Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Využití pohybových senzorů na platformě Android

Plzeň 2015 Martin Hospaska

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 5. května 2015

Martin Hospaska

Abstract

Library and sample application for motion-aware applications

This bachelor thesis deals with a creation and usage of a simple Android library targeted at the area of motion-aware applications. It does so using selected sensors and APIs which can provide required information.

The author of the thesis provides a step-by-step instruction how to programme and build a simple library. To those interested in the direct usage of this library, it further offers a manual for their own applications.

The major output of this thesis is a library which saves the user direct usage of sensors and Google APIs. That can spare the user from having to spend a lot of time and lines of code. The minor output is a simple application which verifies the functionality of the library and demonstrates it in practice.

Abstrakt

Knihovna a ukázková aplikace pro motion-aware aplikace

Tato práce popisuje způsob vytvoření a použití jednoduché knihovny pro OS Android, která cílí na oblast motion-aware aplikací. K tomu využívá vybrané senzory a API, které v této oblasti poskytují požadované informace.

Čtenář se v tomto textu dozví, jak takovouto jednoduchou knihovnu naprogramovat a sestavit. Ten, kdo chce tuto knihovnu začít rovnou využívat, najde zde opět návod, jak ji jednoduše využít ve své vlastní aplikaci.

Výstupem této práce je knihovna, která odstíní uživatele od přímé práce se senzory a API, čímž ušetří čas a spoustu řádek kódu. Druhým výstupem je aplikace, která ověřuje funkcionalitu knihovny a demonstruje vše v praxi.

Obsah

1 Úvod

Účelem této práce je prozkoumání vhodných senzorů a rozhraní pro programování aplikací a jejich následné využití pro *motion-aware* aplikace v systému Android. Po získání dostatku informací o této oblasti aplikací bude následně zhotovena jednoduchá knihovna využívající získaných informací. Ověření funkcionality vytvořené knihovny proběhne v ukázkové aplikaci, která tak uživateli zároveň ukáže, jak tuto knihovnu využívat.

V úvodu textu je čtenář krátce seznámen s historií a verzováním operačního systému Android. Poté následuje výčet několika dostupných aplikací, včetně jejich krátkého popisu, které posloužily jako inspirace pro navrhovanou knihovnu, či které mohou být pro čtenáře jednoduše zajímavé.

Následně jsou čtenáři popsány principy fungování a využívání dostupných senzorů v zařízení. Dále následuje návrt vlastní knihovny, po kterém přichází předvedení využití *Google Play services* a jejích *API*, které lze pro požadovanou oblast aplikace využít. Toto vše je také v textu prakticky předvedeno uvedením fragmentů kódů, díky kterým čtenář snadno pochopí, jak knihovna, respektive ukázková aplikace, pracují. Díky tomu si také může svou vlastní knihovnu jednoduše naprogramovat.

V další části se nachází popis aplikace, na které byly získané informace a vytvořená knihovna otestovány. Čtenář zde tedy najde vysvětlení, jak knihovnu jednoduše využít přímo v aplikaci, prostým voláním dostupných *metod*.

Závěrem se v textu nachází kapitola, ve které jsou rozebrány a zkoušeny různé testovací scénáře pro výslednou aplikaci. Poslední část této kapitoly rozebírá možnosti pro následné rozšiřování vzniklé knihovny.

2 Android

Tato kapitola krátce popisuje dnešní možnosti využití mobilních zařízení a zároveň stručně shrnuje historii a rozšířenost systému Android. Dále také popisuje verze, kterými tento systém při jeho vývoji prošel, a jejich dopad na možnosti programování aplikací.

2.1 Využití zařízení

V dnešní době je mobilní telefon většinou uživatelů již považován za přístroj, který slouží k mnohem více účelům, než pouhému telefonování a psaní SMS. Díky stále více se rozšiřujícímu kvalitnímu internetovému připojení, či prostému připojení přes Wi-Fi, lze dnes telefon používat i k mnoha užitečným činnostem či k zábavě. A právě k těmto nově vzniklým mobilním aktivitám potřebuje mobilní zařízení mnoho různých senzorů.

Využít je lze například k navigaci pomocí GPS senzoru, jako základ pro aplikace určené pro sport či starající se o zdraví uživatele, aplikace proti krádeži zařízení a v neposlední řadě lze telefon využít například i jako vodováhu.

Prvním bodem zadání této práce je prozkoumání senzorů a jejich *API*, jež jsou dostupné na zařízeních s operačním systémem Android. Dobře pro tento bod posloužily webové stránky[1] společnosti Google, Inc., na nichž lze nalézt mnoho rad a tipů pro vývojáře, kteří pracují (či teprve začínají pracovat) s operačním systémem Android.

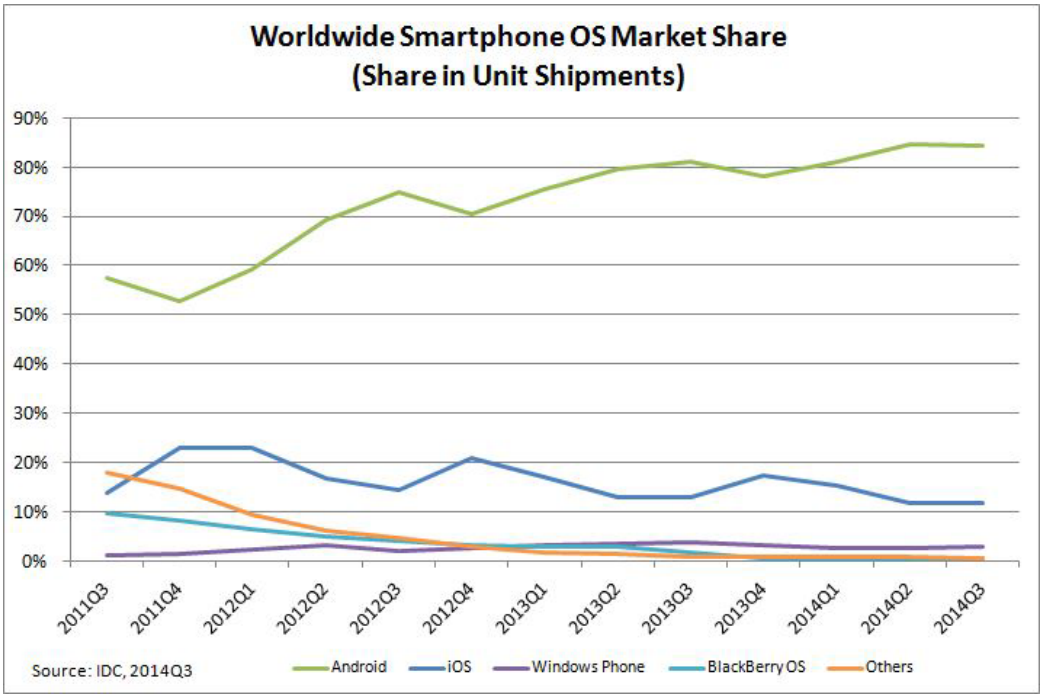
2.2 Historie

Operační systém Android byl původně vyvíjen společností Android Inc., jež byla založena v roce 2003 v Kalifornii. O dva roky později byla tato firma odkoupena společností Google, Inc., která začala tento systém dále rozvíjet a přizpůsobovat pro mobilní telefony. V roce 2007 bylo založeno konsorcium Open Handset Alliance, jenž dodnes sdružuje mnoho významných výrobců

[1] Rozcestí návodů lze najít na adrese: `http://developer.android.com/guide/index.html`.

mobilních telefonů. Mezi ně patří například Google, Sony, HTC, Intel, LG, Motorola, NVIDIA, Qualcomm, Samsung, Texas Instruments a další. Cílem tohoto sdružení bylo vyvinout otevřený standard pro operační systém mobilních zařízení. Tím se stal operační systém Android, jenž staví své základy na Linuxovém jádře verze 2.6.

Prvním produktem tohoto snažení se stal mobilní telefon HTC Dream, který byl jako první mobilní telefon vybaven operačním systémem Android. Do prodeje byl uveden v roce 2008 v Americe (v ČR se oficiálně objevil až v lednu roku 2009). Od této chvíle se Android stále více a více rozšiřuje a dnes jeho podíl na trhu mezi mobilními operačními systémy dosahuje celých 84,4%[2]. Jeho obrovský náskok před několika vybranými (nejrozšířenějšími) mobilními operačními systémy si lze prohlédnout na obrázku 2.1 níže.



Obrázek 2.1: Podíly operačních systémů na trhu mezi roky 2011 až 2014 [4].

[2]Tento údaj byl získán z analytického webu `http://www.idc.com/prodserv/smartphone-os-market-share.jsp` a je platný k 3. čtvtletí roku 2014 [4].

