlivé operační systémy v Evropě.[50]

operačních systémů na zařízeních, která vlastní a denně používají. Hodnocení zobrazené na grafu 1.3 na straně 16 probíhá v mnoha kategoriích, např. dostupnost aplikací, hudební přehrávač nebo spolehlivost. Jak vidíme na přiloženém grafu, nejvíce spokojeni jsou uživatelé s Apple iOS a Windows Phone, na třetím místě se se ztrátou 0.8 bodu umístil Google Android.

Jeden z výsledků studie zobrazené na grafu 1.4 na straně 16 potvrzuje rostoucí tendenci spokojenosti u operačního systému Google Android společně s Apple iOS. Windows Phone v posledním zkoumaném období zaznamenal znatelný skok, ale vzhledem k předchozímu vývoji z toho nelze mnoho usuzovat.

Pro komunitu zrakově postižených bohužel žádné takové ankety ani údaje nemáme, proto se o spokojenosti těchto uživatelů můžeme jen domnívat na základě jejich zkušeností, o které se s námi podělili ať už v rámci uživatelského výzkumu nebo článků na internetu.

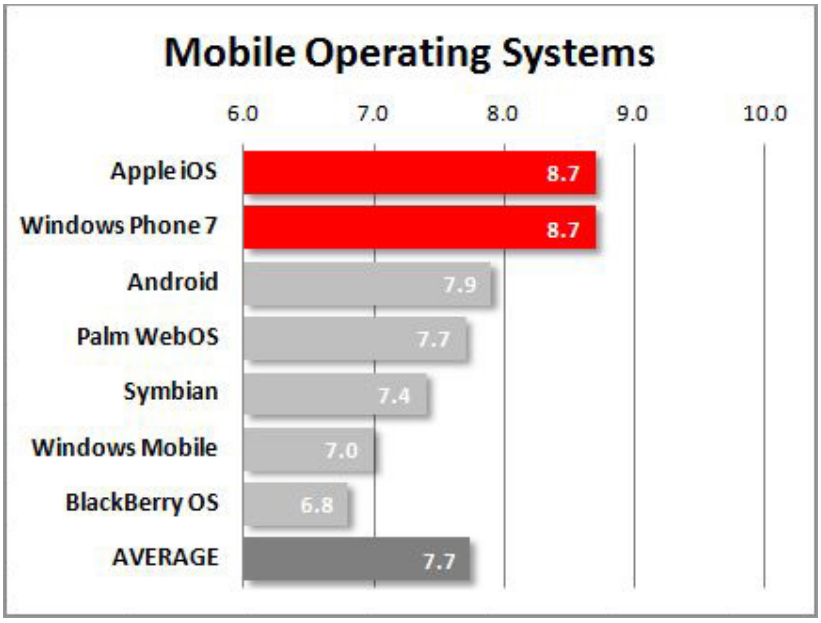
1.2.3 Cena koncových zařízení

Důležitým kriteriem pro výběr nového telefonu může být pro zákazníka tzv. poměr cena/výkon, tedy co všechno dostane za danou cenu. V případě uvažovaných operačních systémů je zřejmé, že výhodu v tomto ohledu budou mít Google Android a Windows Phone a to především proto, že jsou distribuovány na mnoho koncových zařízení s různými cenami a od různých výrobců. Zákazník tak má díky tomu podstatně širší paletu možností.

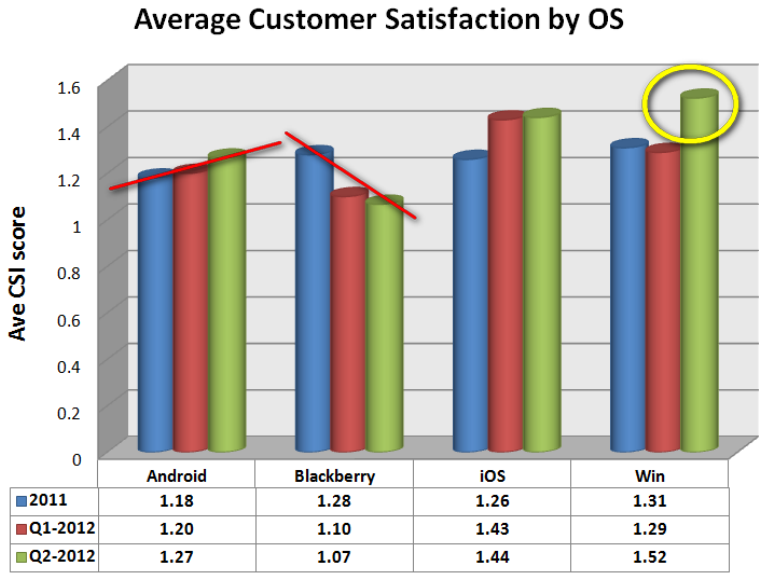
Přesným opakem tohoto systému pak je Apple iOS, který je distribuován výhradně na zařízení produkovaná společností Apple, která si samozřejmě určuje i cenu koncových zařízení. Není žádným tajemstvím, že řešení od společnosti Apple v mnoha případech cenově přesahují srovnatelná zařízení s ostatními operačními systémy - což ovšem dohání například nápaditými a inovativními řešeními v oblasti ovladatelnosti a použitelnosti. Koncový zákazník tímto přístupem získá výhodu uniformního prostředí, ovšem na úkor ztráty variability výběru.

Provedeme-li jednoduchý průzkum například v rámci serveru Heureka[37], dojdeme k jasnému závěru. Na spodním konci cenového rozpětí (tedy nejlevnější řešení) se nachází zařízení s operačním systémem Google Android. Pravým opakem jsou pak řešení s operačním systémem Apple iOS, které jsou v současnosti nejdražší.

Trochu jiným pohledem se však musíme podívat na problematiku zrakově postižených, protože u těchto uživatelů může být měřítko cena/výkon určitým způsobem vychýlené díky případné chybějící funkcionalitě. Omezujícím faktorem pak není tolik cena, jako právě (nadstandardní) možnosti zařízení, jejichž absence pro vidícího uživatele žádný problém nepředstavuje.



Obrázek 1.3: Spokojenost uživatelů s jejich operačním systémem (10.0 - nejlepší) vypracovaný webovou stránkou pcmag.com.[56]



Obrázek 1.4: Přehled průměrné spokojenosti zákazníků s daným operačním systémem vypracovaný v rámci studie Amplified Analytics.[4]

Například se může jednat o hardwarová tlačítka, klávesnici nebo o kvalitu ozvučení systému a jeho ovládání pomocí nadefinovaných gest.

S tím tedy úzce souvisí případná limitace výběru na dražší zařízení a následně dotační politika, na základě které může být zrakově postiženému poskytnuta finanční pomoc. Za jakých podmínek mohou o některou formu příspěvku tito lidé žádat jsme se dozvěděli v odpovědi na náš emailový dotaz od Evy Vonešové z informačního centra Tyflonet (celé znění v příloze B.1 na straně 75). Protože se s velkou pravděpodobností obvykle jedná o pomůcku do 24 000 Kč, může být taková finanční pomoc poskytnuta pouze osobám se stanovenou výškou příjmu.

1.2.4 Dostupné aplikace

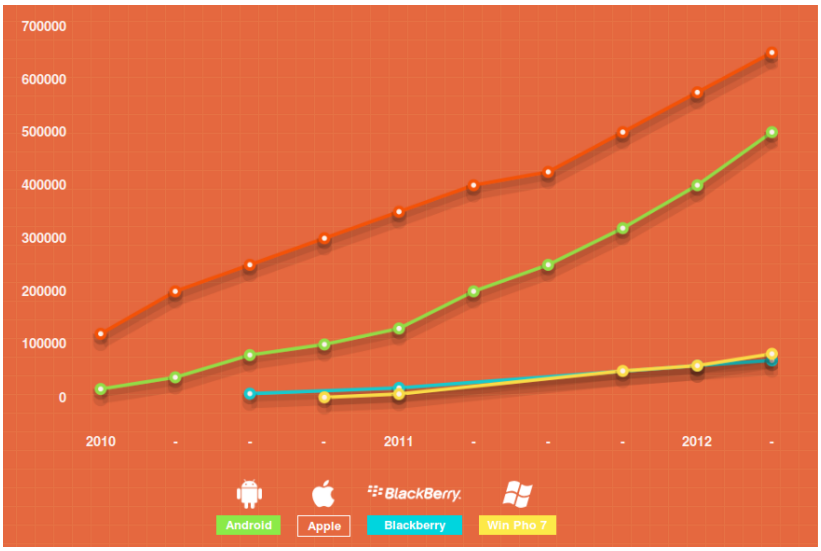
Abychom si udělali ucelený obraz o trhu a jeho vývoji, je vhodné zanést do našeho rozhodování také údaje o dostupných aplikacích pro dané operační systémy. Mělo by nás zajímat, zda počet dostupných aplikací v jednotlivých webových obchodech stoupá nebo stagnuje.

V případě, že daný operační systém zaznamenává rostoucí tendenci, můžeme se domnívat, že ostatní vývojářské firmy stále vidí v této sekci trhu příležitost a především peníze, tedy potencionální zisk. Samozřejmostí je, že tato domněnka platí u stagnujících počtů dostupných aplikací zcela opačně. Tento fakt může především znamenat, že operační systém přestává být pro vývojáře perspektivní nebo může svědčit o aktuálním nasycení trhu.

Na grafu 1.5 na straně 18 si můžeme povšimnout viditelně stoupající tendence u všech námi vybraných operačních systémů. Jasně se ale potvrzuje prozatímní dominantní postavení Google Android a Apple iOS, kteří nabízeli ve druhém kvartálu (Q2) roku 2012 500000 a 650000 aplikací, přičemž v grafu vidíme, že prvně jmenovaný svého hlavního konkurenta začíná dohánět. Zároveň nás také zaujme vývoj počtu dostupných aplikací u operačního systému Windows Phone, což naznačuje jeho stoupající oblíbenost jak mezi uživateli, tak mezi vývojáři.

Měli bychom ale zároveň mít na paměti, že vysoké množství dostupných aplikací nutně nemusí garantovat jejich kvalitu ani použitelnost. V otázce kvality pak záleží na tom, jakou politiku v tomto směru vyznává daná společnost. Například aplikace publikovaná na Apple App Store by měla procházet oproti Google Play, kde je publikování volnější, před svým uveřejněním poměrně složitým schvalovacím procesem. Na druhou stranu, přítomnost více aplikací poskytujících podobnou funkcionalitu dává uživatelům větší volnost a možnost vybrat si tu, která jim nejvíce vyhovuje.

Problém s relevantností tohoto kritéria může nastat v případě zrakově postižených. Z dostupných zdrojů nejsme schopni žádným způsobem usou-



Obrázek 1.5: Vývoj počtu dostupných aplikací pro jednotlivé operační systémy.[51]

dit, kolik z těchto aplikací je skutečně vhodných a užitečných pro tuto skupinu uživatelů (především z hlediska přístupnosti). Abychom si mohli alespoň částečně udělat představu o aspektech tohoto problému, bylo by vhodné vybrat několik zástupců z aplikací, které by mohly být potencionálně zajímavé pro zrakově postižené, a podívat se, jak a zda tyto parametry splňují.

Vhodným nasměrováním při jejich výběru by mohl být uživatelský výzkum a na jeho základě vystavěný vzorový scénář. Pro potřebu porovnání jsme tedy vybrali aplikace z několika oblastí, přičemž se konkrétně jedná o emailového klienta, poslech rádia, mapy, kalendář, vyhledávání spojů MHD, VoIP (Skype), televizní zpravodajství (ČT24), internetový prohlížeč, počasí, poznámky a čtečku knih.

Apple iOS je na zrakově postižené uživatele dobře připraven a aplikace, které jsou nativně zahrnuty v rámci tohoto operačního systému[66], lze prohlásit za plně přístupné. Ze zbývajícího seznamu se nám přizpůsobené pro tuto skupinu podařilo najít všechny aplikace kromě čtečky knih (jako specializované aplikace) a u aplikace IDOS pro vyhledávání spojů uživatelé zaznamenali problematicky vyřešený našeptávač, který komplikuje práci s aplikací.

Google Android je, stejně jako jeho konkurent, také ozvučen. Zároveň zde opět platí, že aplikace, které jsou v zařízení nativně nainstalovány lze prohlásit za přístupné[58][26]. Z ostatních aplikací se nám podařilo najít

všechny kromě čtečky knih (jako specializované aplikace).

Windows Phone v této kapitole zmíníme pouze letmo, protože samotný systém není vůbec ozvučen a z toho důvodu nemůžeme ani hodnotit kvalitu dostupnosti aplikací.

1.2.5 Možnosti a omezení

V závěru naší rešerše se podíváme na jednotlivé operační systémy z pohledu jejich možností a omezení. Tento problém bychom měli uvažovat jak v rovině uživatelské, tak zároveň vývojářské.

Jak jsme si již uvedli, naše aplikace by měla pomáhat mimo jiné uživatelům, kteří mají nějakou formu postižení zraku. Bude nás tedy samozřejmě zajímat, jak jsou schopny jednotlivé operační systémy komunikovat s takto limitovanými uživateli. Jako kritéria jsem vybrali tyto oblasti:

- HW tlačítka (jejich počet, hmatnost, namapovaná funkcionalita)
- Ozvučení systému
- Accessibility API
- Accessibility podpora pro vývoj aplikace + existence a kvalita IDE

Hlavním cílem naší aplikace by měla být přímá modifikace výchozího menu operačního systému a nabízení (případně parametrizovaných) aplikací ke spuštení. Důležité je také propojení naší aplikace s funkcemi telefonu. Z vývojářského hlediska nás tedy zajímá, zda se dá něco takového v rámci dané platformy provést či nikoliv, případně za jakých podmínek a co je k tomu nutné. Jako kritéria jsme zvolili tyto oblasti:

- Nahrazení/modifikace launcheru
- Parametrizované spouštění aplikací
- Propojení se základními funkcemi (telefonování, zprávy, kalendář, atd.)

HW tlačítka

Operační systém Apple iOS se vyskytuje (jak jsme si uvedli již dříve) na omezeném množství zařízení. Tím pádem máme celkem přehled o designu všech modelů a také o výhledu do budoucna. V současné době je vzhled jednotlivých zařízení unifikovaný a politika společnosti Apple nenaznačuje, že by se mělo něco radikálně měnit (není tedy pravděpodobné, že by například začala vyrábět zařízení s HW klávesnicí).

Na těle iPhone najdeme celkem 4 fyzická tlačítka (přední strana - tlačítko Home (Domů), levá strana - dvě tlačítka pro ovládání hlasitosti, horní strana - uspávací/vypínací tlačítko) a jeden přepínač (levá strana - dvoupolohový přepínač tichého a hlasitého módu)[67]. Hmatnost všech tlačítek a přepínače je zřejmá hned na první pohled a neměl by s ní být žádný problém. Všechny ovládací prvky buď vystupují nebo vytvářejí prohlubeň (tlačítko Home) a mají zároveň jasně namapovanou funkcionalitu[5]. Tento fakt ovšem nebrání vývojářům si chování některých prvků přemapovat (například aplikace pro fotografování může fotit stiskem tlačítka pro ovládání hlasitosti), ačkoliv je to striktně proti politice společnosti Apple.

Situace operačního systému Google Android bude přesně opačná. Vzhledem k tomu, že je prodáván na velmi mnoha různých zařízeních je prakticky nemožné najít jakýkoliv unifikovaný design. Pro tato zařízení bývají typická tlačítka pro ovládání hlasitosti (obvykle na boku) a pro uspání/vypnutí (nahoře), s jejichž hmatností by neměl být problém. Pod displayem obvykle najdeme v dolní části 3 ovládací prvky (u některých zařízení ovšem i 4), přičemž u starších zařízení se mohlo jednat i o trackball nebo trackpad. Hmatnost těchto prvků přitom není u novějších zařízení příliš dobrá, protože nijak nevystupují (až na výjimky v podobě některých výrobců, kteří ponechávají vystupující Home tlačítko) a v některých případech se v podstatě nejedná ani o HW tlačítka v pravém slova smyslu. Jejich výhodou je, že poskytují zpětnou vazbu ve formě zavibrování. Některá zařízení navíc disponují HW vysouvací klávesnicí - zde je hmatnost kláves samozřejmě v pořádku.

Všechny prvky mají pevně danou funkcionalitu, ovšem problém může nastat napříč různými výrobci v případě prvků pod displayem. Neexistuje totiž jednotný postup pro mapování určité funkcionality na určitý prvek. Může se tak stát, že například tlačítko pro funkci „zpět“ bude u jednoho výrobce vpravo a u druhého vlevo. V zásadě však platí, že se na těchto pozicích vždy nachází prvky se stejnou funkcionalitou, jenom jinak uspořádané. I v tomto případě je možné přemapovat chování některých prvků a i v tomto případě to není doporučeno ze strany společnosti Google.

Windows Phone na tom je podobně jako Google Android. Necílí primárně na jeden typ zařízení a proto dochází k tomu, že se s ním můžeme setkat u více výrobců (prozatím pouze HTC a Nokia). Nicméně se zdá, že si oproti zařízením s výše zmíněným operačním systémem zachovávají jednotné rozložení prvků pod displayem. V ostatních oblastech se prakticky shodují.

Ozvučení systému

Z našich uživatelských testů a dostupných článků[59][44][64] vyplývá, že Apple iOS nabízí svým zrakově postiženým uživatelům vysoký komfort použitelnosti. Ti mohou aktivovat režim zpřístupnění zcela bez pomoci vidící osoby.

Google Android je na tom oproti konkurentovi z předchozího odstavce hůře (verze 2.3.x), i když po jistých úpravách zařízení (které však, narozdíl od Apple iOS, zrakově postižený uživatel sám nezvládne) lze docílit poměrně uspokojivé formy přístupnosti[57]. Na zvýšení tohoto komfortu navíc Google zapracoval a k jeho podstatnému zlepšení dochází s verzí 4.0, která už se velmi přibližuje úrovni, kterou nabízí Apple iOS[45].