2.3 Verze Androidu

Operační systém Android prošel od svého vzniku již mnoha verzemi. Jejich neúplný (neobsahuje některé ze sub-verzí) výčet si lze prohlédnout v tabulce 2.1 níže:

Jméno a verze OS	**API**	**Jméno a verze OS**	**API**
Alpha (1.0)	1	Gingerbread (2.3 - 2.3.7)	9 - 10
Beta (1.1)	2	Honeycomb (3.0 - 3.2.6)	11 - 13
Cupcake (1.5)	3	Ice Cream Sandwich (4.0 - 4.0.4)	14 - 15
Donut (1.6)	4	Jelly Bean (4.1 - 4.3.1)	16 - 18
Eclair (2.0 - 2.1)	5 - 7	KitKat (4.4 - 4.4.4)	19 - 20
Froyo (2.2 - 2.2.3)	8	Lollipop (5.0 - 5.0.2)	21

Tabulka 2.1: Verze OS Android a jejich čísla API [1]

Každá z těchto verzí taktéž dostala své unikátní číslo pro *API* (neboli rozhraní pro programování aplikací), jenž je navyšováno s každou novou verzí (či její významnou sub-verzí). Nová verze většinou nabízí nové funkčnosti (či vylepšení těch stávajících) systému, jako jsou například zvýšená stabilita OS, menší energetická náročnosti zařízení či různé grafické úpravy uživatelského rozhraní. V některých aktualizacích systému byly například přidány podpory nových senzorů, které mohou zařízení od této verze obsahovat, respektive využívat. Například senzor *TYPE_ORIENTATION* mohl být v zařízení zabudován již od verze *API 3*, avšak programově využíván mohl být až později, od verze *API 9*.

Jak ukazuje následující přehled (tabulka 2.2 na straně 5), většina uživatelů má na svém zařízení OS s *API* verze *15* či vyšší, pouze necelých 7% uživatelů využívá verze *API 10* a menší (poznámka: ještě v lednu roku 2015 se jednalo o přibližně 8.5% uživatelů – z toho si lze snadno vydedukovat, že nástup nových, respektive úbytek starých, verzí systému je relativně rychlý).

Verze	Jméno	API	Podíl verze
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	6.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.7%
4.1.x	Jelly Bean	16	16.5%
4.2.x		17	18.6%
4.3.x		18	5.6%
4.4	KitKat	19	41.4%
5.0	Lollipop	21	5.0%
5.1		22	0.4%

Tabulka 2.2: Podíl verzí Androidu k 6. dubnu 2015. [9]

Kvůli absenci aplikace *Google Play Store* (vyžadující verzi *API 8* a vyšší) nejsou do tabulky 2.2 zahrnuty přístroje obsahující systém Android v nižší verzi než 2.2. Celkový součet zařízení s těmito verzemi však činí přibližně 1% ze všech dostupných verzí (datováno k srpnu 2013).

3 Vybrané aplikace

Při navrhování *knihovny* pro aplikaci, která sleduje pohyb uživatele, je vhodné prozkoumat již dostupné aplikace, které se zabývají podobným tématem a nabízejí tak podobnou funkcionalitu. Jejich průzkumem bylo navíc možné doplnit, jakou funkcionalitu by mohla výsledná práce poskytovat. V této sekci bude představeno několik vybraných pohybových aplikací.

3.1 Runastic

Jde o fitness aplikaci, která uživateli poskytuje mnoho informací o jeho pohybu. Je tedy zaměřena především pro sportovní využití. Ke sledování uživatele využívá systému GPS a dokáže zaznamenávat několik různých hodnot:

- měření uběhnuté vzdálenosti
- dobu trvání pohybu
- rychlost pohybu
- tempo
- spálené kalorie

Taktéž nabízí různé grafy zaznamenaných hodnot, jako například graf rychlosti uživatele při pohybu, graf stoupání či tep uživatele při sportování. Dále nabízí i několik dalších měření v oblasti zdraví uživatele, to však již není pro tuto práci podstatné.

Odkaz na stránky aplikace: `https://www.runtastic.com/cs/apps/runtastic`

3.2 Pedometer

Tato aplikace slouží hlavně jako krokoměr. Uživateli poskytuje, podobně jako výše zmíněná aplikace, několik grafů a různých údajů. Těmi jsou například doba chůze, počet ušlých kroků, spálené kalorie či rychlost chůze.

Odkaz na aplikaci v obchodě Google play:
`https://play.google.com/store/apps/details?id=com.tayu.tau.pedometer`

3.3 MyTracks

Další alternativa k pohybovým aplikacím, jež využívají systému GPS k získávání informací o pohybu uživatele. Aplikace zaznamenává trasu pohybu uživatele, rychlost pohybu, vzdálenost a nadmořskou výšku. Rozpoznat dokáže chůzi, běh, jízdu na kole a další venkovní aktivity. Během záznamu lze sledovat aktuální data, poslouchat hlasová oznámení aplikace či přidávat si textové poznámky k procházené trase. Aplikace je také kompatibilní se zařízeními zvanými *Android Wear*, tedy zařízeními jako jsou např. chytré hodinky.

Odkaz na aplikaci v obchodě Google play:
`https://play.google.com/store/apps/details?id=com.google.android.maps.mytracks&hl=cs`

3.4 Automatic

Tuto aplikaci lze taktéž zařadit mezi pohybové, plní však trochu odlišný účel, než je klasifikace pohybu uživatele. Proto je zde uvedena spíše jen pro zajímavost. Aplikace je určená pro motoristy a pokud se řidič chová podle jejích rad, dokáže například šetřit palivo při jízdě. Při zaparkování si umí zapamatovat pozici auta, umí zobrazit nejbližší benzínové stanice či upozorní řidiče na nedostatek paliva (ještě před rozsvícením kontrolky). Po připojení k počítači automobilu dokáže detekovat například vady v motoru a řidiči následně poradit či nabídnout pomoc. Při havárii dokáže automaticky zavolat záchrannou službu atd.

Více informací o aplikaci na adrese: `https://www.automatic.com/features/`

3.5 GPS Tracker

Jedná se o prototyp aplikace, která opět shromažďuje informace o uživatelově pohybu. V reálném čase tak nabídne klasifikaci pohybu a měření např. rychlosti pohybu či polohy uživatele v čase. Pohyb je v reálném čase zaznamenáván do *Google Maps* a dává tak uživateli grafický přehled, jakým způsobem se kde pohyboval. Rozpoznat dokáže tyto druhy pohybu: chůze, běh, jízda na kole, jízda autem, jízda metrem, jízda autobusem a poloha v klidu. K rozpoznávání pohybu používá GPS senzor a tzv. *machine learning* techniky (jde o takzvané *strojové učení*, které je podoblastí umělé inteligence a zabývá se algoritmy a technikami, které umožňují počítačovému systému „učit se“). Výsledná mapa pohybu může vypadat například jako ta na obrázku 3.1.

Obrázek 3.1: Ukázka aplikace GPS Tracker

Více informací o aplikaci lze nalézt na webu `http://galaxy.hua.gr/~it20934/`

3.6 Mobility Classification System

Opět jde již o klasickou klasifikační aplikaci, jež rozpoznává pohyb uživatele a zaznamenává ho do mapy. Ke správnému rozpoznání pohybu uživatele využívá senzorů GPS, akcelerometru a případně i Wi-Fi *access pointů*. Rozpoznat dokáže čtyři druhy pohybů: chůze, běh, jízda autem a nečinnost. Nasbíraná

data umí opět vizualizovat a zobrazit je tak uživateli v přehledných grafech a mapkách.

Další informace k nalezení na stránkách:
`http://mobilizelabs.org/mobility-classification-system-0`

3.7 ActiTracker

Jako poslední příklad aplikace zabývající se rozpoznáváním pohybu je ActiTracker. Stejně jako některé z předchozích aplikací, umí opět klasifikovat pěší pohyb uživatele, včetně rozpoznání chůze do schodů či zda uživatel stojí nebo sedí. K tomuto využívá hlavně *akcelerometru* v mobilním zařízení. Zároveň také patří k aplikacím, které hlídají zdraví uživatele. Například při dlouhém sezení, na jednom místě bez pohybu, uživatele upozorní, že by se měl nějakým způsobem hýbat.

Více informací, spolu se zajímavými grafy z globálních dat, lze nalézt na stránkách výrobce na adrese `https://actitracker.com/about.php`

3.8 Shrnutí vybraných aplikací

Během prozkoumávání možností již dostupných aplikací bylo získáno několik nápadů, jakým směrem by se následně vytvářená *knihovna* měla ubírat. Jejím hlavním účelem tedy bude rozpoznání a klasifikace pohybu uživatele (minimálně do 4 různých kategorií, jako je chůze, nečinnost, jízda autem či jízda na kole). Zajímavou inspirací byla také aplikace **ActiTracker**, která dokáže uživatele upozornit při jeho delší nečinnosti bez pohybu. Využito by také mohlo být získávání přesné adresy uživatele, díky čemuž by šlo jeho pohyb (včetně druhu pohybu) zaznamenávat do mapy – toto je však již spíše návrh na možné další rozšíření základu knihovny.

Většina těchto vyjmenovancých aplikací je dostupná zdarma, pouze **Automatic** je dostupná za přibližně 100 dolarů.

4 Senzory

Tato kapitola se bude zabírat informacemi ohledně senzorů obsažených v zařízení. V dnešní době obsahují mobilní zařízení velké množství různých senzorů, jež mohou uživateli zjednodušit komfort v používání zařízení či poskytnout dodatečné funkce při používání zařízení. Některé z těchto senzorů lze zařadit mezi senzory, které slouží pro detekci pohybu. Těmi mohou být například akcelerometr, gyroskop, senzor rotace zařízení, senzor tlaku či teploty a podobně. Záleží však pouze na výrobci a na typu zařízení, které z těchto dostupných senzorů bude zařízení obsahovat. Informace o senzorech byly čerpány ze zdrojů [5], [6] a [7].

4.1 Seznam senzorů

Následující tabulka 4.1 obsahuje výpis několika vybraných typů senzorů. První sloupec značí hardwarové označení senzoru, druhý typ senzoru, třetí jeho stručný popis a čtvrtý způsob využití senzoru.

Některé z těchto jmenovaných senzorů byly nahrazeny novějšími, což způsobilo jejich takzvané „zastarání“, jsou tedy označovány jako *deprecated*. S takto označenými senzory lze i nadále pracovat (kvůli kompatibilitě se staršími zařízeními), je však doporučeno je již nepoužívat a využít novějších, modernějších, verzí. Pokud i nadále chceme využívat verzí starších, je nutné jejich použití v kódu aplikace náležitě označit.

Tabulka 4.2 na straně 13 ukazuje, v jakém *API* jsou výše jmenované senzory dostupné a kdy došlo k jejich případnému „zastarání“.

Senzor	**Typ**	**Popis**	**Běžné použití**
ACCELEROMETER	HW	Měří akcelerační sílu (v m/s^2), která je aplikována na zařízení ve všech jeho fyzických osách (x, y a z), včetně gravitační síly.	Detekce pohybu (otřesy, naklánění, atd.).
AMBIENT_TEMPERATURE	HW	Měří okolní teplotu ve stupních Celsia (°C).	Monitorování teploty vzduchu.
GRAVITY	SW či HW	Měří gravitační sílu v m/s^2, která je aplikována na zařízení ve všech jeho fyzických osách (x, y a z).	Detekce pohybu (otřesy, naklánění, atd.).
GYROSCOPE	HW	Měří míru rotace zařízení (v rad/s) okolo všech jeho fyzických os (x, y a z).	Detekce rotace (otočení, náklon, atd.)
LIGHT	HW	Měří hodnotu okolního osvětlení v lx.	Ovládání podsvícení zařízení.
LINEAR_ACCELERATION	SW či HW	Měří akcelerační sílu (v m/s^2), která je aplikována na zařízení ve všech jeho fyzických osách (x, y a z), vyjma gravitační síly.	Měření akcelerace v rámci jedné osy.
MAGNETIC_FIELD	HW	Měří okolní geomagnetické pole všech tří fyzických os zařízení (x, y a z) v μT.	Tvorba kompasu.
ORIENTATION	HW	Měří míru rotace zařízení okolo jeho fyzických os (x, y a z). Od *API 3* lze získat matici sklonu zařízení a matici rotace zařízení použitím *gravity* a *geomagnetic field* senzorů díky metodě *getRorationMatrix()*.	Určení pozice zařízení.

Senzor	**Typ**	**Popis**	**Běžné použití**
PRESSURE	HW	Měří okolní tlak vzduchu v hPa či $mbar$.	Měření změn tlaku okolního vzduchu.
PROXIMITY	HW	Měří vzdálenost objektu v cm vzhledem k obrazovce displeje zařízení. Typicky je tento senzor použit pro zjištění, zda je přístroj držen u ucha při telefonování.	Pozice telefonu během hovoru.
RELATIVE_HUMIDITY	HW	Měří relativní okolní vlhkost v procentech (%).	Měření rosného bodu a absolutní či relativní vlhkosti.
ROTATION_VECTOR	SW či HW	Měří orientaci zařízení tím, že poskytuje vektor natočení zařízení.	Detekce pohybu a rotace.
TEMPERATURE	HW	Měří teplotu zařízení ve stupních Celsia (°C). Implementace tohoto senzoru může být mezi zařízeními různá. Senzor byl v *API 14* nahrazen senzorem AMBIENT_TEMPERATURE.	Měření teplot.

Tabulka 4.1: Seznam vybraných senzorů [7].

Při práci se senzory byl zprvu využíván starší mobilní telefon, který obsahoval Android ve verzi 2.3 (Gingerbread) a jen některé z výše uvedených senzorů. Jelikož se jednalo o přístroj patřící mezi ty levnější, byly jeho senzory méně přesné, avšak k základnímu seznámení se s nimi a způsobu, jak k nim v kódu aplikace přistupovat a získat z nich data při jejich používání, postačil. Poté byl již na testování využíván modernější mobilní telefon (konkrétně Sony Xperia Z1 Compact), který obsahuje téměř nejnovější Android KitKat (verze 4.4) a poskytuje více senzorů, které jsou zároveň přesnější, než ty u předchozího modelu.

Senzor	API 14	API 9	API8	API 3
ACCELEROMETER	Ano	Ano	Ano	Ano
AMBIENT_TEMPERATURE	Ano	n/a	n/a	n/a
GRAVITY	Ano	Ano	n/a	n/a
GYROSCOPE	Ano	Ano	n/a[3]	n/a[3]
LIGHT	Ano	Ano	Ano	Ano
LINEAR_ACCELERATION	Ano	Ano	n/a	n/a
MAGNETIC_FIELD	Ano	Ano	Ano	Ano
ORIENTATION	Ano[4]	Ano[4]	Ano[4]	Ano
PRESSURE	Ano	Ano	n/a[3]	n/a[3]
PROXIMITY	Ano	Ano	Ano	Ano
RELATIVE_HUMIDITY	Ano	n/a	n/a	n/a
ROTATION_VECTOR	Ano	Ano	n/a	n/a
TEMPERATURE	Ano[4]	Ano	Ano	Ano

[3]Senzor tohoto typu byl přidán již v *API 3*, dostupný však byl až od verze *API 9*.
[4]Tento typ senzoru je stále dostupný, patří však mezi *zastaralé* senzory.

Tabulka 4.2: Požadované API jednotlivých senzorů [7].

4.2 Využití senzorů prakticky

Aby bylo možné přistupovat k senzorům, je nutné použít *Android sensor framework*. Ten je součástí balíčku *android.hardware* (více informací o balíčku lze najít na adrese `http://developer.android.com/reference/android/hardware/package-summary.html`) a obsahuje následující *třídy* a *rozhraní*:

- Sensor manager
  Použitím této *třídy* vytvoříme *instanci* práce se senzorem. Poskytuje *metody* pro přístup k senzorům, jejich „naslouchání“, registraci či odregistrování *event listenerů*. Obsahuje také různé *konstanty* použité pro získání přesnosti senzoru, nastavení rychlosti snímání či kalibraci senzoru.

- Sensor
  Touto *třídou* vytvoříme *instanci* konkrétního senzoru. *Třída* poskytuje různé *metody* pro stanovení vlastností senzoru.

- SensorEvent
  Tuto *třídu* používá systém k vytvoření *sensor event* objektu, který poskytuje následující informace o událostech senzoru: přímá data ze sen-

zoru, typ senzoru, který spustil tuto událost, přesnost dat a přesný čas, kdy byla data vygenerována.

- SensorEventListener
  Toto *rozhraní* lze použít pro vytvoření *metod*, které zachytí a zpracují notifikaci od senzoru, pokud nastane změna hodnot, které senzor generuje, či pokud nastane změna přesnosti senzoru.