Windows Phone si ze všech našich zkoumaných operačních systémů vede patrně nejhůře, což také dokazují mnohé články[77][28][71] uživatelů zabývající se touto problematikou. Současná situace tedy podle všeho vypadá tak, že pro tuto platformu neexistuje žádný odečítač obrazovky a proto se v současné době jeví jako absolutně nepoužitelná pro uživatele se zrakovým postižením, ačkoliv si firma Microsoft tento fakt zjevně uvědomuje a některé mírné formy zrakového postižení reflektovala v nastavení přístupnosti[52] (které však pro nás prozatím nejsou dostačující). Toto zjištění může být obzvláště nepříjemné pro uživatele předchozího operačního systému od Microsoftu - Windows Mobile. Ten totiž s přístupností problémy neměl a pokud budou chtít přecházet na jeho nástupce, bude to pro ně v současné chvíli znamenat neřešitelný problém.

Accessibility API

Apple iOS na svých stránkách pro vývojáře poskytuje detailní informace k problematice přístupnosti pro zrakově postižené[2]. Vývojáři jsou předloženy důvody, které by ho měly vést k uvedení těchto principů do praxe a následně zmíněn stručný přehled toho, co může vývojář při svém snažení využít, aby byl finální produkt co nejpřívětivější, v kapitole příznačně nazvané „iPhone Accessibility API and Tools“.

Dokument zabývající se touto problematikou v rámci platformy Google Android poskytuje svým vývojářům prakticky stejné informace jako operační systém zmíněný v prvním odstavci. Za zmíňku jistě stojí kapitola „Accessibility Developer Checklist“, poskytující vývojářům seznam bodů, na základě kterých se ujistí, že je jejich aplikace plně přístupná zrakově postiženým uživatelům.

Vzhledem k celkové absenci odečítače obrazovky na platformě Windows Phone nemůžeme očekávat žádná doporučení a pokyny pro vývoj aplikací

dostupných zrakově postiženým uživatelům.

Vývoj aplikací

Apple nabízí svým vývojářům návody, jak a proč svou aplikaci udělat přístupnou uživatelům se zrakovým postižením[2]. Na problém ovšem narazíme v případě, že se začneme zajímat o způsob, jak aplikace pro Apple iOS vyvíjet. Zjistíme totiž, že budeme potřebovat vývojové prostředí (IDE) nazvané Xcode ve spojení iOS SDK, které je zdarma, ovšem dostupné pouze pro operační systém OS X, což ve spojení s adekvátním hardwarem znamená nezanedbatelné výdaje.

Google v tomto případě samozřejmě nijak nezaostává a svým vývojářům také nabízí oficiální návody, jak zpřístupnit své aplikace zrakově postiženým[1]. Získává ovšem navrch v případě volby vývojového prostředí. Oficiálně podporované je Eclipse IDE s Android SDK, které je multiplatformní [27] (v současné době dostupné pro Linux, Windows a Mac OS X) a zdarma. Pro ty, kteří z nějakého důvodu nechtějí využívat služeb Eclipse IDE, jsou na oficiálních stránkách pro vývojáře[11] připraveny návody, jak vyvíjet aplikace pomocí jakéhokoliv jiného prostředí.

Vzhledem k tomu, že Windows Phone zatím žádnou větší podporu přístupnosti pro zrakově postižené nenabízí, nedá se ani očekávat, že by pro své vývojáře nabízel doporučení nebo ucelený pohled na tuto problematiku. Pro vývoj aplikací na Windows Phone je zapotřebí stáhnout Windows Phone SDK[74], které v sobě obsahuje všechny potřebné nástroje[75] (především Microsoft Visual Studio). Krokem kupředu je uvolnění jejich speciálních verzí zcela zdarma, nicméně faktem zůstává, že k plnohodnotnému vývoji aplikací bude zapotřebí operační systém Microsoft Windows. V případě SDK pro Windows Phone 7.x to jsou Windows 7 / Windows 8 / Windows 8 Pro / Windows Vista, pro Windows Phone 8 pak dochází k zeštíhlení pouze na Windows 8 / Windows 8 Pro.

Nahrazení/modifikace launcheru

V současné chvíli není v případě Apple iOS legálně možné modifikovat launcher (Home screen) jako celek. Existují sice aplikace, které určitým způsobem umožňují jeho modifikaci[40], to nám však pro naše účely nestačí. Nahrazení stávajícího launcheru tedy není možné bez operace nazývané jailbreak[43], která je ovšem porušením licenční smlouvy koncového uživatele[18]. Po jejím provedení je na toto zařízení možné nahrávat neoficiální aplikace a tím pádem i aplikaci umožňující změnu standardního launcheru[21].

Naproti tomu Google Android nemá s touto operací žádný problém, jelikož se u něj jedná o nahrazení stávající aplikace (launcheru) aplikací jinou (podobně jako nahrazení standardní klávesnice, emailového klienta atp.) bez jakékoliv jiné úpravy telefonu. Dokonce existuje zdrojový kód standardního launcheru ke stažení a inspiraci[16] a stejně tak mnoho jiných vývojářských počinů (například ADW Launcher[3]).

Windows Phone vyznává z hlediska vývoje u svého launcheru (Start screen) podobnou myšlenku jako Google Android, nicméně vzhled je naprosto rozdílný. Zdá se však, že požadovaná modifikace launcheru je v zásadě možná, vzhledem k dostupným aplikacím na Windows Phone Marketplace, které řeší podobný problém[70][60].

Parametrizované spouštění aplikací

Podstatnou úsporou času by mohla být pro uživatele situace, kdy jedním kliknutím spustí aplikaci, které bude při jejím spuštění předán jeden nebo více parametrů. V ideálním případě by tím pádem mohl náš launcher ušetřit uživateli čas, který by strávil v rámci ruční konfigurace, a zjednodušit interakci. Například přehrávači rádií by tak mohla být předána konkrétní stanice, vyhledávači spojení start a cíl nebo aplikaci Skype indetifikátor uživatele.

Apple iOS umožňuje toto realizovat pomocí tzv. URL Scheme[19], pomocí kterého se aplikace v operačním systému registrují pro využití ostatními [46] a předat jim parameter[48]. Podobný postup lze uplatnit také v případě operačního systému Google Android v podobě intentů[13]. Windows Phone nabízí podobnou možnost jako oba zmíněné operační systémy v rámci tzv. protocol handlers[47].

Omezujícím faktorem samozřejmě zůstává, že aplikace, které hodláme spouštět, musí tuto funkcionalitu podporovat a korektně se registrovat v rámci operačního systému.

Propojení se základními funkcemi

Apple iOS umožňuje vývojáři komunikovat s vestavěným kalendářem a dokonce se registrovat pro přijímání upozornění o změnách[42]. Nepodařil se nám však dohledat přehled všech možných notifikací, ke kterým se lze v rámci naší aplikace přihlásit. Pravděpodobně to však nebude možné u aplikací spravujících telefonní hovory, textové zprávy nebo emaily, jelikož iOS oficiálně neumožňuje přistupovat k jejich záznamům v logu[23]. Podobně se nám nepodařilo dohledat informace o možném přístupu k nastavení systému (GPS, Wi-Fi atp.).

Google Android také umožňuje komunikaci s vestavěným kalendářem[22][8], nicméně registrace pro přijímání upozornění o změnách není příliš komfortní[14][65]. Oproti operačnímu systému Apple iOS však vývojáři nabízí přístup k záznamům telefonních hovorů[9] a neoficiálně i k textovým zprávám, nicméně se jedná o obecně nedoporučovaný způsob, který nemusí zaručovat potřebnou funkcionalitu na všech zařízeních[6] a podobná situace pravděpodobně nastane v případě emailových zpráv. Požadovaného chování vzhledem k zachytávání upozornění o změnách objektů bychom však mohli dosáhnout pomocí tzv. ContentObserverů[10]. Systémové nastavení, například stav přijímače GPS[12], Wi-Fi[17] nebo zvukový profil[39], se dá jednoduše zjistit.

Windows Phone, stejně jako jeho konkurenti, rovněž nabízí přístup k datům kalendáře[38], ale možnost registrovat se k pozorování změn se nám nalézt nepodařilo. Stejná filosofie jako v případě operačního systému Apple iOS je zde pak uplatňována v případě přístupu k seznamu hovorů a textových zpráv[72]. Podobně jako u platformy Google Android je i zde možné získat informace o GPS[73] a Wi-Fi[76].

1.2.6 Výsledky řešerše

Ze průzkumu provedeném v předchozích kapitolách bychom měli být nyní schopni vybrat mobilní operační systém, který se pro naše potřeby hodí nejvíce. Jak se ukázalo, naprosto nevhodný je v současné chvíli operační systém Windows Phone z důvodu absence odečítače obrazovky a prakticky nulové podpory pro uživatele se zrakovým postižením stejně jako vývojářů zaměřených na tuto skupinu.

Operační systémy Apple iOS a Google Android jsou velmi vyrovnanými soupeři a rozhodovat mezi nimi tedy budeme na základě specifických kritérií. Apple iOS a jeho zařízení jsou vhodnější pro nevidomé uživatele z pohledu uniformního prostředí, nicméně Google Android jasně vyhrává v pro nás důležitých oblastech, kterými jsou snadná modifikace launcheru a multiplatformní IDE, které díky tomu nevyžaduje externí náklady v podobě zakoupení licence opračního systému (lze vyvíjet i na Linuxu). Neméně důležitá je i oficiální možnost přístupu k informacím o stavu mobilního zařízení.

Na základě výše uvedených argumentů budeme pro následný vývoj naší aplikace favorizovat právě operační systém Google Android.

1.3 Architektura systému

Architektura systému bude obsahovat několik nezbytných vrstev. Její hrubý nástin se bude sestávat nejspíše z těchto prvků:

- Senzory
- Data ze senzorů (včetně aktivity OS)
- Sémantika dat
- Interpretace dat

Jednotlivé vrstvy budou propojeny procesy a vznikne nám tak stromová struktura, jejímž kořenem je vrstva „Interpretace dat“. Ta bude pro svoji funkčnost využívat informace ze dvou podřízených vrstev a to „Data ze senzorů“ a „Sémantika dat“. „Data ze senzorů“ společně s vrstvou „Aktivita OS“ pak budou shromažďovat informace z dostupných senzorů ve vrstvě „Senzory“ a stavu operačního systému. Pokud to bude implementace vyžadovat, nemělo by být překážkou přistupování i o více než jedno patro hlouběji ve stromové struktuře.

1.3.1 Senzory

Seznam uvažovaných senzorů, které bychom mohli v rámci naší aplikace monitorovat, je následující:

- Síť, GPS, Wi-Fi, Bluetooth, Internet
- Akcelerometr, Okolní teplota, Gravitace, Gyroskop, Okolní světlo, Magnetické pole, Okolní tlak, Proximity („blízkost“ objektu, např. přiložení k uchu), Okolní vlhkost, Rotace
- Hovory, SMS, Kalendář, Čas + datum, Spuštěné aplikace
- Klávesnice, Kamera, Mikrofon
- Interakce s displejem
- API

V souvislosti s výše uvedeným seznamem musíme brát na vědomí, že na trhu je v současné době dostupné velké množství zařízení, přičemž každé z nich může mít rozlišné vybavení (senzory). Ty mohou být navíc ne/dostupné v čase, například aplikace nebude schopna získávat informace z modulu GPS v případě, že se bude zařízení nacházet v budově nebo analogicky pro modul Síť, pokud se bude nacházet mimo pokrytí signálem. Všechny tyto aspekty budou ovlivňovat spolehlivost a přesnost následně získaných dat.

Hlavním úkolem této vrstvy proto bude poskytovat informace o aktuálně připojených (dostupných) senzorech (v naší terminologii je budeme vnímat jako připojitelné moduly) a jejich vlastnostech. Nemusí se však nutně jednat o „fyzické“ zařízení, data můžeme získávat například z kalendáře, poslaných SMS, uskutečněných hovorů atp.

Síť

Kvalita informací získaných z tohoto senzoru je závislá na kvalitě signálu a místním pokrytí. Tento senzor zároveň ovlivňuje kvalitu senzoru „Internet“ v případě, že není dostupné Wi-Fi připojení. Může tak docházet k absenci internetového připojení na základě nedostupnosti síťového pokrytí nebo jeho zhoršení v případě místní lokace. Například ve velkých městech je mnohonásobně kvalitnější pokrytí a internetové připojení díky dostupnosti lepší generace síťe. Těch existuje několik - v ČR je aktuálně nejlepší možná dostupná síť třetí generace (velká města).

API

Prostřednictvím bluetooth nebo internetu a nainstalovaných aplikací v našem zařízení bychom mohli docílit rozšíření dostupných interních senzorů o senzory externí, což by nám následně umožnilo přistupovat k informacím mimo „fyzické“ senzory aktuálního zařízení. Mohlo by se jednat například o externí zařízení typu výrobní linka, osobní automobil nebo monitor zdravotního stavu. Systém výrobní linky tak může poskytovat informace o

- upozorněních.
- závadách, selháních nebo problémech.
- místě výskytu některé události.
- zaměstnancích.
- servisních kontrolách.

Systém osobního automobilu může poskytovat informace o

- okolní teplotě.
- stavu nádrže.
- dojezdu.
- aktuální spotřebě.

Monitor zdravotního stavu může poskytovat informace o

- systolickém a diastolickém tlaku.
- hodnotách srdečního tepu.

1.3.2 Aktivita OS

Pokud bychom byli schopni sledovat aktivitu operačního systému, pomohlo by nám to upřesnit získaná data ze senzorů. V případě, že zařízení nedokáže detekovat dostupné Wi-Fi sítě, z nastavení systému jsme schopni vydedukovat, zda nejsou žádné v dosahu nebo uživatel manuálně vypnul Wi-Fi senzor. Důležité jsou pro nás především tyto oblasti:

- Životní cyklus běžících aplikací
- Nastavení systému
- Aktuální stav OS (stav baterie, síla signálu)

Možnosti získávání potřebných dat jsou však silně závislé na operačním systému, který zvolíme pro budoucí implementaci. Z tohoto důvodu nejsme nyní schopni určit, zda se nám podaří jednotlivé položky nějakým způsobem efektivně monitorovat.

1.3.3 Data ze senzorů

Na základě informací z dostupných senzorů a z aktivit OS budeme schopni získat data pro další zpracování, například:

- Poloha
- Pohyb
- Činnosti + pravidelnost
- Spouštěné aplikace
- Umístění zařízení (kapsa/taška/batoh atp.)
- Okolní prostředí (venku/vevnitř, světlo, hluk, teplota, vlhkost)
- Aktuální činnosti
- Zdravotní stav

Každá z výše uvedených položek bude přistupovat k informacím z několika odpovídajících senzorů a poskytovat je aplikaci. V závislosti na tom, kolik z nich bude aktuálně dostupných, jaká bude jejich spolehlivost a přesnost, pak bude aplikace vyhodnocovat, v jakém rozsahu se na tato data může spoléhat a jak s nimi dále pracovat (a případně je i archivovat pro pozdější využití).

Jak jsme si nastínili v předchozím odstavci, v rámci naší aplikace nás budou zajímat dva druhy dat. Zaprvé jsou to aktuálně reportované informace ze senzorů a zadruhé je nutné brát v potaz i informace dlouhodobého rázu (především periodicita). Jedná se tedy například o opakované činnosti/aktivity, přičemž data mohou pocházet ze záznamů v kalendáři, ale také ze zaznamenaných a pravidelně se opakujících činností/aktivit.

Takto získaná data bude následně popisovat vrstva „Sémantika dat“. Z tohoto důvodu musí být data popsána nějakým formálním jazykem, pro který budeme schopni tuto vrstvu definovat (například XML). Do budoucna musíme rozhodnout, jak ji budeme realizovat pro data aktuální (nebudou persistentně uložena) a dlouhodobá (budou persistentně uložena).

1.3.4 Sémantika dat

Popis (neboli sémantika) dat bude ve spojení s vrstvou „Data ze senzorů“ sloužit pro využití vrstvou „Interpretace dat“. Předpokládáme, že se v rámci návrhu budeme muset vypořádat s problémem rozličné fyzické interpretace dat aktuálních a dlouhodobých. Aktuální data v některých případech nebude nutné archivovat a proto musí tato vrstva v souvislosti s vrstvou „Data ze senzorů“ popisovat obě formy dat abstraktním způsobem tak, aby se s nimi dalo pracovat co nejuniverzálněji.

Pro samotnou implementaci této vrstvy předběžně počítáme s využitím projektu OWL ve verzi Lite, maximálně DL (jena.apache.org).

1.3.5 Interpretace dat

Nejvýše postavená vrstva ve stromové struktuře bude sloužit k samotné interpretaci získaných dat a od ostatních vrstev se liší tím, že bude popsaná procedurálním způsobem. Při tomto procesu se pravděpodobně setkáme s problémem, kdy se aplikace bude nacházet v tzv. nulovém bodě (nemáme k dispozici ještě žádná dlouhodobá data). Tento stav se budeme snažit eliminovat tak, aby aplikace mohla efektivně pracovat hned od začátku.

Okamžitě po startu bude aplikace schopná získávat data z dostupných senzorů a ty vyhodnocovat. Jak jsme zmínili výše, budou ji však chybět data dlouhodobá (periodicky se opakující) a pak především ta, která se

ze žádného senzoru nedozví, ale která jsou pro fungování aplikace stěžejní. Vyhodnocování informací, získaných ze senzorů, se musí z logiky věci řídit potřebami uživatele a k tomu nám pomůže vytvoření jeho profilu.

Při tomto procesu bude tedy nutné zjistit informace například o omezení, se kterými se uživatel potýká (slepota, hluchota, omezená pohyblivost atp.). Teprve na základě toho pak může aplikace korektně interpretovat získaná data. Uveďme si jednoduchý příklad s okolním prostředím. Uživatel postižený hluchotou bude okolní prostředí považovat za rušivé v případě nepříjemných světelných podmínek (ostré světlo, blikající světla atp.). To ovšem nutně nemusí být rušivé pro někoho, kdo je nevidomý. Tomu může naopak vadit vysoká hladina zvuku nebo nepříjemná frekvence.

Požadovaného výsledku bychom mohli docílit tím, že při prvním spuštění necháme uživatele zodpovědět několik základních otázek, které nám pomohou se „odpíchnout“ a vytvoříme tak pro aplikaci lepší startovní pozici.

Proces interpretace

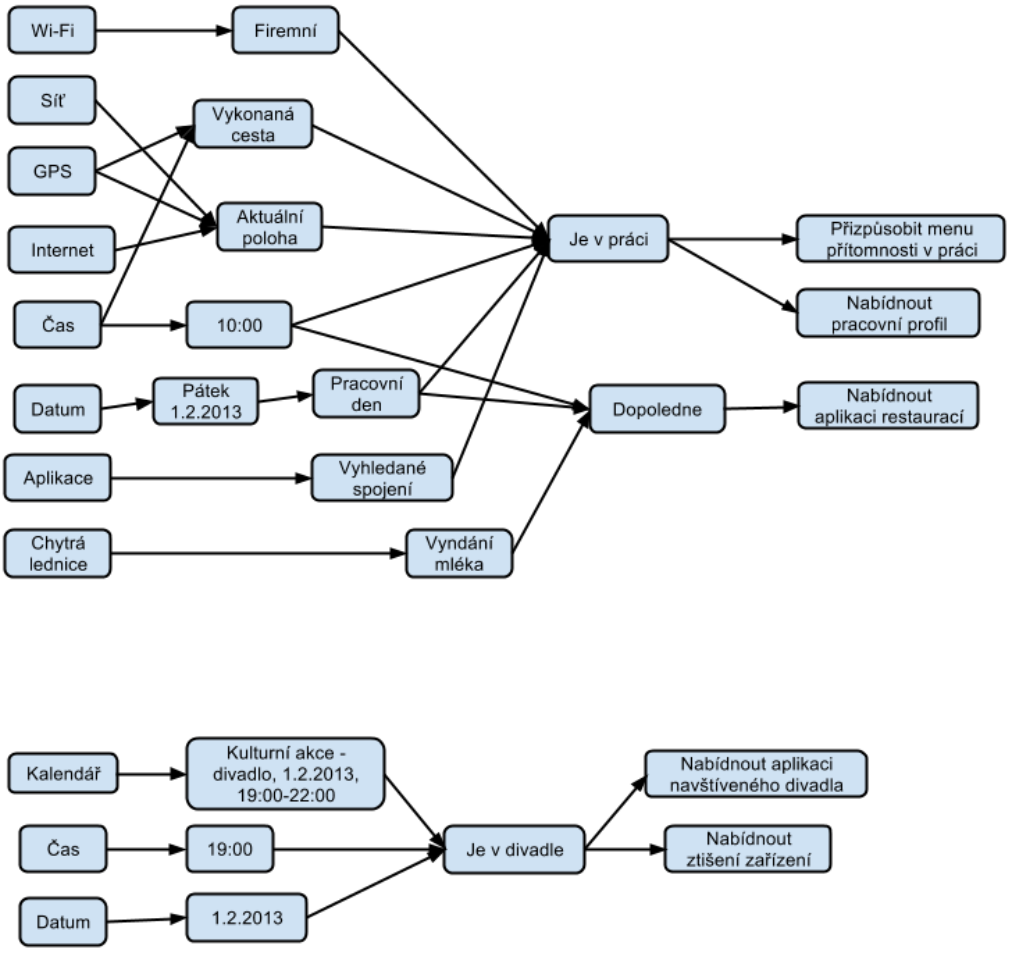
Při pohledu na diagram 1.6 nacházející se na straně 30 si můžeme ilustrovat interpretaci kontextu a pravidelných aktivit v ideálním případě. Je z něho jasně patrné, jak rozsáhlý a velmi provázaný může konečný produkt být a je téměř jisté, že v rámci této práce bude ve výsledku realizována pouze určitá podmnožina takto komplexního problému. Důvodů je hned několik - zaprvé si nejsme jisti, zda a jak dalece bude vybraná platforma vůbec schopna takto definované požadavky pokrýt a za druhé je zřejmé, že rozsah takového řešení přesahuje časový rozpočet vymezený pro tuto práci.

Procesy rozpracované touto formou nám však pomohou lépe uchopit celý problém a uvědomit si, jak bychom se s ním měli potýkat během vývoje. Zároveň nám umožní rozmýšlet finální řešení v takové podobě, aby jeho návrh odpovídal požadavkům na budoucí snadné a přehledné rozšiřování.

1.4 Sémantický model

Sémantický model si nejlépe popíšeme pomocí diagramu 1.7 na straně 31, na kterém je znázorněna struktura OWL modelu. V něm se vyskytují třídy (zvýrazněny tučně), jejich instance (ostatní) a vztahové relace (plná šipka - dědičnost, čárkovaná šipka - vztah) a je rozdělen na tři části - první popisuje třídy a jejich instance, které jsou pro naši aplikaci pevně stanoveny a v současném stavu se nepočítá s jejich modifikací.

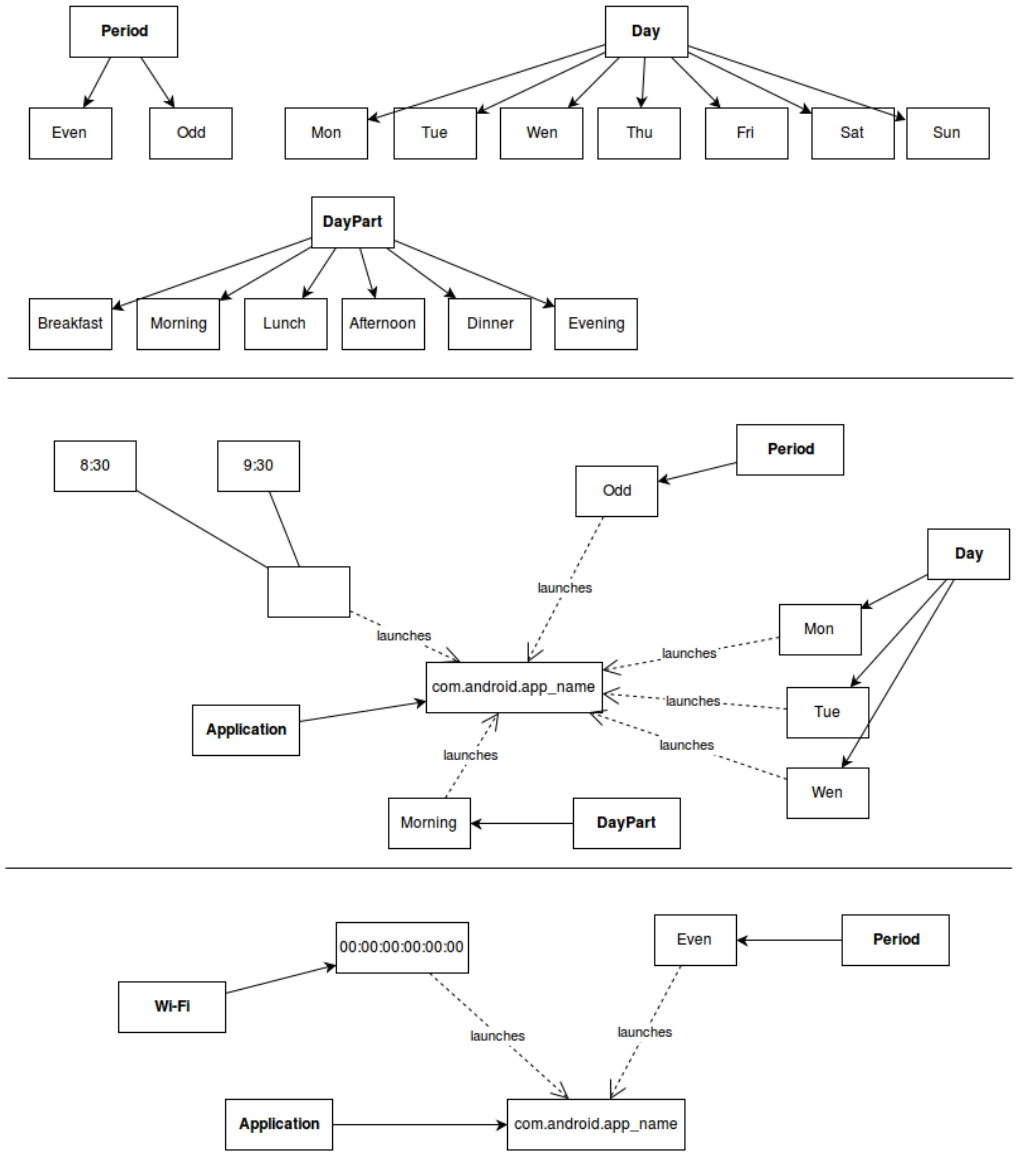
Druhá část už popisuje konkrétní strukturu uložené pravidelné aktivity a je na ní vidět její typ (com.android.app_name) a události, které ji spouštějí



Obrázek 1.6: Diagram možné kontextové interpretace.

- dny v týdnu, časové období, část dne a týdenní perioda. Třetí část se pak zabývá odlišnou strukturou, kdy spouštění aplikace obvykle probíhá v době, kdy je zařízení připojeno ke specifické Wi-Fi a to vždy v lichém týdnu.

Protože bychom v rámci našeho projektu naplno nevyužili všech možností, které OWL model nabízí, a zároveň se obáváme zbytečného nárůstu výpočetních prostředků vzhledem ke komplexnosti a velikosti dostupných reasonerů, které navíc nejsou optimalizovány pro mobilní zařízení, jsme se proto rozhodli převést ho na model v rámci relační databáze. Jednotlivé tabulky budou podrobně rozvedeny v kapitole 1.5 na straně 32.



Obrázek 1.7: Sémantický model.

1.4.1 Period

Z provedených uživatelských rozhovorů vyplývá, že drtivá většina participantů má pravidelný týdenní cyklus. V rámci dosažení univerzálnější použitelnosti naší aplikace jsme se však rozhodli brát v potaz cyklus čtrnáctidenní vzhledem k tomu, že se v životě velké části populace kdekoliv na světě velmi často vyskytuje. Studenti se s tímto systémem sudých/lichých týdnů setkávají již od základní školy. Tomu se navíc, především v nižších ročnících, obvykle přizpůsobují i dospělí. Naplno pak dochází k jeho uplatnění při studiu na vysokých školách nebo v rámci pracovní doby zvané „krátký, dlouhý týden“.

Tato třída tedy bude obsahovat dvě pevně stanovené hodnoty Even (sudý) a Odd (lichý).

1.4.2 Day

Třída reprezentující jednotlivé dny v týdnu s neměnými hodnotami Mon (Monday - pondělí), Tue (Tuesday - úterý), Wen (Wednesday - středa), Thu (Thursday - čtvrtek), Fri (Friday - pátek), Sat (Saturday - sobota) a Sun (Sunday - neděle).

1.4.3 DayPart

Při stanovení pevného rozložení dne do několika stanovených částí jsme vycházeli z běžného rozdělení tak, jak ho zná drtivá většina uživatelů na celém světě. Námi zvolený cyklus dne začíná hodnotou Breakfast (snídaně), následuje Morning (dopoledne), Lunch (oběd), Afternoon (odpoledne), Dinner (večeře) a končí Evening (večer). Tyto údaje nebude schopna naše aplikace zjistit jiným způsobem, než přímou interakcí s uživatelem. Budeme se tedy muset o těchto hodnotách informovat a zároveň zajistit, aby se tento postup pro něj nestal kontraproduktivním.

1.5 Model datového skladu (ER model)

Pro persistentní ukládání dat bude v rámci naší aplikace ideální využití některého z dostupných databázových strojů. Model datového skladu slouží k vytvoření abstraktního popisu databázové struktury a vztahů mezi jednotlivými entitami v ní uložených. Díky tomu je zaručena variabilita a možnost volby konkrétního stroje až ve fázi implementace.

Entitu si můžeme představit jako určitou abstrakci nad všemi konkrétními záznamy dané tabulky, můžeme ji tedy tzv. „instanciovat“ (naplnit

ji) jakýmkoliv záznamem v tabulce. Název jednotlivých tabulek je vždy reprezentován názvem entity v jednotném čísle. V rámci jedné tabulky sice bude postupem času přibývat jednotlivých záznamů a mohlo by se tedy spíše nabízet pojmenování tabulky množným číslem, avšak každý záznam (tedy řádek) v tabulce reprezentuje pouze jednu instanci dané entity. Pomocí unikátního indetifikátoru si tedy saháme právě na ni.

Druhým důvodem pro toto řešení je, že tabulky bývají často v aplikaci mapovány na jednotlivé modely (tedy třídy, reprezentující entity pro potřeby aplikace) pomocí takzvaného ORM. V tomto případě pak tato třída tvoří analogii s entitou v databázi a reprezentuje abstrakci ve formě výše uvedené třídy programovacího jazyka.

Diagram modelu datového skladu1.8 na straně 34, metodiku jeho plnění a strukturu jednotlivých tabulek si dále blíže popíšeme v následujících kapitolách.

1.5.1 Plnění a správa datového skladu

Primárním úkolem našeho datového skladu bude uchovávání informací o uživatelských akcích (tedy něco velmi podobného systémovému logu - aktuální snímek stavu telefonu a operačního systému). Důležité je si uvědomit, že pokud se rozhodneme vytvořit nový logovací záznam, budou do databáze uloženy všechny atributy, které sledujeme (neuloží se tedy pouze poloha, ale i okolní Wi-Fi, spuštěné aplikace atp.). Předpokladem této práce je, že se v těchto datech budou nacházet vzory a opakující se aktivity, na jejichž základě se bude naše aplikace schopna dále učit. Algoritmus učení v sobě bude zahrnovat vyhledání těchto pravidelností a jejich následné vyexportování do připravených tabulek, aby s nimi interpretační vrstva mohla pohodlně pracovat.

Při plnění a správě databáze registrujeme několik problematických oblastí, pro které je potřeba najít vhodné řešení tak, aby měl výsledný produkt co nejlepší vlastnosti a univerzální chování.

Dvojitý původ dat

Data, která budeme schraňovat pro pozdější zpracování budou pocházet ze dvou typově rozdílných zdrojů. Některá budou z podstaty věci o své změně dávat vědět sama (změna polohy), u ostatních musíme jejich zaznamenání řídit sami v pravidelných intervalech (seznam běžících aplikací). Na první pohled se zdá, že by se mělo jednat o dva různé algoritmy.

Tomuto implementačnímu pojetí se ve skutečnosti velmi pravděpodobně nevyhneme, nicméně tento postup by v důsledku vedl k jejich vzájemné

Obrázek 1.8: Model datového skladu (ER model).

kolizi v případě, že měnící se poloha by inicializovala ukládání do databáze stejně jako pravidelný časovač. Toto chování by samotné samozřejmě nevedlo k chybě aplikace, je ale nežádoucí vzhledem k tomu, že by docházelo k opakovanému záznamu identické informace a tím zbytečnému zahlcování databáze.

Proto bude nutné oba algoritmy určitým způsobem sjednotit pod nějaký nadřízený prvek, který bude nad jejich součinností dohlížet a efektivně usměrňovat jejich fungování. V současné chvíli se jeví jako nejlepší takové řešení, že bude časovač nastaven na konstantní hodnotu a postupně odkládán, pokud dostatečný přísun informací z prvního typu zdroje zaručí uspokojivé plnění databáze daty.

Během implementace plnění databáze pomocí výše zmíněného časovače bude nutné se vypořádat s případem, kdy nedochází k činnosti ze strany uživatele. Jedná se především o dobu spánku, ale tento problém se dá zobecnit na všechny případy, kdy není mobilní zařízení aktivně využíváno. Vezmemeli v úvahu dobu, kdy průměrný uživatel spí, jedná se obvykle zhruba až o jednu třetinu dne (myšleno dvacetičtyřhodinový cyklus). Pokud bychom se tímto nezabývali, ukládali bychom do databáze relativně velké množství dat zcela zbytečně.

Při plnění databáze by tedy bylo vhodné reflektovat stav předchozího záznamu. Jestliže se liší jen nepatrně, není zapotřebí provádět nový záznam. Definici míry diference bude teprve potřeba stanovit, pro začátek může být triviálním nastavením rozdílný čas - tedy pokud se aktuálně ukládaný a nejnovější uložený záznam liší pouze v čase, není potřeba nový vytvářet.

Redundance dat

Při plnění databáze se zcela jistě setkáme s jevem, kdy bude během vytváření nových logovacích záznamů jedna (či více) hodnot konstantních v čase. Bude se například měnit poloha, ale zvukový profil nebo stav GPS přijímače bude stále stejný. Pokud bychom v takovém případě ukládali veškeré informace, bude naše databáze zbytečně zahlcována redundantními daty. Proto bychom se v této kapitole měli zamyslet nad tím, zda dokážeme vymyslet nějaké efektivní řešení tohoto problému, případně zda takové vůbec existuje.

Jeden způsob, který se nabízí, je zaznamenávat pouze změny hodnot. Pokud by se tedy měnila poloha, ale zvukový profil by byl ve všech případech stejný, došlo by k zaznamenání této informace (zvukového profilu) pouze v případě zjištění, že byl telefon přepnut do jiného režimu. V ostatních případech by tento údaj vůbec nebyl přítomen a nepřímo by tak odkazoval na starší záznam (nejbližší změnu v minulosti). Zřejmou výhodou

je odbourání zátěže databáze neměnícími se daty, která by se tím pádem vůbec neukládala. Naproti tomu je ovšem nevýhodou snížení dostupnosti informací. Pokud bychom si chtěli zjistit stav mobilního zařízení třeba pro určitý den a hodinu, bylo by nutné (v rámci vybraného záznamu) hledat pro všechny nepřítomné hodnoty jejich poslední změny v minulosti, což by ve výsledku mohlo značně zvýšit komplikovanost a snížit rychlost a flexibilitu takových operací.