Nejprve si tedy vytvoříme *instanci SensorManageru* a jako *argument* jí předáme *SENSOR_SERVICE*:

```
private SensorManager mSensorManager;
mSensorManager = (SensorManager) getSystemService
                 (Context.SENSOR_SERVICE);
```

Pokud chceme získat *list* všech dostupných senzorů na zařízení, stačí použít následující kód:

```
List<Sensor> deviceSensors = mSensorManager.getSensorList
                             (Sensor.TYPE_ALL);
```

Pro zjištění dostupnosti některého ze senzorů je použit následující kód (musíme mít získanou *instanci SensorManageru* – viz výše). Zde je testována přítomnost *PROXIMITY* senzoru:

```
if (mSensorManager.getDefaultSensor
  (Sensor.TYPE_PROXIMITY) != null) {
// Ano, tento senzor se zde nachází.
}
else {
// Tento senzor není bohužel dostupný. }
```

Když je ověřena přítomnost konkrétního senzoru, lze získat jeho *instanci*, pomocí *SensorManageru*, takto:

```
private Sensor senz;
...
senz = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
```

Pomocí tohoto *manageru* lze následně získat různé informace ze senzoru, například minimální časový interval, ve kterém je senzor schopný poskytovat jeho data. Učiníme tak použitím *metody getMinDelay()*. Výsledkem této *metody* je minimální časový interval v milisekundách, ve kterém je senzor schopný svá data poskytovat.

```
int time = senz.getMinDelay();
```

Pokud chceme získávat data ze senzoru průběžně, v reálnem čase, musí naše *aktivita* implementovat *SensorEventListener*. Musíme použít (a *překrýt*) *metody onSensorChanged()* a *onAccuracyChanged()*. V první jmenované *metodě* získáme „update“ pokaždé, když senzor vygeneruje novou hodnotu jeho dat. V té druhé, pokud nastala změněna přesnosti senzoru. Konkrétně tedy:

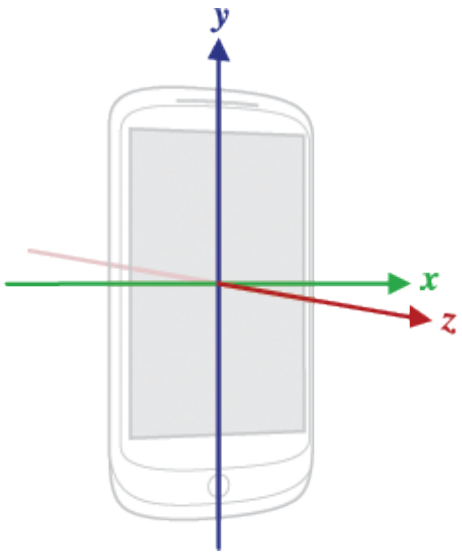
```
@Override
public final void onAccuracyChanged(Sensor sensor, int accuracy) {
// Dělej něco, pokud se změnila přesnost senzoru.
}

@Override
public final void onSensorChanged(SensorEvent event) {
// Proximity senzor vrací jen jednu hodnotu.
// Mnoho senzorů však vrací hodnoty 3, pro každou z os přístroje.
int distance = (int)event.values[0];
// Dělej něco se získanou hodnotou.
}
```

Jak bylo zmíněno v kódu výše, některé senzory vrací pouze jednu hodnotu, jiné až tři různé hodnoty. Například uvedený *proximity* senzor (viz tabulka 4.1 – měří vzdálenost od blízkého objektu). Následující senzory generují až tři hodnoty, pro každou z os natočení přístroje:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

Osy X, Y a Z jsou umístěny takto (pokud držíme přístroj před sebou, displejem otočeným směrem k obličeji, „postaveným na výšku“):
– osa X směřuje zleva doprava
– osa Y od spodního okraje přístroje k vrchnímu
– osa Z vyčnívá zpoza přístroje směrem k obličeji

Lépe si to lze představit na obrázku 4.1. Nejpodstatnější však je si uvědomit, že osy zůstávají rozloženy stále stejně, dojde-li ke změně orientace obrazovky přístroje. To znamená, že souřadný systém zařízení se nemění s tím, jak se zařízení hýbe. Tento samý souřadný systém používá i *OpenGL* (`https://www.opengl.org/`).



Obrázek 4.1: Souřadný systém v přístroji [7].

5 Návrh knihovny

Tato kapitola se bude zaobírat návrhem *knihovny*, která bude využívat pohybových senzorů a dostupných *API*. Díky tomu bude možné tuto *knihovnu* využívat při následném vyvíjení aplikací založených na rozpoznávání pohybu uživatele zařízení. Její využití ušetří uživateli mnoho času a řádek kódu, bude stačit využívat jejích předem připravených *metod*.

5.1 Funkce

Navrhovaná *knihovna* tedy bude schopná rozpoznat konkrétní aktivitu, neboli typ pohybu uživatele (jízda dopravním prostředkem, jízda na kole, chůze, nečinnost a naklápění zařízení). Závisle na tomto předpokladu do ní budou dále programovány další funkcionality. Protože využití přímých dat poskytovaných senzory pro rozpoznání aktivity uživatele by bylo příliš složité, bude v této *knihovně* využíváno veřejne dostupného *Google API*. Toto *API* sice také využívá jednotlivých senzorů zařízení, avšak jako výsledek poskytuje již „vypočítaná“ data, na která byly použity algoritmy obsažené v tomto *API*.

Aktivita uživatele je vždy rozpoznáná s určitou pravděpodobností, uživateli *knihovny* tedy bude poskytnuta možnost si tuto pravděpodobnost nastavit dle vlastního uvážení. Implicitně je procentuální hodnota rozpoznané aktivity nastavena na 60% z důvodu, že ne vždy jsou získaná data zcela přesná. Tím bude zamezeno akceptaci špatně rozpoznané aktivity. Pokud tedy následně získáme akceptovanou aktivitu, bude zaznamenána doba, po kterou byla tato aktivita „aktivní“. Tyto časy budou zaznamenávány pro každou z rozpoznatelných aktivit a budou ukládány do paměti zařízení. Všechna takto nasbíraná data půjde jednoduše zobrazit, případně si nechat zobrazit procentuální podíl časů jednotlivých aktivit.

Další funkcí navrhované *knihovny* bude funkce zazvonění, pokud uživatel setrvává po delší dobu na jednom místě bez pohybu. Jednoduše tak uživatele upozorní, že se dlouho nehýbe a měl by se nějakým způsobem „protáhnout“. Toto lze využít v aplikacích starajících se o zdraví uživatele, jenž je používá.

V rámci využívání *Google API* bude do *knihovny* implementována možnost nechat si zobrazit svou aktuální polohu či adresu. Jelikož však jde již o

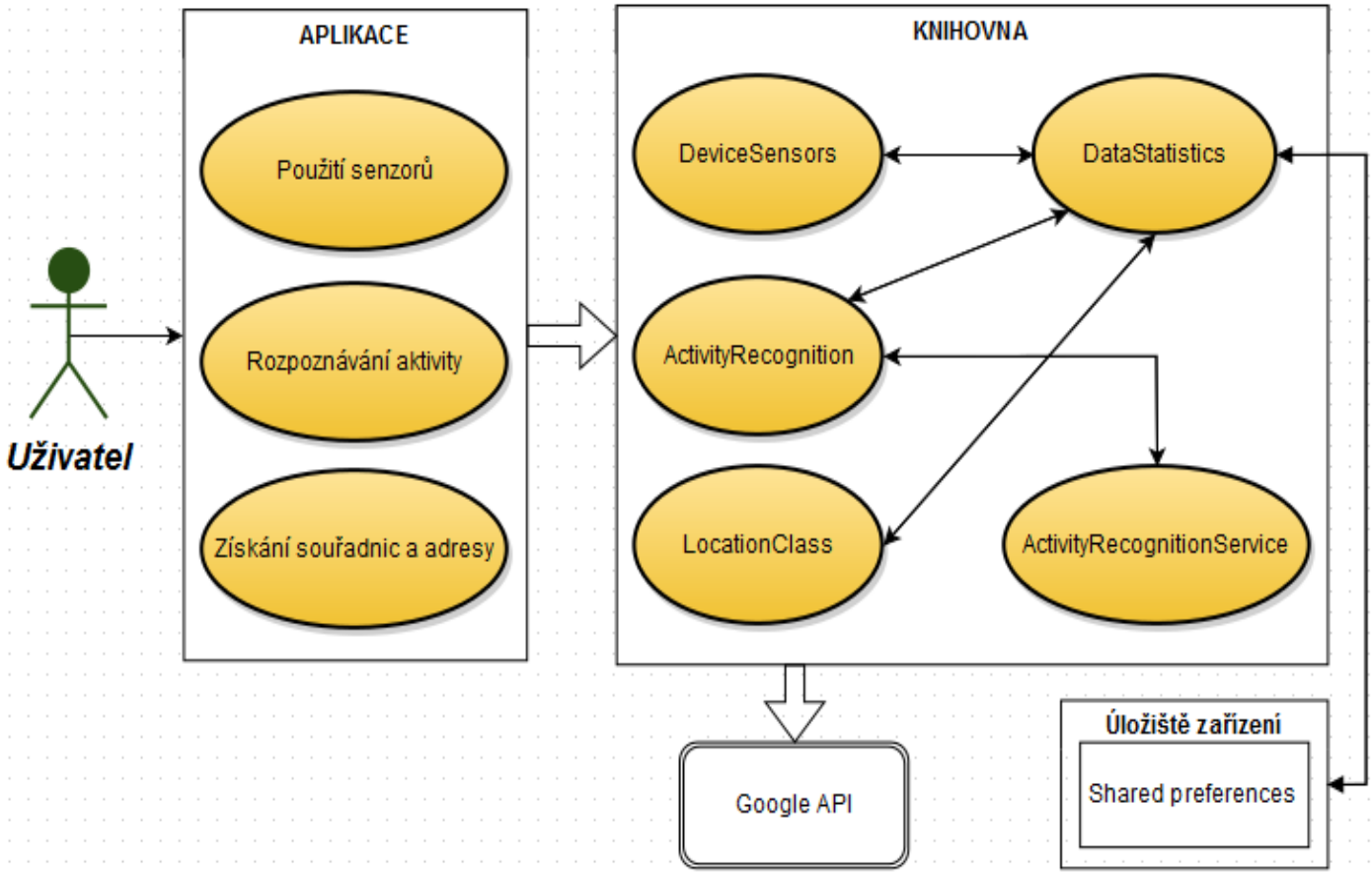
spíše okrajovou část této práce, která souvisí se zadáním jen částečně, budou její další možnosti rozebrány v kapitole 9.2 na straně 53.