Další možností by bylo pojmout uspořádání databáze řekněme klasickým relačním způsobem. V rámci každé tabulky by tedy byly uloženy pouze známé (v minulosti zaznamenané) hodnoty a každý nový logovací záznam by obsahoval odkazy na ty, které by v danou chvíli odpovídaly aktuálnímu stavu zařízení (v případě, že by požadovanou hodnotu algoritmus v databázi nenašel, vytvořil by pro ni nový záznam). Tento postup by měl ve výsledku mnoho výhod. Pokud zjišťujeme stav mobilního zařízení pro určitý den a hodinu, máme okamžitě po ruce všechny informace, které si pro tyto účely do databáze ukládáme. V případě, že bychom si chtěli zjistit, kdy a za jakých hodnot je prováděno například spouštění určité aplikace, nám toto uspořádání, oproti možnosti z předchozího odstavce, celý proces usnadní (a obdobně pro ostatní sledované hodnoty). Nevýhodou tohoto řešení bude naopak nutnost u některých vazeb zavést určité „spojovací“ tabulky (tedy vzroste objem ukládaných dat), které budou udržovat informace o tom, jaké hodnoty byly uloženy ke konkrétnímu záznamu (v grafu nejsou znázorněny). To by však ve spojení s efektivním čištěním databáze nemělo představovat fatální problém.

Protože preferujeme přístup k informacím v databázi právě prostřednictvím klasického relačního způsobu a vzhledem k obavám z nárůstu časové složitosti ve spojení s nutností implementace algoritmu, který by v prvním případě dohledával relevantní informace pro většinu požadovaných záznamů, jsme se rozhodli přiklonit se k druhému řešení.

Čištění databáze

Vzhledem k předpokládanému objemu ukládaných dat je nezbytné vytvořit určitou strategii pro čistění a promazávání databáze. Nesmí se tak ovšem dít na úkor získaných informací - v žádném případě tedy nesmí docházet ke zkreslování pravidelností v důsledku neopatrného promazávání starších dat. Sestrojit algoritmus, který by efektivně prováděl operace zmíněné v úvodní větě, bude jedním z nejobtížnějších úkolů, se kterými se budeme při implementaci potýkat.

Rozebereme-li si tento proces detailněji, měl by tedy tento algoritmus procházet nasbíraná data a hledat v nich vzory a pravidelnosti. Pokud by

měl dostatek informací, aby s jistotou mohl prohlásit zkoumanou aktivitu za pravidelnou, mohl by při dalších iteracích postupně začít odmazávat starší (přebytečné) záznamy o této aktivitě a díky tomu tak udržovat databázi v „minimalistickém“ stavu. Tím by zmenšil nároky na databázový stroj a zrychlil by případné další dotazy, které by už nemusely procházet víc dat, než je nezbytně nutné.

S přihlédnutím k faktu, že v této fázi stejně budeme hledat pravidelné aktivity, se zdá v současné chvíli jako nejlepší řešení nalezené vzory při této příležitosti rovnou vyexportovat (v kapitole 1.4 na straně 29 jsme zmínili důvody, které nás vedly k migraci od OWL řešení směrem k relační databázi) a tím dosáhnout efektivnějšího přístupu k nim. Je zcela zřejmé, že takový postup vyžaduje rozšíření naší aplikace o algoritmus, který by procházel již uložené pravidelné aktivity a porovnával je se záznamy v databázi, aby zajistil jejich aktuálnost - tedy že tato aktivita probíhá i nadále. Díky tomu bychom měli být schopni určitým způsobem odmazávat pravidelnosti v databázi, které se nám podařilo tímto procesem přesunout mimo logovací záznamy do speciálních tabulek. Jak efektivní tento způsob bude, záleží především na struktuře a provázanosti jednotlivých pravidelností.

1.5.2 Přehled tabulek datového skladu

Úkolem této kapitoly je popsat jednotlivé tabulky vyskytující se na diagramu datového skladu. U každé z nich se nachází stručný popis a detaily jednotlivých atributů. Samotný graf je rozdělen na dvě části.

Logovací informace

První část grafu popisuje tabulky pro skladování logovacích informací a jejich atributů. Veškeré vazby jsou v zásadě nepovinné, jelikož se u většiny záznamů může stát, že nebude dostupná informace o Wi-Fi, BTS nebo třeba poloze.

LogRecord slouží jako určitá forma středobodu všech logovacích záznamů a je tedy provázaný se všemi tabulkami, které shromažďují informace o uživatelské nebo systémové činnosti. Vždy musí mít vazbu na záznam v tabulce Timestamp, který určuje časovou linearitu. Ostatní vazby jsou nepovinné (tedy k danému času nemusí existovat záznamy v ostatních tabulkách), protože není zaručeno, že tyto informace budou v danou chvíli dostupné. Pokud v následujících kapitolách zmiňujeme

sousloví "informace o aktuálních..." máme tím na mysli právě záznamy vztažené k tomuto časovému záznamu.

BTS obsahuje informace o BTS, ke které je telefon aktuálně připojen. V případě konkrétního záznamu se uchovává kód lokace a její ID.

WiFiNetwork obsahuje informace o aktuálních Wi-Fi sítích dostupných telefonu. V případě konkrétního záznamu se uchovává jméno Wi-Fi sítě a zda je k ní telefon připojen.

RunningApp obsahuje informace o aktuálně bežících aplikacích. V případě konkrétního záznamu se uchovává jméno spuštěné aplikace a údaj, zda byla v době jeho pořízení spuštěna či již běžela.

PhoneCall obsahuje informace o provedených telefonních hovorech. V případě konkrétního záznamu se uchovává číslo volaného.

SMS obsahuje informace o odeslaných textových zprávách. V případě konkrétního záznamu se uchovává telefonní číslo příjemce a text zprávy.

PhoneSettings obsahuje informace o aktuálním nastavení telefonu. Jedná se o profil, stav GPS senzoru, WiFi senzoru a Airplane módu. V případě konkrétního záznamu se uchovává, zda je zapnut zvuk a/nebo vibrace a zda jsou senzory zapnuty či nikoliv.

Timestamp obsahuje informace o aktuálním času, při kterém je pořízen záznam do tabulky LogRecord. V případě konkrétního záznamu se uchovává datum (počet sekund od začátku epochy (1.1.1970)), čas (počet sekund od půlnoci), název dne a perioda.

BatteryState obsahuje informace o aktuálním stavu baterie telefonu. V případě konkrétního záznamu se uchovává stav (v procentech).

Location obsahuje informace o aktuální geografické poloze telefonu. V případě konkrétního záznamu se uchovává zeměpisná šířka a délka, rychlost, nadmořská výška a přesnost.

Pravidelné aktivity

Druhá část grafu pak popisuje již vyexportované pravidelné aktivity a jejich atributy. Veškeré vazby jsou v zásadě nepovinné, může se stát, že jedinou dohledatelnou pravidelností bude například pouze připojená BTS nebo Wi-Fi.

App obsahuje informace o aplikaci, jejíž spouštění bylo zaznamenáno jako opakující se ve spojení s některým z uchovávaných atributů. Jedná se, podobně jako u tabulky LogRecord, o určitý středobod všech záznamů o pravidelných aktivitách.

BTS_p obsahuje informace o BTS, ke které byl telefon připojen při spouštění dané aplikace. V případě konkrétního záznamu se uchovává její jméno.

WifiNetwork_p obsahuje informace o Wi-Fi, ke které byl telefon připojen nebo která byla dostupná při spouštění dané aplikace. V případě konkrétního záznamu se uchovává její SSID (jméno) a BSSID (identifikátor přístupového bodu).

Period obsahuje informace o časové periodě, ve které docházelo k pravidelnému spouštění dané aplikace. V případě konkrétního záznamu se uchovává její jméno.

Day obsahuje informace o dnech, ve kterých docházelo k pravidelnému spouštění dané aplikace. V případě konkrétního záznamu se uchovává jeho jméno.

DayPart obsahuje informace o části dne, ve které docházelo k pravidelnému spouštění dané aplikace. V případě konkrétního záznamu se uchovává jeho jméno.

Time obsahuje informace o čase. Můžeme si zde všimnout dvou vazeb s tabulkou App pojmenovaných "from" a "to". Tímto způsobem pokrýváme určité časové období, během kterého byla daná aplikace pravidelně spouštěna (nelze se spoléhat na fakt, že uživatel bude spouštět aplikace přesně v pevně daný čas, ale musíme se vyrovnat s jistou flexibilitou). V případě konkrétního záznamu se uchovává časový údaj (počet sekund od půlnoci).

Location_p obsahuje informace o poloze, ve které docházelo k pravidelnému spouštění dané aplikace. V případě konkrétního záznamu se uchovává její zeměpisná šířka a délka. Tento údaj (podobně jako čas) nelze brát dogmaticky, jelikož uživatel nebude stát vždy na tom samém místě, je tedy potřeba v rámci interpretace počítat s jistou, dopředu stanovenou, kruhovou odchylkou.

1.6 Správa datových zdrojů

V případě spravování datových zdrojů jsme se rozhodli pro podobný princip jaký preferuje Google Android například v případě získávání informací o poloze[32]. V zásadě se v tomto modelu setkáme se dvěmi součástmi, které si v následujícím textu blíže rozebereme. Kromě toho si detailně popíšeme vzájemnou spolupráci těchto součástí a také vlastnosti, které takto definovaná struktura má.

1.6.1 Manager

Manager (LocationManager[34]) je definován jako třída[24] a slouží k poskytování veškerých informací v rámci konkrétního zdroje stejně jako registraci modulů, které chtějí být upozorněny v případě nastalých změn. V našem případě může být celkové pojetí již definovaných Managerů (LocationManager, WifiManager atp.) zbytečně komplikované a naopak některou doplňkovou funkcionalitu nemusí obsahovat. Proto jsme se rozhodli toto rozhraní co nejvíce zjednodušit pomocí předsunutého prvku, který nám umožní i dodatečné úpravy tak, abychom dosáhli požadovaného chování. Aby mohl být vývojáři definovaný modul registrován pro odběr změn, musí splňovat specifikované požadavky. Zde přichází ke slovu druhá součást a tou je Listener.

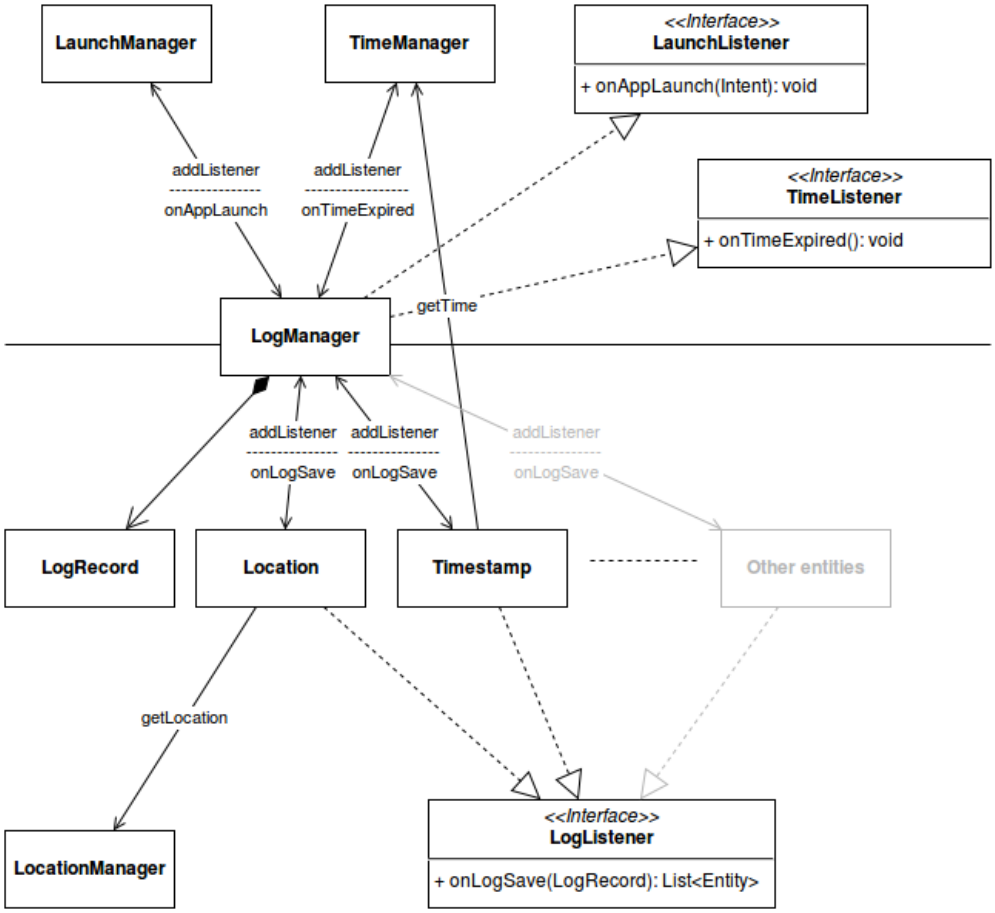
1.6.2 Listener

Listener (LocationListener[33]) je definován jako interface[41], který musí každý modul, usilující o registraci, implementovat. Díky tomu máme jistotu, že jsme nad všemi registrovanými moduly (listenery) schopni zavolat potřebné metody (například metodu spojenou s akcí “nastala změna pozice”). Zároveň tento způsob poskytuje dostatečnou volnost vývojářům, kteří mohou na nastalou situaci reagovat vlastní implementací těchto definovaných metod.

1.6.3 Princip

Proces fungování takového celku si můžeme ilustrovat na zjednodušeném (jedná se o výřez z celkového systému) diagramu 1.9 na straně 41 (část od vodorovné čáry výše). Vidíme, že prvek LogManager by chtěl být informován o změnách LaunchManageru (uživatelské spouštění aplikací) a TimeManageru (nastavení časovače).

Implementuje proto odpovídající interface (LaunchListener pro LaunchManager a analogicky TimeListener pro TimeManager) a pomocí metody



Obrázek 1.9: Výřez systému ilustrující princip spolupráce součástí Manager a Listener umožňující sledování změn a následné ukládání logovacích záznamů.

addListener se zaregistruje u každého prvku zvlášť. Ve chvíli, kdy dojde ke sledované změně (spuštění aplikace), LaunchManager projde všechny své LaunchListenery a na nich zavolá definovanou metodu onAppLaunch. LogManager pak na základě toho provede v rámci vlastní implementace této metody odpovídající akci, v tomto případě zarchivuje stav mobilního zařízení (uloží logovací záznam). Vlastnosti

Takto definovaná struktura nám dává výhodu vzájemné nezávislosti a obstojné flexibility prostřednictvím využití principu registrace Listenerů. Prvky tak mezi sebou nejsou pevně svázány, což nahrává budoucí recyklaci kódu, stejně jako jednoduché rozšiřitelnosti. Díky tomu navíc dosáhneme přehledné správy implementace. U každého modulu je zřejmé, kde

může být v současném stavu registrován (vzhledem k uvedenému seznamu implementovaných Listenerů) a Managery zase poskytují přehledně definovaný a jednoduchý přístup k přesně ohraničené části informací (neposkytují zbytečně nic navíc).

Neměl by tak být problém přidat během dalšího vývoje nové Managery a jejich Listenery. Zásah do stávajícího kódu je zřejmý, nicméně popsaný a jasně řízený. Z toho důvodu stačí, když se bude vývojář řídit výše specifikovaným návrhem a dodrží popsanou strukturu.

1.7 Ukládání logů

Požadavkem pro efektivní ukládání logovacích záznamů je dodržení flexibilní a vzájemně nezávislé struktury především z důvodu pravděpodobných změn (rozšíření) během budoucího vývoje. Proto jsme využili mírně upraveného principu popsaného v kapitole 1.6 na straně 40 (Manager - Listener). V následující kapitole si popíšeme proces jeho fungování a také v čem spočívají drobné modifikace tohoto systému a jaké nás k tomu vedly důvody.

Podíváme-li se na diagram 1.9 na straně 41, nejsou zde na první pohled zřejmé žádné procesní změny. Listenery se registrují obvyklým způsobem, mírně rozdílný je však postup při ohlášení změny. LogManager má přímý odkaz na hlavní databázovou entitu LogRecord, která v sobě obsahuje odkazy na mnoho dalších entit, které je potřeba společně uložit do databáze. Aby mohl LogManager entitu LogRecord naplnit, musel by vlastnit odkazy na všechny relevantní entity v databázi. Tento postup by ale odporoval požadavkům na nízkou provázanost a flexibilitu, proto jsme zvolili odlišný algoritmus.

Ve chvíli, kdy LogManager obdrží informaci o změně, vytvoří novou instanci entity LogRecord a tu rozešle všem svým Listenerům (ostatním entitám), kterých v současné chvíli evidujeme dva druhy. První určitým způsobem modifikuje LogRecord a to tak, že mu předá odkaz na svou instanci a LogManageru vrací prázdný odkaz (null). Druhý vytvoří instance spojovacích tabulek (nutné pro vazby many-to-many, například vazba LogRecord - RunningApp), vloží do nich odkaz na obě entity a vrací odkaz na seznam svých vlastních instancí (pokud chceme k LogRecordu evidovat tři běžící aplikace, potřebujeme tři vazby LogRecord - RunningApp). Neprázdné odkazy si LogManager uchovává a po projití registrovaných Listenerů uloží všechny nashromážděné entity najednou, včetně LogRecordu. Díky tomuto procesu je zaručeno, že do databáze se budou ukládat pouze korektně provázané a úplné záznamy.

Je zřejmé, že než dojde k uložení jednotlivých instanciovaných entit, je potřeba je naplnit relevantními daty. K tomu využijeme již nadefinované moduly z kapitoly 1.6 na straně 40, které kromě registrace k odběru změn poskytují zároveň přístup k relevantním datům.