Poslední hlavní funkcí *knihovny* bude využití senzorů pro vytvoření jednoduchého krokoměru. Ten lze vytvořit dvěma různými způsoby, kdy první z nich poskytne častější aktualizace senzoru, avšak menší přesnost. U druhého způsobu jsou data ze senzorů přepočítána algoritmy senzoru, což poskytuje přesnější výsledky. Díky tomu je sice aktualizace senzoru méně častá, avšak přesto naprosto dostačující. Navíc je zde větší záruka správně spočítaných kroků. *Knihovna* také bude uživateli poskytovat i přímá data z několika vybraných senzorů, jako jsou akcelerometr, senzor rotace či senzor okolního osvětlení. Mezi těmito senzory si půjde libovolně volit ten, který bude uživatele zajímat, včetně obou verzí krokoměru.

Na základě těchto uvedených funkcí byl sestaven diagram případů užití (obrázek 5.1 na straně 19), jak mezi sebou budou jednotlivé části knihovny (a testovací aplikace) komunikovat.

5.2 Požadavky

Použité *API* v této *knihovně* vyžaduje verzi Androidu nejméně 2.2 Froyo (*API level 8*) a poslední verzi *knihovny Google Play services*. Využity jsou zde však také senzory *Step Counter* a *Step detector*, které byly do systému přidány až ve verzi 4.4 KitKat (*API level 19*), což je tedy nejnižší možná verze pro následný program využívající tuto knihovnu. Pokud ale uživatel nebude požadovat využití těchto dvou senzorů, bude stačit ona starší verze 2.2.



Obrázek 5.1: Diagram případů užití

6 Realizace návrhu

V následujícím textu (část 6.1 a kapitola 7) bude prakticky popsáno, jak nainstalovat a využít výše jmenované *API*. To je poskytováno přímo společností Google, pod názvem *Google Play services location API*. Nahradilo tak původní *Android framework location API* a je důrazně doporučeno využít právě onu novější verzi. Jak *knihovnu* náležitě označit a následně ji přidat do jiné aplikace uvádí zdroj [16].

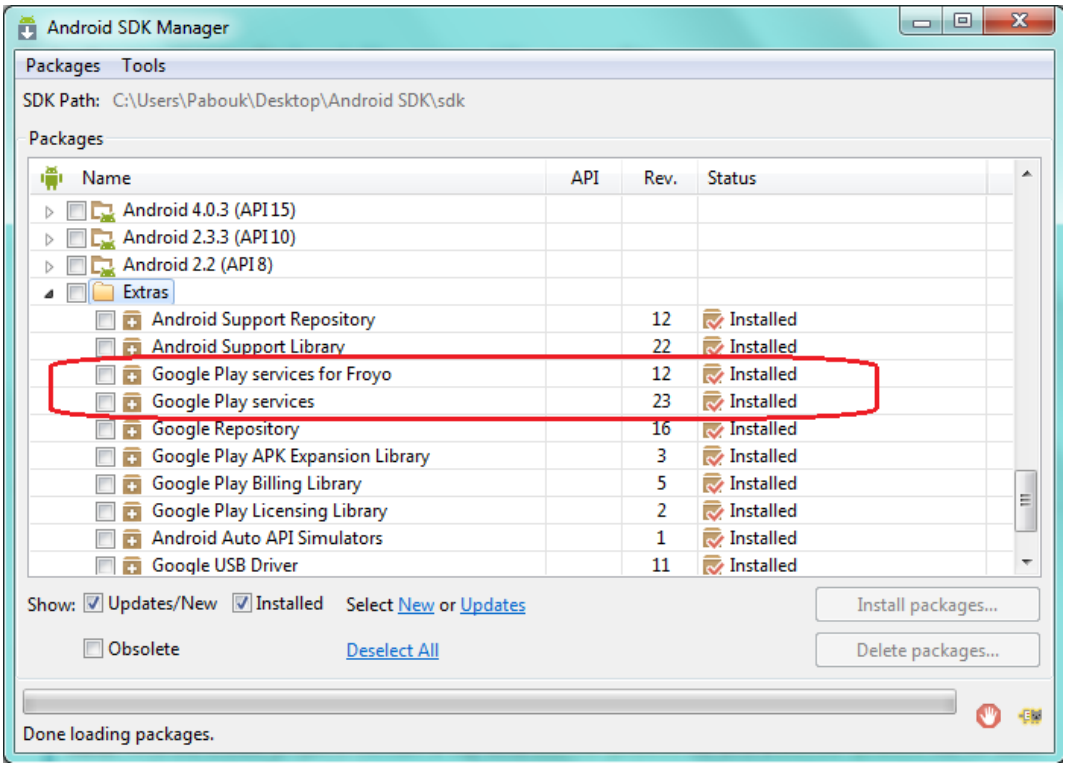
Obrázek 6.2 na straně 23 ukazuje, do kolika *tříd* je výsledná *knihovna* rozdělena. Zároveň je v něm zobrazen výčet nejdůležitějších *metod*, které každá z těchto *tříd* poskytuje. Pro jejich úplný seznam a podrobnější popis je nutné si přečíst přiloženou programátroskou dokumentaci. Podrobněji jsou tyto *třídy* popsány v kapitole 7 začínající na straně 25.

Co se rozsahu výsledného kódu samotné *knihovny* týče, dosahuje počet řádků kódu přibližně čísla 1500 (započítáno včetně všech komentářů, reálná hodnota „čistého“ kódu by mohla být tedy přibližně 1200 řádek).

Počet *metod*, které jednotlivé *třídy* obsahují, je různý. Již dříve zmíněný obrázek 6.2 ukazuje, že nejvíce *metod* obsahuje *klientská třída*. Jejich celkový počet je 27, včetně *konstruktoru, překrytých metod* a všech *metod* nastavujících či získavajících hodnoty proměnných. Druhý největší počet *metod* se nachází ve *třídě LocationClass*, přesně 23, počítaje opět všechny *metody*. *Třída DataStatistics*, pro zobrazování dat, obsahuje dohromady už jen 14 *metod*. Další *třída*, která pracuje se senzory, obsahuje také celkem 14 *metod*. Poslední, *servisní třída*, pouze *metody* 3.

6.1 Import knihovny Google Play

Aby bylo možné výše zmíněné *API* používat, je nutné si přes *Android SDK Manager* stáhnout odpovídající balíčky. Tento postup byl převzat ze zdroje [2]. Balíček *Google Play services* se nachází v záložce *Extras*, která se nachází až na konci seznamu dostupných balíčků. Poté jednoduše stačí tento balíček zvolit a příslušným tlačítkem jej (po odsouhlasení licenčních podmínek) nainstalovat. Kde přesně tento balíček nalézt si lze prohlédnout na obrázku 6.1 na straně 21.



Obrázek 6.1: Google Play services

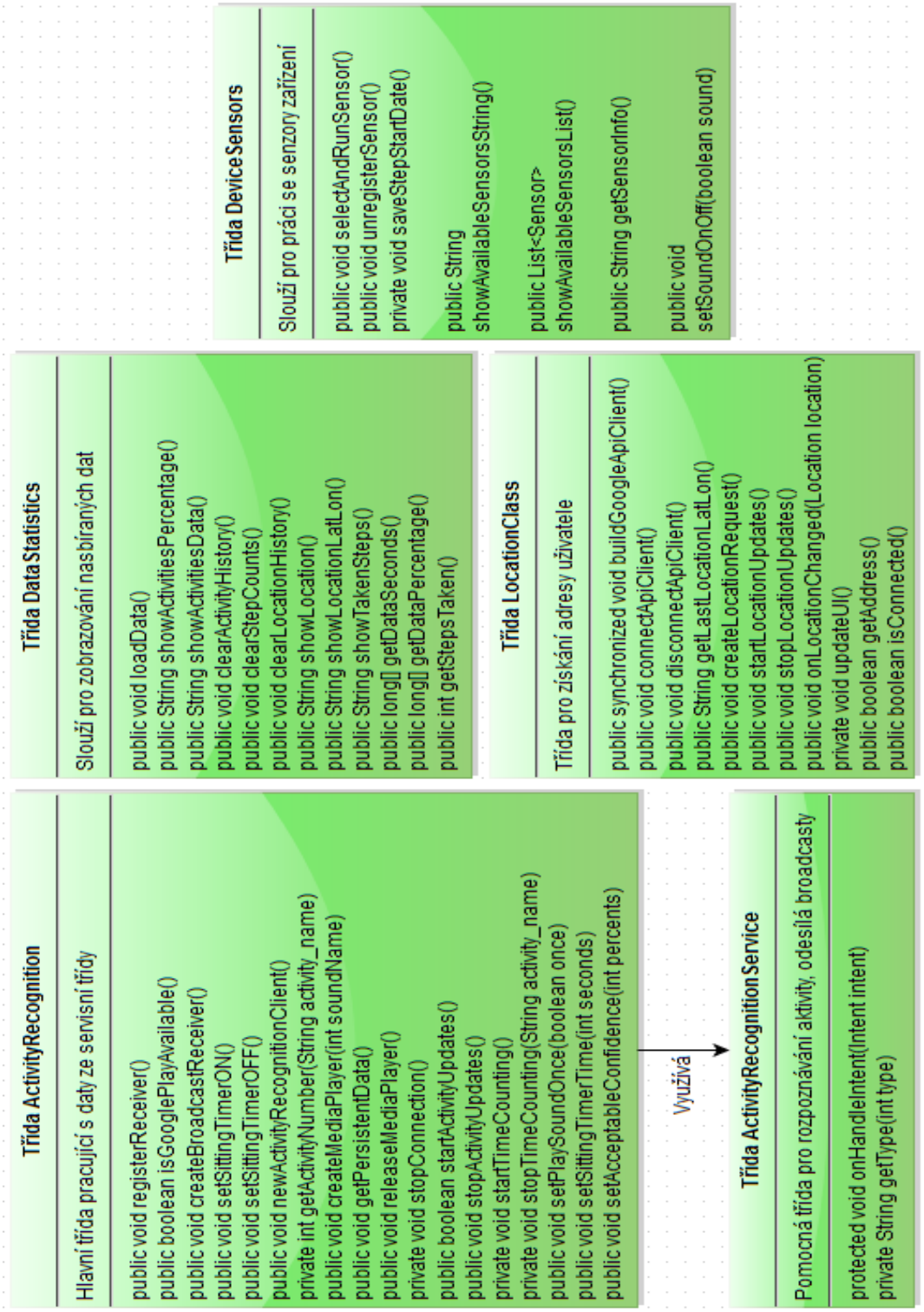
Následně je tuto knihovnu nutné nakopírovat do pracovního adresáře vývojového prostředí. *Knihovna* by se, po jejím stažení výše uvedeným postupem, měla nacházet v adresáři *SDK Manageru*, konkrétně tedy v systémové složce *<android-sdk>/extras/google/google_play_services/libproject/google-play-services_lib/*. Při použití vývojového prostředí Eclipse stačí tuto složku „přetáhnout“ do *package exploreru* v tomto prostředí. Případně ji lze do vývojového prostředí Eclipse nakopírovat tímto způsobem:

– klikneme na záložku **File** a zvolíme **Import**. Následně vybereme záložku **Android** a zde zvolíme **Existing Android Code into Workspace**. Poté už jen zvolíme cestu k dříve staženému balíčku a import dokončíme. Tímto je tedy knihovna přítomna ve vývojovém prostředí. Ještě je však nutné na tuto knihovnu odkázat v projektu (to ilustruje obrázek 6.3 na straně 24), ve kterém ji chceme využívat:

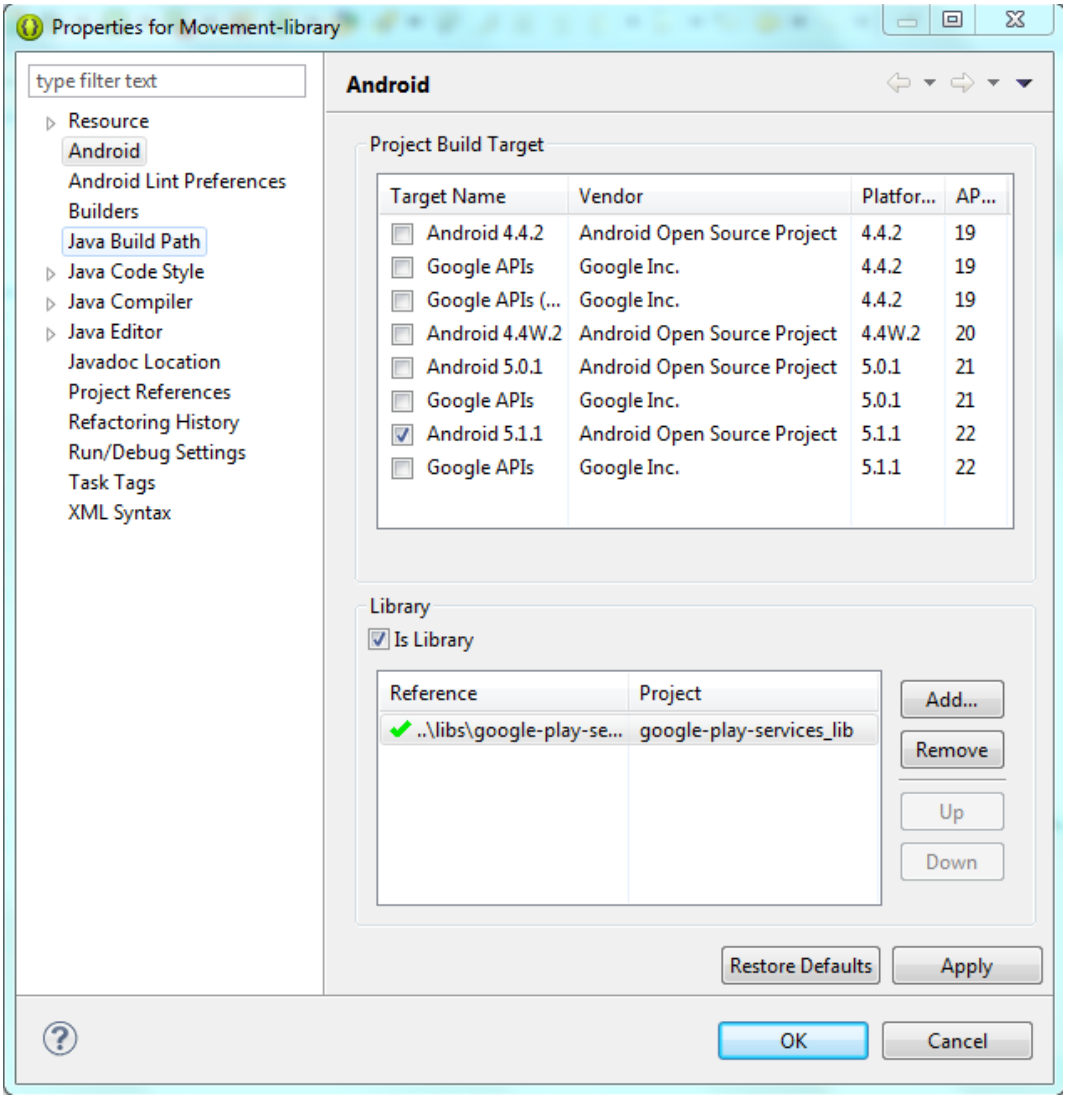
1. Projekt i knihovna se musí nacházet ve stejném vývojovém prostředí (*Workspace*).

2. V **Package exploreru** klikneme pravým tlačítkem myši na příslušný projekt, ve kterém chceme knihovnu použít, a zvolíme **Properties**.
3. V levé části okna **Properties** zvolíme položku **Android**. V jejím pravém dolním rohu se nachází okno **Library**.
4. Zde klikneme na tlačítko **Add...**. Tím se otevře dialog **Project Selection**.
5. V tomto okně zvolíme příslušnou knihovnu a vybereme ji tlačítkem **OK**.
6. Po zavření dialogu použijeme změny kliknutím na tlačítko **Apply** v okně **Properties**.
7. Nyní lze toto okno již zavřít, knihovna by měla být naimportována.

V případě použití vývojového prostředí *Android Studio*, které vzniklo relativně nedávno, je postup *importu knihovny* do pracovnáho prostředí trochu odlišný. Jeho podrobný popis je dostupný na adrese `https://developer.android.com/google/play-services/setup.html`. Stažení knihovny přes *SDK Manager* proběhne stejným způsobem.