1.8 Vyhledávání pravidelností

Aby měla naše aplikace z čeho čerpat data pro zobrazování odpovídajících aplikací na hlavní obrazovce, je potřeba v nasbíraných datech najít vzory pravidelností, nad kterými se pak budeme dotazovat. PeriodicActivityManager tak používá již několikrát zmiňovanou koncepci registrovaných modulů, pro tento účel mírně poupravenou. Všechny implementují rozhraní PeriodicSeeker a PeriodicActivityManager si udržuje jejich seznam.

Jak jsou postupně spouštěny, vyhledávají v databázi logovacích záznamů pravidelnosti spojené s jimi sledovaným atributem/atributy a ukládají je do databázových struktur. Jednotlivé moduly jsou na sobě nezávislé a díky tomu nezáleží na pořadí, ve kterém jsou případně spouštěny.

Vykonávat tuto operaci by bylo vhodné ve chvíli, kdy uživatel neprovádí na mobilním zařízení žádnou činnost (například spí). Jednou z možností je nechat chvíli spouštění zmíněného procesu pouze na něm a naučit se, kdy je nejlepší tuto operaci provádět. V dnešní době už však není pravidlem vypínat mobilní zařízení na noc, především z důvodu jejich funkčnosti (u vypnutého telefonu například nefunguje budík), ale obvykle se pouze přepínají do „letadlového“ módu. V tomto stavu nepřijímají žádný signál, ale celou noc jsou zapnutá (otevírá se tu tedy možnost pro vykonávání operací na pozadí) a díky tomu bychom mohli spouštění naší operace registrovat právě na tuto událost. Druhé řešení se nám zdá být v současné době postačující, přičemž navíc ještě uvažujeme o periodickém provádění, například každých deset nebo dvacet hodin, pro případ, že by uživatel tento mód nepoužíval.

1.9 Filtrování pravidelností

Hlavním smyslem naší aplikace je v daný okamžik nabídnout uživateli co nejlepší seznam aplikací a proto musíme provádět určitou formu selekce dostupných pravidelných záznamů. Abychom zachovali nízkou provázanost a do budoucna zvýšili komfort rozšiřování, vytvořili jsme interface Filter, který implementují všechny třídy, jenž umožňují filtraci seznamu aplikací (ten je jim předán jako parametr). PeriodicActivityManager uchovává jejich

seznam, který při každé obměně uživatelského rozhraní prochází a u filtrů spouští metodu zodpovědnou za provedení odpovídající úlohy.

Díky tomu jednotlivé filtry provedou potřebné operace, aniž by PeriodicActivityManager musel vědět, jaký vlastně aktuálně spouští. Nutnou podmínkou tohoto přístupu je, že na sobě musí být zcela nezávislé a z toho důvodu ponechávají v seznamu jednak aplikace, které odpovídají jejich kriteriím, ale zároveň ty, které nemají žádnou informaci o filtrovaném atributu (to znamená, že na tomto atributu nezáleží). Pokud tedy například filtr aktivních Wi-Fi připojení zjistí, že hodnota odpovídá aktuálnímu kontextu, aplikaci v seznamu ponechá, stejně jako tu, která ji má blíže nespecifikovanou (typicky null). Prostřednictvím tohoto mechanismu je zajištěno, že ze seznamu nebudou odstraněny aplikace nezávislé na právě testovaném kritériu.

1.10 Uživatelské rozhraní

Během implementace našeho projektu se z časových důvodů a vzhledem ke stanoveným cílům nebudeme zabývat hlavní viditelnou součástí, kterou je Launcher. Existuje několik jeho v principu podobných verzí s dostupným zdrojovým kódem, ze kterých vybereme tu nejvhodnější a využijeme ji v našem projektu. Nám tato volba ušetří čas a umožní se soustředit na hlavní cíle projektu, navíc díky tomu bude zaručeno, že nebudeme znovu vynalézat kolo a nutit uživatele zvykat si na něco jiného než co poměrně důvěrně zná a již se s tím naučil pracovat. Jak budeme zobrazovat seznam relevantních aplikací a komunikovat s uživatelem si popíšeme v následujících kapitolách.

1.10.1 Seznam aplikací

Upravovat stávající uživatelské rozhraní tedy budeme pouze minimálně, především z důvodu změny domovské obrazovky, na kterou chceme umístit seznam aplikací. Ten se bude měnit podle toho, jaký bude aktuální kontext, ve kterém se mobilní zařízení (uživatel) nachází, ve spojení s již naučenými pravidelnostmi. Registrujeme několik oblastí, které musíme pokrýt a naplánovat jejich vhodné řešení.

V reakci na měnící se aktuální okolní situaci bude potřeba obnovovat pořadí zobrazených aplikací. Jednou z možností je upravovat seznam aplikací při jakémkoliv vnějším podnětu (změna času, připojení k Wi-Fi). Jeho nevýhodou je ovšem fakt, že by se systém snažil o změnu pořadí i v případě, že uživatel si právě vybírá požadovanou aplikaci. Pokud bychom tuto operaci omezili tím způsobem, aby běžela pouze když je na popředí

jiná aplikace, zbytečně bychom mrhali výpočetními zdroji a energií. Tento nedostatek nám však vyřeší způsob, kdy bude k obnovení pořadí docházet v případě, že se naše aplikace vrací na popředí, což umožňuje implementace metody onResume(). Díky tomu budeme kontrolovat aktuálnost pořadí pouze v případě, že se na něj uživatel chystá přejít.

V tomto případě ale může nastat situace, kdy uživatel setrvává na hlavní obrazovce, nicméně nevyvíjí žádnou aktivitu (telefon leží na stole a uživatel nad nečím přemýšlí). V takovém případě by bylo vhodné i přesto, že se naše aplikace nachází v popředí, obnovit pořadí aplikací a nabídnout tak uživateli lepší možnosti vzhledem k aktuálnímu kontextu. Pro vyřešení tohoto problému bude vhodné využít služeb časovače (podobně jako u automatického ukládání logovacích záznamů), který po definované době, během které systém nezaregistruje činnost uživatele, mu pošle zprávu o tom, že má provést revizi stávajícího pořadí.

Tím jsme vyřešili obnovu seznamu aplikací, které ale bude potřeba na hlavní stránce určitým způsobem řadit a toto pořadí se bude měnit v čase. Opět se zde nabízí několik možností, jak přechody mezi jednotlivými stavy vyřešit. Triviálním řešením by bylo se v daném okamžiku podívat, které aplikace je aktuálně relevantní zobrazit, a seřadit je, například abecedně. Vzhledem k tomu, že v současné době nemáme k dispozici žádná data, podle kterých bychom mohli usuzovat na velikost průměrně zobrazovaného seznamu, nelze ani rozumně zhodnotit efektivnost tohoto přístupu.

Sofistikovanějším způsobem řazení by bylo se zaměřit na stanovení priorit, které jednotlivé aplikace v seznamu mají, a na jejich základě v něm pořadí upravovat. Je však nutné nalézt metodu, pomocí které převedeme vlastnosti jednotlivých pravidelných záznamů na jediné číslo - prioritu. Když se na celý problém podíváme detailněji, tak zjistíme, že to není zcela jednoduchá operace. Máme-li aplikaci, která je spouštěna během jednoho dne v týdnu, a druhou, která je spouštěna v přítomnosti specifické Wi-Fi, nejsme v případě, že jsou obě podmínky splněny, schopni určit, která z nich by měla mít prioritu vyšší. Proto se v současné chvíli omezíme na určování priorit pouze u těch aplikací, které mají jasně vymezené časové období (například od 8:00 do 9:00), a ostatním ponecháme tu nejnižší.

1.10.2 Komunikace s uživatelem

Při celkovém pohledu na vyvíjenou aplikaci je zcela zřejmé, že některé informace nejsme schopni zjistit pouze prostřednictvím pasivního sledování uživatelské činnosti. Bez aktivních podnětů tak například těžko zjistíme rozložení dne (jaké časové rozmezí pro uživatele představuje snídaně, dopoledne nebo oběd) nebo jak dané pořadí aplikací odpovídá jeho aktuálním potře-

bám. Z toho důvodu jsme nuceni nalézt formu, prostřednictvím které budeme s uživatelem komunikovat, tyto informace od něj získávat nebo si nechávat potvrzovat správnost našeho úsudku.

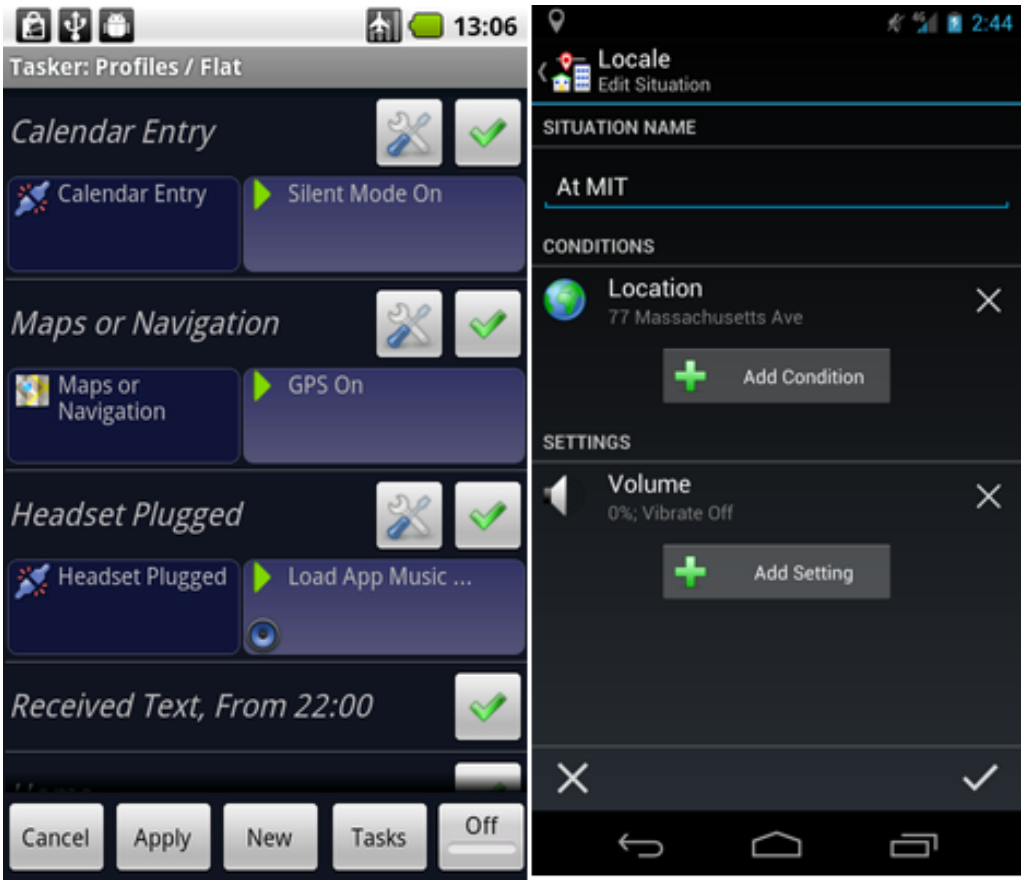
Největším problémem této oblasti je, že se tento způsob shromažďování informací stane pro uživatele otravné nebo ho bude zdržovat při práci. Z jeho pohledu není nic horšího, než když si potřebuje rychle najít spoj a naše aplikace se bude snažit z něj vymámit informace o spokojenosti s aktuálně nabízeným seznamem nebo v jaké se aktuálně nachází fázi dne. Z uvedeného příkladu vyplývá, že si musíme důkladně rozmyslet, kdy budeme po uživateli chtít doplňující informace a když už k tomu dojde, tak především zvolit co nejstručnější formu a dát mu například vybrat z několika nejpravděpodobnějších možností.

Jedním z možných řešení je komunikovat s uživatelem v zásadě pasivně a nechat rozhodnutí, kdy má čas věnovat se zodpovídaní našich dotazů, na něm. Tento způsob by se dal realizovat zobrazením upozornění ve stavové liště, která má ovšem tu nevýhodu, že pro většinu uživatelů může být tato informace obtížně dostupná/zastrčená. Bylo by také nutné shromažďovat více sad dotazů, které bychom pak uživateli předložili a díky tomu by opět mohlo dojít k tomu, že uživatele zahltíme a tím pádem velmi rychle odradíme z důvodu časové zátěže.

Toto provedení pravděpodobně nebude ideální, i když vezmeme v úvahu, že se bude počet dotazů do budoucna pravděpodobně snižovat z důvodu rozsáhlejšího naučení systému. S přihlédnutím k zamýšlené formě sběru informací by bylo vhodnější se ptát po každé provedené operaci. Po spuštění aplikace by se tak uživateli zobrazilo okno s otázkou a po zodpovězení by vše pokračovalo v započatém procesu. Pro zvýšení komfortu by bylo vhodné se uživatele zeptat, zda má v tuto chvíli čas se nám chvíli věnovat. I tak ovšem hrozí, že ho budeme zbytečně zdržovat a proto bychom mohli dotaz raději přesunout na okamžik, kdy se vrací naše aplikace na popředí a je tedy pravděpodobné, že uživatel svou aktuální práci ukončil a mohl by na nás mít více času. V současné chvíli se nám zdá toto řešení jako nejvhodnější.

1.11 Veřejné API a podobné aplikace

Vzhledem k faktu, že naše aplikace bude sbírat různá data o uživatelské činnosti a stavu v čase, nabízí se myšlenka k jejich dalšímu využití a poskytnutí třetím stranám. Jak si ukážeme v následujících odstavcích, existuje mnoho aplikací zabývajících se řešením určitých podoblastí našeho problému, kterým by tyto informace mohly dobře posloužit a zlepšit jejich uživatelský komfort. Jako příklady poskytovaných dat si můžeme uvést aktuální



Obrázek 1.10: Obrazovky aplikací Tasker[63] a Locale[49] (zleva).

část dne (dopoledne, odpoledne atd.), typ týdne (sudý, lichý) nebo místo (v práci, doma atd.).

První z aplikací je Tasker[63], který umožňuje uživatelům zvýšit produktivitu zařízení tím, že si mohou nastavit různé úlohy (Tasks) sestávající se z množiny akcí, které jsou spojeny s určitým kontextem (připojení sluchátek, spuštění konkrétní aplikace atd.). Na první obrazovce 1.10 na straně 47 vidíme několik příkladů nadefinovaných úloh. Jejich rozsáhlejší (ne však úplný, jak uvádějí autoři) seznam pak nalezneme na jejich domovské stránce (citováno výše). Zajímat by nás mohla například položka

```
during the night, turn on airplane mode to conserve
battery/reduce radiation, but turn it off every 15 minutes
to check for SMS/voicemail.
```

Pojem „night“ je vcelku široký, v rámci týdne se může podstatně lišit.

Zde by tato aplikace mohla využít naši veřejnou API a zjistit si, jaká časová hranice vymezuje noc (například pro dnešní den) a podle toho aplikovat vytvořené pravidlo flexibilně. Uživatel by jich tak nemusel nastavovat několik podobných pro různé dny (s odlišnými časy), ale stačilo by mu specifikovat pouze jedno univerzální.

Další aplikací, kterou bychom letmo zmínili, je Atooma[20]. Staví na jednoduchém konceptu, který je podobný programovacímu bloku podmínka (pokud platí → udělej něco, tedy IF → DO). Díky tomu je uživateli dovoleno v části IF specifikovat množinu podmínek, která když je splněna, vykonají se následně definované akce v části DO. Aplikace by mohla využívat naši API podobným způsobem jako v předchozím odstavci a umožnit tak uživateli definovat například

```
IF night DO airplane mode on.
```

Locale[49] je poslední aplikací, kterou jsme vybrali do tohoto přehledu. Pracuje na podobném principu, jak je vidět na druhé obrazovce 1.10 na straně 47. Uživatel definuje množinu podmínek a na jejich základě pak požadovanou změnu nastavení zařízení. Vidíme zde hned pěkný příklad využití naší API. Uživatel pracuje na několika místech a na všech chce aplikovat totožné nastavení (například ztlumit zvuk). Nyní musí v aplikaci ručně přidat všechna umístění, kde chce toto pravidlo aplikovat. Pokud by však aplikace spolupracovala s naší API, stačilo by mu vybrat pouze pojem „Work“. Pokud by se v budoucnosti seznam takových míst modifikoval, nemusel by zdlouhavě měnit všechna nadefinovaná pravidla, ale vše by se provedlo automaticky.

Z uvedených příkladů je patrné, že většina aplikací obsahuje velmi těsné vazby a nedovoluje mnoho flexibility. Tento stav by se mohl díky spolupráci s naší API o něco zlepšit.

1.12 Verifikace a testování

V rámci návrhu verifikace a testování našeho projektu se zaměříme na několik hlavních oblastí. Jedná se o návrh (model), implementaci a uživatelské rozhraní (UI). V případě modelu nás bude zajímat, zda to, jak je navržen, plně pokryje potřeby, které na něj kladou situace, jenž jsme si stanovili v rámci definice problému, ale především v rámci následně vytvořeného scénáře. Ideální tedy bude znovu si rozebrat samotný scénář a na základě něho kontrolovat, že systém je schopen sbírat potřebná data, zpracovat je a následně je zobrazovat uživateli očekávaným způsobem.

Samotnou implementaci lze testovat několika způsoby. Prvních z nich je testování menších a navzájem nezávislých jednotek, tzv. unit testing. Velmi podstatné budou však v našem případě následné uživatelské testy, které nám pomohou nasbírat různorodá data od sledované skupiny. Tento způsob je však v našem případě velmi náročný na lidské zdroje a čas z důvodu podstaty řešeného problému - nasbírání potřebných dat může trvat klidně i měsíc a déle, než bude výstup aplikace rozumně hodnotitelný a nasbíraná data porovnatelná.