Obrázek 6.2: Třídy knihovny



Obrázek 6.3: Odkaz na knihovnu v projektu

7 Realizace knihovny

Pro získávání pravidelných updatů aktivit je nutné vytvořit dvě *třídy*. Jedna z nich je *třída* klientská, kde probíhá zpracovávání dat, další je *třída* servisní, která při každém updatu aktivity odesílá data pomocí *broadcastu* (neboli všesměrového vysílání dat) do *třídy* klientské. Dále se v této *knihovně* nacházejí ještě další tři *třídy*. Jedna z nich slouží pro zobrazování dat uložených do úložiště přístroje, druhá pro získávání informací o aktuální poloze uživatele a poslední pro práci se senzory. Všechny tyto *třídy* budou v této sekci podrobněji popsány. Informace k této sekci byly čerpány ze zdrojů [3], [10] a [12].

7.1 Klientská třída

Aby mohla klientská *třída* využívat *knihovny Google Play services*, musí implementovat dvě *rozhraní*. Prvním z těchto rohzraní je *GooglePlayServicesClient.ConnectionCallbacks* a druhým z nich je *GooglePlayServicesClient.OnConnectionFailedListener*. Dále je nutné vyžádat si povolení rozpoznávání aktivit v souboru *AndroidManifest*. Proto do tohoto souboru musíme přidat následující kód:

```
<uses-permission android:name=
”com.google.android.gms.permission.ACTIVITY_RECOGNITION” />
```

Při následné tvorbě vlastní *knihovny* či programu je nejprve nutné zkontrolovat, zda jsou vůbec *Google Play services* dostupné. To lze provés například následující *metodou*:

```
public boolean isGooglePlayAvailable()
{
int resp = GooglePlayServicesUtil.isGooglePlayServicesAvailable(mContext);
if(resp == ConnectionResult.SUCCESS) {
return true; }
else { return false;}
}
```

Následně vytvoříme nového *Activity Recognition* klienta, který bude poté „registrován“ v klientské *třídě*, která musí implementovat dříve uvedené rozhraní:

```
private ActivityRecognitionClient arclient;
...
// Parametry jsou context, connectedListener a connectionFailedListener
arclient = new ActivityRecognitionClient(mContext, this, this);
```

7.1.1 Broadcast receiver

Aby bylo možné přijímat updaty ze servisní *třídy*, je nutné ve *třídě* klientské vytvořit *broadcastReceiver*. Ten dokáže „odchytávat“ zprávy, které jsou označeny universálním textovým řetězcem. V případě této knihovny je z přijaté zprávy získán text s názvem aktivity a číslo, které reprezentuje procentuální jistotu, s kterou byla tato aktivita určena. Další postup metody *broadcastReceiveru* je komentován v následujícím kódu:

```
private BroadcastReceiver receiver;
...
public void createBroadcastReceiver() {
receiver = new BroadcastReceiver() {
@Override
public void onReceive(Context context, Intent intent) {
String actiName = intent.getStringExtra("Activity");
int actiConfidence = intent.getExtras().getInt("Confidence");
// If the confidence is accepted...
if(actiConfidence >= ACCEPTABLE_CONFIDENCE) {
// If user enables sittingTimer and the activity is recognized as Still
if(actiName.equals("Still") && sittingTimer) {
sittingTimerStill = true; }
else {
sittingTimerStill = false; }
// First update from service class
if(firstUpdate) {
lastActivityName = actiName;
// Starts the time counting for this activity
startTimeCounting();
```

```
firstUpdate = false; }
// If the activity name has changed
if(!lastActivityName.equals(actiName)) {
// Stop the time counting for the previous activity
// and save it to shared preferences
stopTimeCounting(lastActivityName);
// Mark the new activity as current
lastActivityName = actiName;
// And start time counting for it
startTimeCounting();
} } } };
}
```

Poté je nutné tento *receiver* zaregistrovat, aby bylo možné naslouchat. Vytvoříme filtr, který bude odchytávat jen ty zprávy, které jsou označené požadovaným řetězcem. Zde řetězec *com.library.movement_library.ActivityRecognitionService*. Poté dříve *deklarovaný receiver* jednoduše zaregistrujeme.

```
public void registerReceiver() {
IntentFilter filter = new IntentFilter();
filter.addAction(
"com.library.movement_library.ActivityRecognitionService");
this.mActivity.registerReceiver(receiver, filter);
}
```

Ještě je však nutné si druhou, servisní, *třídu* „zaregistrovat“ v souboru *AndroidManifest*. Zde do *tagu „<activity>“* vložíme *tag „<service>“*, takto:

```
<activity android:name=".ActivityRecognition" ...
<service android:name=
"com.library.movement_library.ActivityRecognitionService"></service>
...
</activity>
```

7.1.2 Připojení

Následně už jen zavoláme *metodu connect* nad objektem *ActivityRecognitionClient*, který byl deklarován dříve. Všechny dříve vypsané *metody* v této sekci lze zavolat veřejnou *metodou startActivityUpdates*, který zavolá postupně každou z nich, a není tak nutné volat každou zvlášť.

```
public boolean startActivityUpdates() {
// Load data from persistent storage
getPersistentData();
firstUpdate = true;
// "Start it all"
if(isGooglePlayAvailable()) {
newActivityRecognitionClient();
createBroadcastReceiver();
registerReceiver();
arclient.connect();
return true; }
else { return false; }
}
```

S *metodou connect* souvisí ještě další *metody*, které je nutné v aplikaci mít, z důvodu implementace výše zmíněných *rozhraní*. Jde o *metody onConnectionFailed*, která v tomto případě vypíše *Toast*, pokud se spojení nevydaří a *onDisconnected*, která nemá v případě této knihovny žádný účel. Dále ještě *metoda onConnected* – ta obstará předání *Intentu servisní třídě* a vyžádá si od ní pravitelné updaty. Vypadá následovně:

```
@Override
public void onConnected(Bundle arg0) {
// If connected, "start" the service class
Intent intent = new Intent(mContext,
                    ActivityRecognitionService.class);
pIntent = PendingIntent.getService(mContext, 0, intent,
                     PendingIntent.FLAG_UPDATE_CURRENT);
arclient.requestActivityUpdates(SERVICE_CLASS_UPDATES,
                                        pIntent);
}
```

7.1.3 Úložiště zařízení

Další důležitou součástí programu je ukládání dat do úložiště. Aby bylo možné uchovávat naměřené hodnoty i po ukončení aplikace, je nutné tato data nějakým způsobem uložit. Systém Android poskytuje několik následujících typů úložišť dat (čerpáno ze zdroje [8]):

- Shared Preferences
  Ukládá data *primitivních datových typů* v podobě textový klíč-hodnota. Data jsou uložena s přístupem *private*.
- Internal Storage
  Uloží data do paměti zařízení. S těmito daty se pracuje jako se soubory a přístup k ním je opět omezen na *private*.
- External Storage
  Uložení dat na sdílené externí úložiště (například paměťová karta v zařízení). Přístup k datům je zde *public*.
- SQLite Databases
  Umožní uložit strukturovaná data do databáze s přístupem *private*.
- Network Connection
  Uloží data na webové úložiště zvolené autorem aplikace (nepoužívá tedy žádné vlastní úložiště společnosti Google).

Pro tuto *knihovnu* bylo zvoleno ukládání dat do *Shared Preferences*, kvůli jednoduché a pro tuto *knihovnu* dostačující implementaci. Fragment kódu, který se stará o ukládání dat, může vypadat například takto:

```
...
// "Získání" Shared Preferences
// PREFS_NAME je unikátní označení souboru s daty
// v této knihovně pojmenovaném jako "PreferencesActivityData"
SharedPreferences shPref =
        this.mActivity.getSharedPreferences(PREFS_NAME, 0);
// Umožní editovat data
SharedPreferences.Editor editor = shPref.edit();
...
// Uložení proměnných typu Long do úložiště, každá z nich je uložena
// pod své unikátní označení textovým řetězem
```

```
editor.putLong((activity_name + ” seconds”),
        activitySeconds[activityNumber]);
editor.putLong(”sumNumber seconds”, activitySeconds[6]);
// Potvrzení změny dat v úložišti, nesmí být opomenuto!
editor.commit();
...
```

Metoda getPersistentData načte do přístušných proměnných data, která byla za chodu aplikace uložena do úložiště zařízení. Jde o časy jednotlivých aktivit, po jak dlouhou dobu každá z nich byla „aktivní“. Tato data jsou uložena do *pole* o sedmi prvcích. Kvůli nemožnosti uložení celého pole je však nutné jednotlivé prvky uložit zvlášť, každý pod svoje unikátní textové označení. Například celkový počet vteřin, po který aplikace rozpoznávala aktivitu uživatele, je uložen pod označení „sumNumber seconds“. Stejně tak jsou v tomto úložišti uloženy i hodnoty pro počet kroků krokoměru či souřadnice a adresa uživatele. Část kódu, který se stará o načítání dat, vypadá následovně:

```
// ”Získání” Shared Preferences
// PREFS_NAME je unikátní označení souboru s daty
// v této knihovně pojmenovaném jako ”PreferencesActivityData”
SharedPreferences shPref =
        this.mActivity.getSharedPreferences(PREFS_NAME, 0);
// Umožní editovat data
SharedPreferences.Editor editor = shPref.edit();
...
// Získání proměnných typu Long z úložiště, každá z nich je uložena
// pod své unikátní označení textovým řetězem
activitySeconds[0] = shPre.getLong("In Vehicle seconds", 0);
activitySeconds[1] = shPre.getLong("On Bicycle seconds", 0);
...
```

7.1.4 Další metody

V klientské *třídě* se dále nachází *privátní metoda* pro získání čísla aktivity (*getActivityNumber*), která jako výsledek vrací číslo rozpoznané aktivity, aby bylo možné uložit její čas, po který běžela, do příslušné hodnoty v *poli* časů

aktivit. Pokud nebyla žádná aktivita rozpoznána, vrací maximální hodnotu typu *Integer*, díky čemuž lze poté následně zjistit „chybu“ a náležitě s ní naložit.