Při ověřování uživatelského rozhraní nás bude zajímat především fakt, zda se nám podařilo jej navrhnout a implementovat tak, aby splňovalo specifické potřeby. To ověříme opět nejlépe v rámci uživatelského testování, ale v tomto případě nám tento proces nezabere tak rozsáhlé časové období. Zaměřovat se budeme hlavně na stanovené uživatelské skupiny, zejména zrakově postižené. Především budeme zjišťovat úroveň uživatelského komfortu a jednoduchosti při práci s ním, tedy zda je uživatelské rozhraní možno intuitivně ovládat bez vizuálního kontaktu a zda poskytuje zpětnou vazbu (hlasovou, vibrační atp.) na akce provedené uživatelem.

Kapitola 2

Realizace

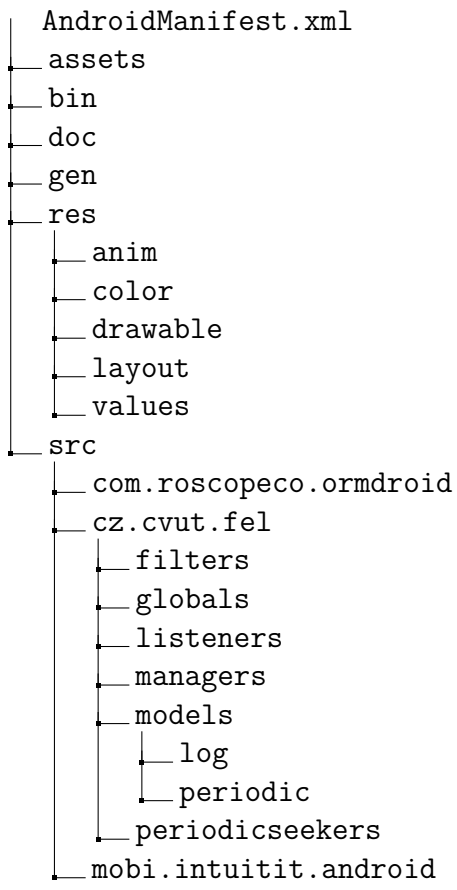
V předešlých kapitolách tohoto dokumentu jsme si definovali čím se bude tato práce zabývat, stanovili jsme si cíle, kterých bychom chtěli dosáhnout, vybrali nejvhodnější platformu, vytvořili návrh a specifikovali řešení nejdůležitějších oblastí, které se budou v naší aplikaci vyskytovat. Nyní se nacházíme ve fázi, kdy přecházíme k samotnému popisu implementace, který by ale nebyl možný bez předchozí práce. Bez důkladně provedeného návrhu řešení a analýzy by tato kapitola byla naprosto bezpředmětná a celkový projekt by mohl snadno skončit zbytečným neúspěchem.

V této kapitole si přiblížíme některé zajímavé části implementace naší aplikace. Vysvětlíme si strukturu Android projektu tak, jak ho generuje Eclipse IDE. Probereme komunikaci s databází, výběr vhodného Launcheru a zmíníme některé zajímavé části implementace. Na závěr celé kapitoly si uvedeme, které externí knihovny jsme použili pro řešení typických implementačních problémů.

2.1 Adresářová struktura

Jak nadpis podkapitoly napovídá, popíšeme si zde strukturu Android projektu [61] tak, jak je generován při založení nového projektu v Eclipse IDE (diagram 2.1 na straně 52) a zároveň se zaměříme na popis struktury adresářů naší aplikace.

AndroidManifest.xml je XML soubor, který uchovává informace o Android aplikaci, jako jsou práva potřebná pro fungování této aplikace (například právo zapisovat na filesystém, právo pro přístup na internet) nebo seznam všech aktivit v systému. Je to zcela nezbytný soubor



Obrázek 2.1: Adresářová struktura Google Android projektu

pro každý Android projekt. Pokud tento soubor chybí nebo je v něm chyba, aplikace se nespustí, popřípadě bude nestabilní.

assets/ má podobný význam jako adresář res, ale zdroje v něm jsou skladovány v „syrovém“ stavu a musejí být do aplikace načítány jiným způsobem.

bin/ obsahuje přeložené zdrojové soubory a instalační balíček soardroid.apk

doc/ obsahuje vygenerovanou dokumentaci pomocí nástroje Javadoc

gen/ obsahuje soubor R.java, který je generován automaticky a jsou v něm ukládány odkazy na externí zdroje uložené v res.

res/ je adresář obsahující externí zdroje Android aplikace, které jsou pak přístupné ve zdrojovém kódu díky souboru R.java.

res/anim je adresář s XML soubory obsahujícími definici animací.

res/color je adresář s XML soubory obsahujícími definici barev.

res/drawable je skladiště obrázků používaných v aplikaci. V některých případech mohou přítomny adresáře s příponou -ldi, -mdi a -hdi, které pak obsahují mutace zdrojů podle rozlišení.

res/layout obsahuje XML soubory s definicemi šablon pro jednotlivé aktivity. Jeden soubor obsahuje vždy jednu šablonu.

res/values obsahuje XML soubory s texty, barvami atp. Důvodem pro jejich externí skladování je pak snadnější internacionalizace [15].

src/ je poslední a nejdůležitější složka, jsou v ní totiž obsaženy veškeré zdrojové kódy aplikace.

src/com.roscopeco.ormdroid obsahuje zdrojové kódy knihovny pro ORM přístup k databázi.

src/cz.cvut.fel obsahuje zdrojové kódy naší aplikace.

src/cz.cvut.fel/filters obsahuje filtry používané Interpreterem, které mu umožňují vyfiltrovat pouze adekvátní aplikace, které mají být následně zobrazeny na hlavní obrazovce.

src/cz.cvut.fel/globals obsahuje globální atributy aplikace (především konstanty).

src/cz.cvut.fel/listeners obsahuje všechny definované Listenery, které naše aplikace využívá.

src/cz.cvut.fel/managers obsahuje všechny definované Managery, které naše aplikace využívá.

src/cz.cvut.fel/models obsahuje modely mapované na Entity v databázi.

src/cz.cvut.fel/models/log obsahuje Entity logovacích záznamů.

src/cz.cvut.fel/models/periodic obsahuje Entity nalezených pravidelných aktivit.

src/cz.cvut.fel/periodicseekers obsahuje třídy, které využívá PeriodicActivityManager pro hledání pravidelností v logovacích záznamech.

src/mobi.intuitit.android je adresář se soubory Launcheru, který pro naše účely využíváme.

Specifikace a vysvětlení struktury projektu naší aplikace je důležité z hlediska pochopení následujících podkapitol, kde se na některé výše zmíněné adresáře odkazujeme.

2.2 Databáze

Android poskytuje pro potřeby běhu aplikací API pro správu a přístup k databázovému stroji SQLite. Nicméně toto rozhraní se nám, po shlédnutí několika tutoriálů a následném letmém ozkoušení, zdálo přinejmenším zbytečně složité. Co nás v dnešní době velmi překvapilo, je úplná absence nativního přístupu k databázi pomocí ORM. Rozhodli jsme se vyhledat některou alternativu, která by nám tento přístup umožnila. Přímo pro projekty v Androidu se nám podařilo nalézt více knihoven, mezi nimi ActiveRecord[7], OrmLite[55], ORMDroid[54] a greenDAO[35]. Všechny tyto knihovny zapouzdřují veškerou komunikaci s databází a programátorovi dávájí k dispozici velmi příjemné rozhraní s ORM přístupem k databázi. Pro potřeby naší implementace jsme nakonec zvolili knihovnu ORMDroid 2.4.1, především z důvodu její jednoduchosti a minimalističnosti ve spojení s vyhovující funkcionalitou.

2.3 Launcher

Hlavním viditelným prvkem našeho projektu je aplikace zvaná Launcher. Variant, které si mohou uživatelé operačního systému Google Android nainstalovat, existuje mnoho[69]. Z uvedeného seznamu jsou pro nás důležité sloupce „Free“ a především „Opensource“, který značí, že je jeho zdrojový kód k dispozici a je tedy možné ho dále legálně upravovat. Zároveň tím pádem nejsme nijak svázáni a můžeme náš projekt směřovat k opensource řešení a nezpoplatněné distribuci. Zmíněným kriteriím odpovídají pouze tři záznamy, které bylo potřeba otestovat prostřednictvím simulátoru, a to ADW.Launcher, Android Launcher Plus a Trebuchet Launcher.

2.3.1 ADW.Launcher

ADW Launcher vypadá na první pohled ze všech možností nejslibněji. Zasluhují se o to zajímavé recenze uživatelů i jeho zařazení do některých osvědčených modifikací. Bohužel, dostupné zdrojové kódy využívají privátní API, které není dostupné v oficiálním SDK, a z důvodu této provázanosti není možné jeho využití v rámci našeho projektu.

Po naimportování do Eclipse IDE jsme se snažili o odstranění těchto závislostí, nicméně se ukázalo, že se jedná o tak komplexní úkol, že nebylo možné ho úspěšně dokončit. Povedlo se nám sice zdrojový kód zkompilovat a spustit v simulátoru (za cenu značných modifikací), jeho výsledná funkcionalita však nebyla úplná. Zdrojové kódy k samostatné verzi (bez závislostí na privátní API) se nám nepodařilo nalézt a tudíž ani vyzkoušet.

2.3.2 Android Launcher Plus

Android Launcher Plus je jediný ze všech tří, který se nám podařilo naimportovat do Eclipse IDE bez chyb a spustit v simulátoru. Jedná se o upravenou verzi oficiálního Google Android launcheru 2.0, kterou její autor zbavil závislosti na privátní API a zachoval plnohodnotnou funkcionalitu. Jediný rozdíl oproti zbývajícím dvěma aspirantům je chybějící lišta na spodní části obrazovky, umožňující rychlý přístup ke zvoleným aplikacím. To by pro nás však neměl být problém, jelikož právě zlepšením a zrychlením spouštění aplikací se náš projekt zabývá.

Během testování tohoto launcheru jsme zaregistrovali jeho nekompatibilitu s verzemi operačního systému Google Android 4.0.x a vyšší.

2.3.3 Trebuchet Launcher

Tato aplikace byla poslední volbou, kterou jsme mohli uvažovat. Podařilo se nám vyhledat několik verzí zdrojových kódů, nicméně ani jeden se nám nepodařilo zprovoznit. Klasická verze opět využívá privátní API, která není součástí SDK. Upravená samostatná verze je sice této provázanosti zbavena, nicméně funkční je pouze pro Google Android 4.1 a vyšší, což nesplňuje naše požadavky na verzi systému.

Z výše uvedeného výčtu aplikací je zřejmé, že požadavky na bezproblémový import do Eclipse IDE i spustitelnost splnil pouze Android Launcher Plus a proto jsme se rozhodli postavit naši aplikaci právě na něm. Do budoucna však bude nutné ho upravit tak, aby byl bezproblémově spustitelný i na verzích operačního systému Google Android 4.0.x a vyšších.

2.3.4 Zobrazení a řazení aplikací

Jednou z větších modifikací kódu použitého Launcheru byla nutnost zobrazit řazený seznam aplikací. Vzhledem k praktické absenci jakékoliv dokumentace se tato operace ukázala jako netriviální, když jsme museli přepsat výchozí chování jedné z obrazovek. Na ní se obyčejně zobrazují odkazy

na aplikace, widgety a složky, které jsou zároveň persistentně uloženy v databázi a které si tam uživatel sám vložil. Pro naše potřeby jsme využili mírně modifikovanou stávající funkcionalitu pro vkládání odkazů na aplikace. Především jsme museli zajistit, aby se námi vložené položky neukládaly do databáze, čehož jsme docílili přidáním parametru saveToDatabase funkci

```
private void onDropExternal(int x, int y, Object dragInfo,
    CellLayout cellLayout,
    boolean insertAtFirst, boolean saveToDatabase)
```

v souboru mobi.intuitit.android.p.launcher/Workspace.java:

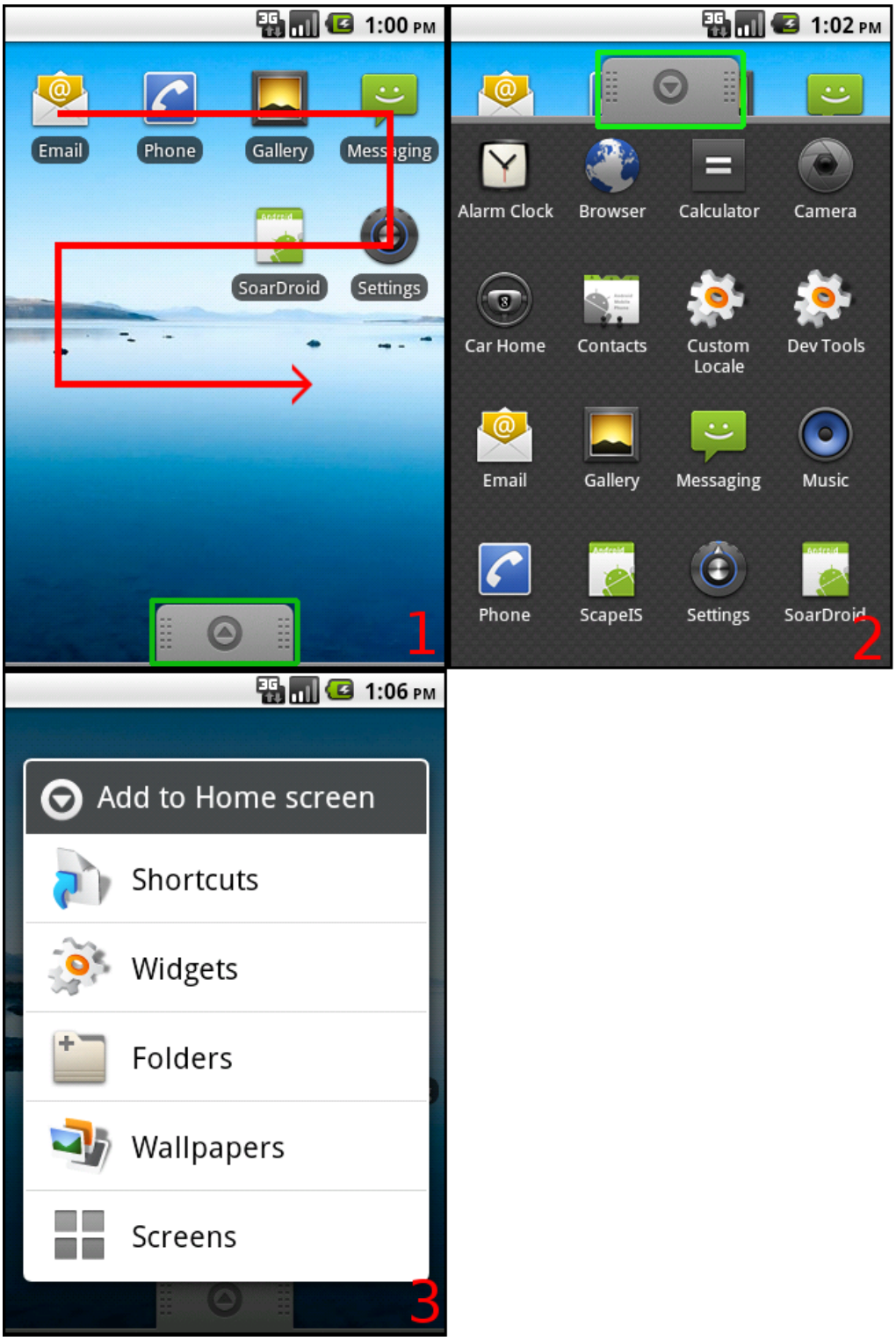
```
if(saveToDatabase){
    LauncherModel.addOrMoveItemInDatabase(mLauncher, info,
        LauncherSettings.Favorites.CONTAINER_DESKTOP,
        mCurrentScreen, lp.cellX, lp.cellY);
}
```

Díky tomu máme zaručeno, že naše pravidelné změny nebudou zbytečně operovat s přístupem k databázi. Je patrné, že v současné chvíli dochází pouze k překrytí uživatelsky vložených persistentních položek (pokud se nějaké na námi upravované obrazovce objevují) a ne vymazání z databáze. Domníváme se, že se nejedná o tak závažný problém a proto jsme se rozhodli ho neřešit vzhledem k tomu, že do budoucna budou pravděpodobně prováděny operace vedoucí k nahrazení stávajícího zobrazení prvkem, který je lépe dostupný pro zrakově postižené uživatele (vyvíjen také na katedře počítačové grafiky a interakce, ČVUT FEL).

Dále bychom ještě zmínili praktické řešení výpočtu priorit aplikací v seznamu. V kapitole 1.10.1 na straně 44 jsme si uvedli, že budeme využívat generování priorit z vlastností, které jednotlivé pravidelnosti obsahují (v současné chvíli pouze určité časové rozpětí). Využijeme k tomu obyčejnou převrácenou hodnotu z rozdílu začátku a konce (doby trvání):

```
Aplikace je pravidelně spouštěna od 8:00 do 9:00.
9:00 - 8:00 = 60 minut => priorita = 1/60
```

Z uvedeného příkladu jasně vidíme, že kratší časové období bude mít vyšší prioritu než delší, což je přesně výsledek, který požadujeme.



Obrázek 2.2: Přehled obrazovek Android Launcher Plus.

2.3.5 Uživatelské rozhraní

Uživatelské rozhraní (obrázek 2.2 na straně 57) se prakticky neliší od standardního Android Launcheru verze 2.0, ze kterého Android Launcher Plus vychází a jeho filosofie se nemění ani u novějších verzí. Díky tomu by měl být pro uživatele, který používá mobilní zařízení s Google Android, přechod na naší aplikaci v zásadě bezproblémový a pro ty ostatní bude vždy nutností naučit se něco podobného používat.

První screenshot zachycuje hlavní obrazovku s aktuálně nabízenými aplikacemi, které by uživatel mohl chtít spustit. Právě tato nabídka je finálním uživatelským výstupem celého našeho procesu sběru a interpretace dat a zároveň se jedná o jedinou viditelnou modifikaci Launcheru. Jelikož se ikony na obrazovce zobrazují v mřížce 4x4, rozhodli jsme se pro zajištění většího komfortu při pohybu seznamem řadit jednotlivé položky tak, jak je znázorněno červenou šipkou. Ne tedy klasicky zleva doprava, ale do podoby jakéhosi hada - liché řádky zleva doprava a sudé zprava doleva. Nevidomý uživatel tak při procházení seznamu bude mít usnadněný pohyb, ať už bude používat šipky na HW klávesnici, trackpad/trackball nebo například režim „explore by touch“[29].

Druhý screenshot zobrazuje hlavní menu, ve kterém jsou abecedně seřazeny všechny dostupné aplikace nainstalované v telefonu. Jeho zobrazení/skrytí se inicializuje kliknutím na zeleně orámované tlačítko a stejně tak lze skrýt tuto nabídku tlačítkem zpět.

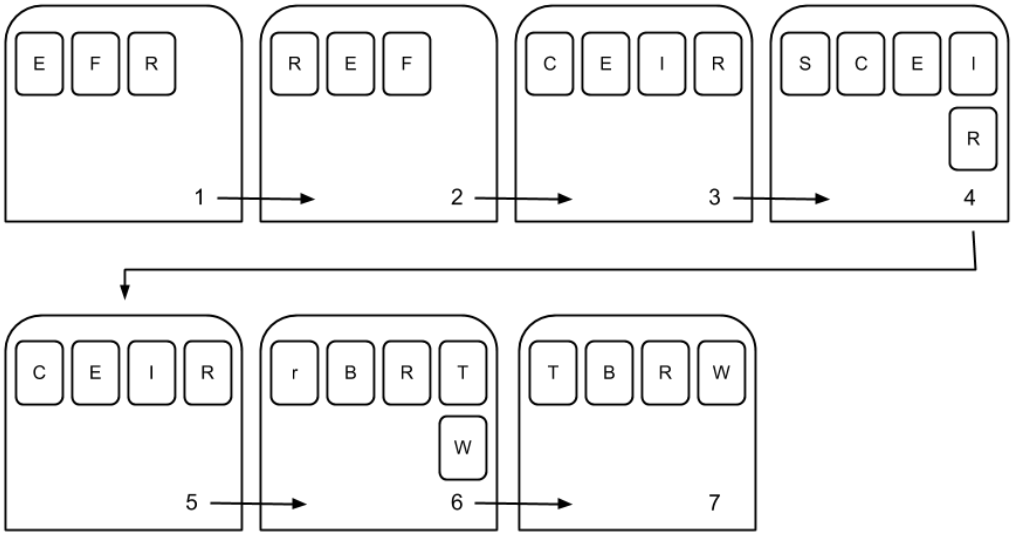
Poslední třetí obrazovka zachycuje menu, které se zobrazí na dlouhý uživatelský stisk provedený kdekoliv na volné ploše. Obsahuje nastavení původního Launcheru a umožňuje na ostatní obrazovky umisťovat ikony aplikací, widgety a podobně. Jeho skrytí je opět inicializováno stiskem tlačítka zpět.

2.3.6 Reakce UI na vytvořený scénář

Pro lepší ilustraci, jak naše aplikace funguje na uživatelském výstupu, bude nejvhodnější vzít v potaz vytvořený scénář 1.1.2 na straně 8. V závislosti na něm si namodelujeme jednotlivé obrazovky pro dílčí položky a získáme tak přehled o tom, jak systém reflektuje aktuální kontext.

K pochopení principu nám postačí vzít určitou omezenou oblast, v našem případě všední den, se specifickými vsuvkami v podobě týdenních aktivit. Jak se budou nabízené aplikace a jejich pořadí měnit, si popíšeme s pomocí diagramu 2.3 na straně 59 zobrazujícím výřez horní části hlavní obrazovky s ikonami aplikací tak, jak se postupně mění v čase.

Na něm si můžeme povšimnout přechodu z první obrazovky na druhou, kdy se do popředí dostává [R] (rádio - popis symbolů vizte legenda u dia-



Obrázek 2.3: Diagram ilustrující reakci systému na měnící se kontext v závislosti na vytvořeném scénáři. **Legenda**: [**B**] - čtečka knih (Books), [**C**] - kalendář (Calendar), [**E**] - emailový klient (Email client), [**F**] - předpověď počasí (Forecast), [**I**] - tvorba účtů (Invoices), [**R**] - rádio (Radio), [**r**] - recepty na vaření (Recipes), [**S**] - Skype, [**T**] - ČT24 (Television)

gramu 2.3 na straně 59). Děje se tak z toho důvodu, že uživatel poslouchá pravidelný rozhlasový pořad (scénář - všední den, 8:30). Přechod z druhé obrazovky na třetí ilustruje konec ranních činností a začíná nabízet sadu aplikací, kterou uživatel obvykle využívá při práci. Všimneme si vsunuté čtvrté obrazovky - do popředí se dostává [S] (Skype; scénář - pátek, 14:00) a po skončení této aktivity se nabídka vrací k totožnému pořadí jako na obrazovce číslo tři. Přechod obrazovek pět, šest a sedm reflektuje konec pracovní činnosti a změnu režimu v souvislosti s večerními aktivitami. Obrazovka šest je opět vsunutá týdenní pravidelnost (scénář - úterý a čtvrtek, 18:00), kdy se do popředí dostává aplikace [r] (recepty na vaření) nabízející možnost výběru receptu.

2.3.7 Reakce systému na uživatelské aktivity

V současné verzi systém zaznamenává pouze uživatelské aktivity, které se týkají spouštění aplikací v rámci samotného Launcheru. Systém tedy naslouchá specifické události, kterou emituje Launcher při jakémkoliv spuštění aplikace a pokud taková situace nastane, provede příslušné operace - v našem případě uložení aktuálního stavu sledovaných senzorů. Ty následně

slouží ke zpracování a vyhledávání pravidelných aktivit.

2.4 Externí knihovny

V této podkapitole si blíže popíšeme všechny externí knihovny použité v naší aplikaci. Jednotlivé projekty se zdají být udržované a především využívané širokou komunitou vývojářů. Tento fakt nám dává určitou dávku jistoty, že tomu tak bude i do budoucna. Čím více koncových aplikací danou knihovnu používá a spoléhá se na ni, tím větší je pravděpodobnost, že její tvůrci budou pokračovat ve vývoji nebo ji budou alespoň udržovat na kvalitní úrovni. Díky tomu také vývojáři dokáží vytvořit tlak (v dobrém slova smyslu) na tvůrce, kteří tím spíše neustanou v řešení problémů a zjištěných chyb.

2.4.1 OMRDroid

ORMDroid je knihovna umožňující mapování tříd programovacího jazyka na entity v databázi pomocí principu zvaného ORM. Je vyvíjena speciálně pro projekty vytvářené na platformě Android a její operace pokryjí základní práci s SQLite databází (CRUD). Poskytuje automatické vytváření tabulek podle struktury tříd v Javě. Pro práci s daty není nutné psát jakékoliv SQL příkazy, což nám poskytuje větší komfort při vývoji a zároveň se tím zmenšuje riziko programátorské chyby.

2.5 Projektový repositář

Běžnou rutinou každého programátora je zajištění ochrany kódu před poškozením nebo ztrátou. Z toho důvodu jsme se rozhodli využít služeb některého z dostupných repositářů, konkrétně GitHub[31], jejichž další výhodou je jednoduchost sdílení zdrojového kódu mezi vývojáři, snadné „mergování“ (spojování) a „verzování“ (uchovávání starších verzí pro snadnou obnovu) jednotlivých úprav.

Naším požadavkem bylo využívání Gitu[30] vzhledem k tomu, že nabízí o mnoho více možností správy zdrojového kódu než jeho konkurent Subversion[62], který je navíc v dnešní době mírně na ústupu. Vzhledem k tomu, že náš projekt není ničím tajným, nevidíme problém ve využití veřejného webového nástroje GitHub[31], který nám zdarma zajišťuje serverovou část. Podmínkou je však zobrazení repositáře ostatním uživatelům v módu read-only (jen ke čtení). Náš projekt na serveru GitHub[53].

Kapitola 3

Budoucí práce

Úkolem této kapitoly by mělo být vytvoření vize budoucích prací a dalšího rozvíjení potencionálu našeho projektu. Jak se mohl čtenář během studování této práce přesvědčit, jedná se o velmi komplexní a rozsáhlý problém, který nabízí nepřeberné množství oblastí, jenž se dají rozšiřovat a vylepšovat. Obrovské možnosti budoucího systému jsme si nastínili především v kapitole 1.3 na straně 24, která popisuje jednotlivé součásti způsobem, jak by mohly fungovat v plně dokončeném projektu.

Z tohoto faktu je patrné, že v rámci našeho projektu jsme byli schopni implementačně pokrýt pouze omezenou podmnožinu problému. Díky detailní analýze a důrazu na nízkou provázanost jednotlivých součástí je však možné nezávisle rozšiřovat jednotlivé oblasti, aniž bychom museli zásadně modifikovat stávající zdrojový kód. Všechny operace jsou o to snazší, že je princip fungování jednotlivých částí rozepsán v textu.

Jednou z nutných oblastí, které musí být provedeny pro plnohodnotné nasazení, je testování na reálných datech. Rozhodně je však potřeba před jeho samotným masovým započetím doimplementovat funkcionalitu spojenou s čištěním databáze od starých záznamů (jak logovacích, tak periodických), která zaručí aktuálnost nasbíraných a interpretačních dat.

Naším velkým úkolem do budoucna by mohlo být navržení a vytvoření veřejně dostupné API tak, jak je popsána v kapitole 1.11 na straně 46, která by ostatním běžícím aplikacím umožňovala efektivní spolupráci s naším Launcherem. Uživateli by toto spojení poskytovalo o několik úrovní vyšší komfort a zjednodušení spouštění aplikací. Ruku v ruce jde s tímto záměrem úsilí o specifikaci pokynů (anglicky guidelines), které zdokumentují a popíší nejlepší způsoby (best practices) vývoje mobilních aplikací tak, aby jejich spolupráce s naším Launcherem byla maximálně efektivní. Aplikace by mohly vystavovat některá interní data, která by následně mohla

být využita námi, stejně jako spouštět eventy, které bychom byli schopni zachytit a lépe tak například monitorovat jejich životní cyklus. Vzhledem k tomu, že naše aplikace cílí na zrakově postižené uživatele, mohl by navíc součástí těchto specifikací být návod, jak správně vyvíjet mobilní aplikace, aby jejich přístupnost těmto skupinám byla co nejlepší.

V rámci úprav uživatelského rozhraní a komunikace s uživatelem bude potřeba implementovat aplikační logiku popsanou v kapitole 1.10.2 na straně 45. Za pozornost navíc stojí využití (jako hlavní obrazovky) projektu vyvíjeného na ČVUT FEL, katedře počítačové grafiky a interakce. Jedná se o modifikované Grid View[36], které poskytuje vylepšenou funkcionalitu a zlepšení přístupnosti zrakově postiženým.

Závěr

V úvodní kapitole jsme si nastínili celkový problém, který jsme se snažili v rámci této práce popsat, stanovit si hlavní cíle a vytvořit funkční řešení. V závěru bychom se měli zpětně na tyto body podívat a kriticky zhodnotit, zda jsme si vedli úspěšně - co se nám povedlo naplnit a co ne.

Základem vyhovujícího řešení byla nutnost dobře poznat cílovou skupinu uživatelů, její potřeby a pravidelný cyklus. Myslíme si, že tento krok se nám podařilo splnit velmi dobře a provedené rozhovory byly pro nás nedocenitelnou profesní zkušeností. Díky jejich dostatečnému množství pak bylo možné stanovit realistický pravidelný režim průměrného uživatele naší aplikace, což nám poskytlo ještě lepší vhled do nitra celkového problému. Na základě toho jsme byli schopni zavést pro takový režim sady abstraktnějších pojmů, které nám pomohly lépe pokrýt některé vazby a stavy.

Detailně provedená rešerše operačních systémů nám pomohla vybrat ten, který optimálně splňuje naše požadavky. V rámci jednotlivých kriterií jsme se navíc snažili reflektovat specifické potřeby naší cílové skupiny a produkt tak lépe směřovat právě na ně. Zároveň jsme naši pozornost upřeli na existující řešení, která se zabývají podobnými typy problémů nebo by mohly využívat některé oblasti naší aplikace.

Dále jsme provedli samotný návrh klíčových součástí našeho systému. Po celou dobu vývoje naší aplikace jsme měli na mysli jejich výslednou flexibilitu a jednoduchou rozšiřitelnost spojenou s nízkou provázaností. Veškerou jejich specifikaci protíná linie podobného rozvržení na jednotlivé moduly, jež disponují přesně definovanou a striktně omezenou funkcionalitou (neobsahují zbytečně nic navíc), která právě tyto požadavky zaručuje. Navíc jsou mezi sebou propojeny pomocí flexibilních vazeb (například Manager - Listener zmiňovaný v kapitole 1.6 na straně 40), což také vyhovuje stanoveným podmínkám. Řešením, kde naopak vidíme místo pro zlepšení, je návrh mod-

ulu pro vyhledávání pravidelností, kde si nejsme zcela jisti, že tento koncept dokáže pokrýt budoucí nároky na rozšiřování aplikace.

Od začátku byl v práci kladen důraz především na dobře provedenou analýzu a návrh. Tomuto požadavku jsme následně museli přizpůsobit časový plán a tím pádem přidělit méně času vlastní implementaci, která díky tomu pokrývá určitou podmnožinu komplexně definovaného problému. Nicméně naprogramovaná aplikace je funkční a v současné chvíli umožňuje zaznamenávání jednotlivých aktivit na základě časových pravidelností a dostupných Wi-Fi sítí. Jedná se tedy o určitou beta verzi připravenou ke komplexnějšímu testování, jenž je z podstaty problému samo námětem na samostatnou práci, především z důvodu časové náročnosti v případě testování na reálných datech.

Literatura

[1] *Accessibility*. [cit. 22.4.2013]
http://developer.android.com/guide/topics/ui/accessibility/index.html.

[2] *Accessibility on iPhone*. [cit. 22.4.2013]
http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/iPh

[3] *adw-launcher-android*. [cit. 22.4.2013]
http://code.google.com/p/adw-launcher-android/.

[4] *Amplified Analytics - Customer Satisfaction with Windows smart phones rise by 18%*. [cit. 22.4.2013]
http://blog.amplifiedanalytics.com/2012/07/customer-satisfaction-with-windows-smart-phones-rise-by-18/.

[5] *Anatomy of iPhone*. [cit. 22.4.2013]
http://ipod.about.com/od/introductiontotheiphone/ss/Anatomy-Of-Iphone-4.htm.

[6] *Android 1.5: Reading SMS messages*. [cit. 22.4.2013]
http://stackoverflow.com/questions/900294/android-1-5-reading-sms-messages.

[7] *android-active-record*. [cit. 22.4.2013]
http://code.google.com/p/android-active-record/.

[8] *Android Calendar Events*. [cit. 22.4.2013]
http://stackoverflow.com/questions/4302209/android-calendar-events.

[9] *Android: CallLog.Calls*. [cit. 22.4.2013]
http://developer.android.com/reference/android/provider/CallLog.Calls.html.

[10] *Android: ContentObserver*. [cit. 22.4.2013]
http://developer.android.com/reference/android/database/ContentObserver.html.

[11] *Android Developers*. [cit. 22.4.2013]
http://developer.android.com/.

[12] *Android GPS on or off state*. [cit. 22.4.2013]
http://stackoverflow.com/questions/4300572/android-gps-on-or-off-state.

[13] *android: how do i open another app from my app?* [cit. 22.4.2013]
http://stackoverflow.com/questions/2923265/android-how-do-i-open-another-app-from-my-app.

[14] *Android: How to get calendar change events? Is calendar observer working?* [cit. 22.4.2013]
http://stackoverflow.com/questions/9440993/how-to-get-calendar-change-events-is-calendar-observer-working.

[15] *Android i18n*. [cit. 25.4.2013]
http://developer.android.com/guide/topics/resources/localization.html.

[16] *Android Launcher - Git repository*. [cit. 22.4.2013]
https://android.googlesource.com/platform/packages/apps/Launcher2.git.

[17] *Android WifiManager*. [cit. 22.4.2013]
http://developer.android.com/reference/android/net/wifi/WifiManager.html#isWifiEnabled

[18] *Apple oficiálně upozorňuje na negativa Jailbreaku*. [cit. 22.4.2013]
http://www.muj-ipad.cz/apple-oficialne-upozornuje-na-negativa-jailbreaku.

[19] *Apple URL Scheme Reference*. [cit. 22.4.2013]
http://developer.apple.com/library/ios/#featuredarticles/iPhoneURLScheme_Reference/In

[20] *Atooma*. [cit. 31.5.2013]
http://www.atooma.com/welcome/.

[21] *The Best App Launcher for iPhone*. [cit. 22.4.2013]
http://lifehacker.com/5836229/the-best-app-launcher-for-iphone.

[22] *Calendar Provider*. [cit. 22.4.2013]
http://developer.android.com/guide/topics/providers/calendar-provider.html.

[23] *Call history, SMS history, Email history in iOS.* [cit. 22.4.2013] http://stackoverflow.com/questions/9822031/call-history-sms-history-email-history-in-ios.

[24] *Class (Java 2 Platform SE v1.4.2).* [cit. 22.4.2013] http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Class.html.

[25] *Co je to vlastně ten Launcher?* [cit. 22.4.2013] http://www.androidmarket.cz/navod/co-je-to-vlastne-ten-launcher/.

[26] *Co se zabydlelo v mém Androidu.* [cit. 22.4.2013] http://www.geekblind.com/2013/01/co-se-zabydlelo-v-mem-androidu.html.

[27] *Cross-platform.* [cit. 22.4.2013] http://en.wikipedia.org/wiki/Cross-platform.

[28] *Ease of access settings improve Windows Phone 8 accessibility.* [cit. 22.4.2013] http://allaboutwindowsphone.com/flow/item/15887_Ease_of_access_settings_improv.p

[29] *Explore by touch.* [cit. 22.4.2013] http://support.google.com/android/bin/answer.py?hl=cs&answer=2492750.

[30] *Git version control system.* [cit. 29.4.2013] http://git-scm.com/.

[31] *GitHub.* [cit. 29.4.2013] https://github.com/.

[32] *Google Android: Location Strategies.* [cit. 22.4.2013] http://developer.android.com/guide/topics/location/strategies.html.

[33] *Google Android: LocationListener.* [cit. 22.4.2013] http://developer.android.com/reference/android/location/LocationListener.html.

[34] *Google Android: LocationManager.* [cit. 22.4.2013] http://developer.android.com/reference/android/location/LocationManager.html.

[35] *greenDAO – Android ORM for SQLite.* [cit. 22.4.2013] http://greendao-orm.com/.

[36] *Grid View.* [cit. 6.5.2013] http://developer.android.com/guide/topics/ui/layout/gridview.html.

[37] *Heureka.* [cit. 22.4.2013]
http://www.heureka.cz/.

[38] *How to access calendar data for Windows Phone.* [cit. 22.4.2013]
http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh286421(v=vs.105).aspx

[39] *How to check if phone is mute, vibrate or loud in android.* [cit. 22.4.2013]
http://stackoverflow.com/questions/11985448/how-to-check-if-phone-is-mute-vibrate-or-loud-in-android.

[40] *How To Create Custom Home Screen Shortcuts For Settings Toggles Without Jailbreaking.* [cit. 22.4.2013]
http://www.iphonehacks.com/2011/11/how-to-create-custom-home-screen-shortcuts-for-settings-toggles-without-jailbreaking.html.

[41] *Interfaces (The Java Tutorials, Learning the Java Language, Interfaces and Inheritance).* [cit. 22.4.2013]
http://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html.

[42] *Introduction to Calendars and Reminders.* [cit. 22.4.2013]
http://developer.apple.com/library/ios/#documentation/DataManagement/Conceptual/Ev

[43] *iOS jailbreaking.* [cit. 22.4.2013]
http://en.wikipedia.org/wiki/IOS_jailbreaking.

[44] *Jablečné doteky: Apple iPhone 4S a uživatelské dojmy poslepu.* [cit. 22.4.2013]
http://fanapple.cz/jablecne-doteky-apple-iphone-4s-a-uzivatelske-dojmy-poslepu/.

[45] *Jak začít se čtyřkovým Androidem?* [cit. 22.4.2013]
http://www.blind-android.cz/clanky/ice-cream-sandwich-pod-lupou/1.-dil–jak-zacit-se-ctyrkovym-androidem-.html.

[46] *Launch an app from within another (iPhone).* [cit. 22.4.2013]
http://stackoverflow.com/questions/419119/launch-an-app-from-within-another-iphone.

[47] *Launch another app from wp7.* [cit. 22.4.2013]
http://stackoverflow.com/questions/13496119/launch-another-app-from-wp7.

[48] *Launching Your iPhone App Via Custom URL Scheme.* [cit. 22.4.2013] http://iosdevelopertips.com/cocoa/launching-your-own-application-via-a-custom-url-scheme.html.

[49] *Locale.* [cit. 31.5.2013] http://www.twofortyfouram.com/product.

[50] *Mobile OS market share.* [cit. 22.4.2013] http://stats.areppim.com/stats/stats_mobiosxtime_eu.htm.

[51] *Mobile statistics.* [cit. 22.4.2013] http://www.mobilestatistics.com/mobile-statistics/.

[52] *Nokia talks accessibility, confirms there is no screen reader for Windows Phone 8.* [cit. 22.4.2013] http://wmpoweruser.com/nokia-talks-accessibility-confirms-there-is-no-screen-reader-for-windows-phone-8/.

[53] *Náš projekt SmartLauncher na serveru GitHub.* [cit. 22.4.2013] https://github.com/machbohu/smartlauncher.

[54] *ORMDroid.* [cit. 22.4.2013] https://github.com/roscopeco/ormdroid.

[55] *OrmLite - Lightweight Java ORM Supports Android and SQLite.* [cit. 22.4.2013] http://ormlite.com/sqlite_java_android_orm.shtml.

[56] *PCMag - Readers choice awards.* [cit. 22.4.2013] http://www.pcmag.com/article2/0,2817,2402202,00.asp.

[57] *Přístupnost Androida 2.3.x pro zrakově postižené.* [cit. 22.4.2013] http://www.androidmarket.cz/ruzne/pristupnost-androida-2-3-x-pro-zrakove-postizene/.

[58] *Přístupnost aplikací na Google Android.* [cit. 22.4.2013] http://www.blind-android.cz/clanky/pristupnost-aplikaci.html.

[59] *Přístupnost iPadu.* [cit. 22.4.2013] http://poslepu.blogspot.cz/2011/05/ipad-standard-pristupnosti-dotykovych.html.

[60] *Start Screen for Windows Phone 8.* [cit. 22.4.2013] http://www.windowsphone.com/en-us/store/app/start-screen-for-windows-phone-8/621dc25e-9be7-497c-8ad9-7a621ea7ebf9.

[61] *Struktura Google Android projektu.* [cit. 23.4.2013] http://developer.android.com/tools/projects/index.html.

[62] *Subversion version control system.* [cit. 22.4.2013] http://subversion.apache.org/.

[63] *Tasker.* [cit. 31.5.2013] http://tasker.dinglisch.net/.

[64] *V útrobách jednoho iPhonu.* [cit. 22.4.2013] http://www.geekblind.cz/?p=59.

[65] *What is the best way to handle updates when the calendar syncs new events?* [cit. 22.4.2013] http://stackoverflow.com/questions/9170903/what-is-the-best-way-to-handle-updates-when-the-calendar-syncs-new-events.

[66] *Wikipedia - iOS included applications.* [cit. 22.4.2013] http://en.wikipedia.org/wiki/IOS#Included_applications.

[67] *Wikipedia - iPhone.* [cit. 22.4.2013] http://cs.wikipedia.org/wiki/IPhone.

[68] *Wikipedia - Počítačová platforma.* [cit. 22.4.2013] http://cs.wikipedia.org/wiki/Platforma_(informatika).

[69] *Wikipedia: List of Android launchers.* [cit. 22.4.2013] http://en.wikipedia.org/wiki/List_of_Android_launchers.

[70] *Windows Phone - Tiler.* [cit. 22.4.2013] http://www.windowsphone.com/en-us/store/app/tiler/a28a37f8-48a5-429c-b8e7-fd04e1ccf5da.

[71] *Windows Phone 8 Accessibility.* [cit. 22.4.2013] http://www.freelists.org/post/real-eyes/ATI-Windows-Phone-8-Accessibility.

[72] *Windows Phone 8 support to capture SMS or a incoming call?* [cit. 22.4.2013] http://stackoverflow.com/questions/13176545/windows-phone-8-support-to-capture-sms-or-a-incoming-call.

[73] *Windows Phone: GeoCoordinateWatcher.Status Property.* [cit. 22.4.2013] http://msdn.microsoft.com/en-us/library/windowsphone/develop/system.device.location.geocoordinatewatcher.status(v=vs.105).aspx.

[74] *Windows Phone SDK.* [cit. 22.4.2013]
https://dev.windowsphone.com/en-us/downloadsdk.

[75] *Windows Phone SDK 8.0.* [cit. 22.4.2013]
http://www.microsoft.com/en-us/download/details.aspx?id=35471.

[76] *WindowsPhone: DeviceNetworkInformation.IsWiFiEnabled Property.* [cit. 22.4.2013]
http://msdn.microsoft.com/en-us/library/windowsphone/develop/microsoft.phone.net.networkinformation.devicenetworkinformation.iswifienabled(v=vs.105).aspx.

[77] *Úvaha: nevidomí a dotykové telefony, bezradní či ne?* [cit. 22.4.2013]
http://geekblind.blogspot.cz/2012/10/uvaha-nevidomi-dotykove-telefony.html.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

BSSID Basic Service Set IDentifier

BTS Base Transceiver Station

CRUD Create Read Update Delete

GPS Global Positioning System

HW HardWare

ID IDentification number

IDE Integrated Development Environment

IDOS Integrovaný DOpravní Systém

MHD Městská Hromadná Doprava

ORM Object Related Mapping

OS Operating System

OWL Web Ontology Language

SDK Software Development Kit

SSID Service Set Identifier

UI User Interface

URL Uniform Resource Locator

Wi-Fi Wireless Fidelity

XML eXtensible Markup Language

Příloha B

Seznam ostatních příloh

B.1 Tyflonet - dotační politika

Dobrý den pane Machu,

reaguji na Váš dotaz, který jste zaslal na náš informační portál Tyflonet.

Příspěvek na zvláštní pomůcku (nárok, podmínky, výši apod.) vymezuje zákon č. 329/2011 Sb., o poskytování dávek osobám se zdravotním postižením a prováděcí vyhláška č. 388/2011 Sb., o provádění některých ustanoveních zákona o poskytování dávek osobám se zdravotním postižením.

NÁROK NA PŘÍSPĚVEK NA ZVLÁŠTNÍ POMŮCKU má osoba, která má (podrobnější výčet zdravotního postižení odůvodňující přiznání příspěvku na zvláštní pomůcku a zdravotní stavy vylučující přiznání příspěvku je popsáno v příloze k zákonu č. 329/2011 Sb.):

- těžkou vadu nosného nebo pohybového ústrojí,
- těžké sluchové postižení,
- těžké zrakové postižení.

Je-li pomůckou motorové vozidlo, má nárok na příspěvek na zvláštní pomůcku osoba, která má těžkou vadu nosného nebo pohybového ústrojí anebo těžkou nebo hlubokou mentální retardaci.

Okruh zdravotních postižení odůvodňujících přiznání příspěvku a zdravotní stavy vylučující jeho přiznání jsou uvedeny v příloze k zákonu č. 329/2011 Sb., o poskytování dávek osobám se zdravotním postižením.

PODMÍNKOU PRO POSKYTNUTÍ PŘÍSPĚVKU NA ZVLÁŠTNÍ POMŮCKU dále je, že:

1. Osoba je starší a) 3 let (motorové vozidlo, úprava bytu), b) 15 let (vodicí pes), nebo c) 1 roku (všechny ostatní pomůcky).

2. Zvláštní pomůcka umožní osobě sebeobsluhu nebo ji potřebuje k realizaci pracovního uplatnění, k přípravě na budoucí povolání, k získávání informací, vzdělávání anebo ke styku s okolím.

3. Osoba může zvláštní pomůcku využívat nebo může zvláštní pomůcku využívat ve svém sociálním prostředí.

4. Zvláštní pomůcka není zdravotnickým prostředkem, který je plně či částečně hrazen z veřejného zdravotního pojištění anebo je osobě zapůjčen příslušnou zdravotní pojišťovnou. Také nesmí jít o zdravotnický prostředek, který nebyl osobě uhrazen z veřejného zdravotního pojištění nebo zapůjčen zdravotní pojišťovnou z důvodu nedostatečné zdravotní indikace.

5. Je-li pomůckou motorové vozidlo, je také podmínkou, že se osoba opakovaně v kalendářním měsíci dopravuje a že je schopna řídit motorové vozidlo nebo je schopna být vozidlem převážena.

Příspěvek na zvláštní pomůcku se poskytuje na zvláštní pomůcku v základním provedení, které osobě vzhledem k jejímu zdravotnímu postižení plně vyhovuje a splňuje podmínku nejmenší ekonomické náročnosti.

Seznam druhů a typů zvláštních pomůcek, na které je příspěvek určen, je obsažen ve vyhlášce (č. 388/2011 Sb.). Příspěvek se poskytuje i na pomůcku, která ve vyhlášce uvedena není, a to za podmínky, že jí krajská pobočka ÚP považuje za srovnatelnou s některou z pomůcek, která ve vyhlášce uvedena je.

STANOVENÍ VÝŠE PŘÍSPĚVKU NA ZVLÁŠTNÍ POMŮCKU Zákon č. 329/2011 Sb. rozlišuje, zda je o pomůcku v ceně do 24 000 Kč nebo přes 24 000 Kč a speciální úpravu má pro motorové vozidlo.

Na pořízení zvláštní pomůcky v ceně nižší než 24 000 Kč se příspěvek na zvláštní pomůcku poskytne jen osobě, která má příjem (příjem s ní společně posuzovaných osob) nižší než 8násobek životního minima jednotlivce nebo životního minima společně posuzovaných osob. Výše příspěvku na zvláštní pomůcku se stanoví tak, že spoluúčast osoby činí 10 již zaplacené ceny zvláštní pomůcky, nejméně však 1 000 Kč. Z důvodů hodných zvláštního zřetele, zejména žádá-li osoba opakovaně o příspěvek na různé zvláštní pomůcky v ceně do 24 000 Kč, lze tento příspěvek poskytnout, i když příjem osoby a příjem osob s ní společně posuzovaných přesahuje výše uvedený násobek životního minima.

Výše příspěvku na pořízení zvláštní pomůcky, jejíž cena je vyšší než 24 000 Kč, se stanoví tak, že spoluúčast osoby činí 10 zaplacené ceny zvláštní pomůcky. Jestliže osoba nemá dostatek finančních prostředků ke spoluúčasti, krajská pobočka ÚP určí nižší míru spoluúčasti (s přihlédnutím k míře využívání zvláštní pomůcky, k příjmu osoby a příjmu osob s ní společně posuzovaných a k celkovým sociálním a majetkovým poměrům), minimálně však 1 000 Kč.

Výše příspěvku na zvláštní pomůcku (motorové vozidlo) se stanoví s přihlédnutím k četnosti a důvodu dopravy, příjmu osoby a příjmu osob s ní společně posuzovaných a celkovým sociálním a majetkovým poměrům. Maximální výše příspěvku na zvláštní pomůcku poskytovaného na pořízení motorového vozidla činí 200 000 Kč.

Limity výše příspěvku:

Maximální výše příspěvku na zvláštní pomůcku, jejíž cena je vyšší než 24 000 Kč, činí 350 000 Kč. 400 000 Kč v případě příspěvku na zvláštní pomůcku na pořízení schodišťové plošiny. Součet vyplacených příspěvků na zvláštní pomůcku nesmí v 60 kalendářních měsících po sobě jdoucích přesáhnout částku 800 000 Kč; 850 000 Kč, pokud byl v této době poskytnut příspěvek na zvláštní pomůcku na pořízení schodišťové plošiny.

POVINNOST VRÁTIT PŘÍSPĚVEK NA ZVLÁŠTNÍ POMŮCKU Osoba, které byl vyplacen příspěvek na zvláštní pomůcku, je povinna tento příspěvek nebo jeho poměrnou část vrátit, jestliže:

- nepoužila příspěvek do 3 měsíců ode dne jeho vyplacení nebo ve lhůtě stanovené krajskou pobočkou ÚP na pořízení zvláštní pomůcky,
- nepoužila vyplacený příspěvek v plné výši do 3 měsíců ode dne jeho vyplacení nebo ve lhůtě stanovené krajskou pobočkou ÚP,
- v období před uplynutím 60 kalendářních měsíců po sobě jdoucích ode dne vyplacení příspěvku nebo v období před uplynutím 120 kalendářních měsíců po sobě jdoucích ode dne vyplacení příspěvku poskytnutého na pořízení motorového vozidla pozbyla vlastnické právo ke zvláštní pomůcce,
- v období před uplynutím 60 kalendářních měsíců po sobě jdoucích ode dne vyplacení příspěvku nebo v období před uplynutím 120 kalendářních měsíců po sobě jdoucích ode dne vyplacení příspěvku poskytnutého na pořízení motorového vozidla přestala zvláštní pomůcku užívat,
- se přestala opakovaně dopravovat nebo přestala být schopna převozu motorovým vozidlem, byl-li vyplacen příspěvek na pořízení motorového vozidla,

- použila příspěvek v rozporu s rozhodnutím o jeho přiznání nebo
- se prokáže, že osoba, uvedla v žádosti o příspěvek na zvláštní pomůcku nepravdivé nebo zkreslené údaje.

Osoba není povinna vyplacený příspěvek na zvláštní pomůcku nebo jeho poměrnou část vrátit, jestliže:

- v období před uplynutím 60 kalendářních měsíců po sobě jdoucích ode dne jeho vyplacení přestala užívat zvláštní pomůcku z důvodu změny zdravotního stavu nebo v období před uplynutím 120 kalendářních měsíců po sobě jdoucích ode dne vyplacení příspěvku poskytnutého na pořízení motorového vozidla se z důvodu změny zdravotního stavu přestala opakovaně dopravovat nebo pozbyla schopnost být převážena motorovým vozidlem,
- byl vyplacen příspěvek na pořízení vodícího psa a pes v období před uplynutím 60 kalendářních měsíců po sobě jdoucích ode dne vyplacení příspěvku zemře nebo ztratí své dovednosti z důvodu onemocnění nebo úrazu, k němuž došlo bez zavinění příjemce dávky, nebo
- osoba zemřela.

Povinnost vrátit poměrnou část příspěvku nevzniká, jestliže tato částka nepřesahuje 100 Kč. Krajská pobočka Úřadu práce může z důvodů hodných zvláštního zřetele rozhodnout o prominutí povinnosti vrátit příspěvek na zvláštní pomůcku nebo jeho poměrnou část.

Možnosti financování zvláštních pomůcek:

- vlastní finanční zdroje,
- státní podpora - podáním žádosti o příspěvek přes Úřad práce,

Pokud žadateli nebyl finanční příspěvek na pořízení kompenzační pomůcky přiznán v dostatečné výši, aby si mohl kompenzační pomůcku koupit a jsou vyčerpány veškeré možnosti dovolání se nároku na příspěvek na pomůcku (využití odvolání u MPSV), existují v ČR nadace a nadační fondy, které pomáhají v sociální a zdravotní oblasti.

Přeji hezký den.
Eva Vonešová

Příloha C

Obsah přiloženého CD

- `README.md` ..... stručný popis projektu a obsahu CD
- `docs`
  - `master_thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `pdf` ..... text práce ve formátu PDF
- `interview` ..... zvukové záznamy uživatelských rozhovorů
- `source` ..... zdrojové kódy implementace