Metodou createMediaPlayer, jejímž argumentem je název zvuku uloženého ve složce *resources* aplikace, vytvoříme instanci *mediaPlayeru*. Díky němu lze následně argumentem předaný zvuk přehrát. Naopak *metodou releaseMediaPlayer* tento přehrávač „uvolníme“.

Metoda stopConnection má za úkol odpojit a odregistrovat klientskou aplikaci od veškerého „naslouchání“. Vše potřebné tak probíhá v jedné *metodě* a není nutné jich volat několik či ukončovat každou připojenou a naslouchající službu zvlášť. Zároveň tato *metoda* obstará zapsání času běhu poslední rozpoznané aktivity do příslušného místa v *poli* aktivit.

Na konec se v této *třídě* ještě nachází několik *metod* získavajících či nastavujících hodnoty příslušných proměnných *třídy*.

7.2 Servisní třída

Tato *třída* využívá *Intentu*, který získává od *třídy* klientské. Aby mohla pracovat správně, je nutné *oddědit* od *třídy IntentService*, což se děje pomocí použití slova *extends* za jménem *třídy*. Pokud toto *dědění* učiníme, je následně nutné implementovat *metodu onHandleIntent*, jejímž parametrem je právě onen předaný *Intent*. Pokud poté tato *třída* nabídne nějaký výsledek, je uložen do proměnné *„result“*. Výsledek je taktéž zaznamenán do vývojářské konzole v podobě „typ_aktivity - pravděpodobnost_aktivity“. Následně jsou tyto dvě položky uloženy do nového *Intentu* a *broadcastem* odeslány všem „přijímačům“. *Metoda* vypadá takto:

```
@Override
protected void onHandleIntent(Intent intent) {
if(ActivityRecognitionResult.hasResult(intent)) {
ActivityRecognitionResult result =
            ActivityRecognitionResult.extractResult(intent);
Log.i(TAG, getType(result.getMostProbableActivity().getType())
    +" - " + result.getMostProbableActivity().getConfidence() +"%");
Intent i = new Intent
```

```
        ("com.library.movement_library.ActivityRecognitionService");
i.putExtra("Activity",
        getType(result.getMostProbableActivity().getType()) );
i.putExtra("Confidence",
        result.getMostProbableActivity().getConfidence());

sendBroadcast(i); }
}
```

Servisní *třída* obsahuje ještě jednu *metodu (getType)*, která vrací slovně popsaný typ rozpoznané aktivity.

7.3 DataStatistics

Jde o *třídu*, která uživateli poskytne data, jenž byla dříve zaznamenána a uložena do persistentního úložiště zařízení. Způsob, jakým jsou data do tohoto úložiště ukládána, respektive získávána, si lze přečíst v sekci 7.1.3.

Zde, pomocí *metody loadData*, načteme všechna potřebná data do příslušných proměnných.

Metoda showActivitiesData vrátí textový řetězec, který na každou řádku zvlášť vypíše, jak dlouho byla která příslušná aktivita „aktivní“ a také celkový čas rozpoznávání všech aktivit dohromady. Veškerý čas je zde uveden v sekundách.

Metoda showActivitiesPercentage má podobnou funkcionalitu jako *metoda* výše zmíněná, avšak místo konkrétní doby běhu jednotlivých aktivit zobrazí procentuální podíl jejich doby běhu.

Pokud si uživatel přeje dříve uložená data vynulovat, postačí použít *metodu clearActivityHistory*. Ta jednoduše do úložiště místo půdovních hodnot vloží číslici nula, čímž původní hodnoty vynuluje. Ke stejnému účelu slouží také *metody clearStepCount* a *clearLocationHistory*. Ty mají za úkol opět vynulovat data, tentokrát počet ušlých kroků, respektive poslední získanou adresu a její souřadnice.

Nakonec se zde nachází ještě několik *metod*, které také vrací data uložená v úložisti, avšak v jiné podobě. Časy, respektive procenta, aktivit si tak lze

vyžádat jako data uložená v *poli*. Uživatel si poté může zvolit vlastní podobu, jak tato data ve finále zobrazit.

7.4 LocationClass

Další *třídou knihovny* je *LocationClass*, která má za úkol získat a uložit (či zobrazit) přesnou polohu uživatele. Aby bylo možné polohu určit, je nejprve nutné si o toto povolení zažádat v souboru *AndroidManifest*. Pokud nám postačí přibližná poloha, vložíme do něj následující:

```
<uses-permission android:name=
        ”android.permission.ACCESS_COARSE_LOCATION”/>
```

Případně tento kód, pokud chceme získat polohu co nejpřesnější (což má ovšem také větší negativní dopad na výdrž baterie). Navíc je toto přesné určení polohy nezbytné pro následné získávání přesné adresy, kde se momentálně zařízení nachází. Obě tyto možnosti vyžadují připojení k síti, dále zmíněná *FINE* varianta navíc ještě použití GPS senzoru.

```
<uses-permission android:name=
        ”android.permission.ACCESS_FINE_LOCATION”/>
```

Aby bylo možné požadovaně pracovat s *Google API*, musí tato *třída* opět implementovat několik *rohzraní*, konkrétně *ConnectionCallbacks, OnConnectionFailedListener* a *LocationListener*.

Třída také obsahuje několik různých *konstruktorů*, z důvodů jednodušší implementace při zobrazování získaných dat. Jeden z těchto *konstruktorů* obsahuje navíc odkaz na *TextView*, do kterého budou následně zobrazována získaná data. Díky tomu není nutné implementovat další servisní *třídu* a nemusí se tak vysílat a „odchytávat“ další *broadcasty*.

7.4.1 Připojení

První důležitou *metodou* je zde *buildGoogleApiClient*, jenž vytvoří novou *instanci* pro připojení ke *Google API*. Kód pro ní vypadá takto:

```
public synchronized void buildGoogleApiClient() {
mGoogleApiClient = new GoogleApiClient.Builder(this.mContext)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .addApi(LocationServices.API)
            .build();
}
```

Kódy pro samotné připojení a odpojení vypadají takto:

```
mGoogleApiClient.connect();
```

```
mGoogleApiClient.disconnect();
```

Toto připojení, respektive odpojení, je použito ve veřejné *metodě connectApiClient*, respektive *disconnectApiClient*, aby mohl uživatel tuto činnost řídit ze své aplikace.

Pokud dojde při připojování k chybě, je tato chyba „zachycena“ *překrytou metodou onConnectionFailed*. Zde je pouze uživateli pomocí *Toastu* vypsána krátká zpráva, že se připojení nezdařilo.

```
@Override
public void onConnectionFailed(ConnectionResult result) {
Toast.makeText(this.mContext, "Location connection failed!",
                    Toast.LENGTH_SHORT).show(); };
```

7.4.2 Získání souřadnic

V opačném případě, při úspěchu, je použita *překrytá metoda onConnected*. V ní následně získáme, ze dříve vytvořené instance *Google API*, přesnou lokaci uživatele, získanou v konkrétních souřadnicích.

```
private Location mLastLocation;
...
@Override
```

```
public void onConnected(Bundle connectionHint) {
mLastLocation =
LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);
}
```

Pomocí veřejné *metody getLastLocationLatLon* získáme, v případě úspěchu připojení a správného uložení souřadnic, textový řetězec se souřadnicemi, který je zároveň uložen jako poslední známá pozice do persistentního úložiště. V případě neúspěchu při připojení vrátí tato *metoda* řetězec „No coordinates found“.

```
public String getLastLocationLatLon() {
if(mLastLocation != null) {
latLonLocation =
    ("Lat: " + Double.toString(mLastLocation.getLatitude())
    + "\n Lon: " + Double.toString(mLastLocation.getLongitude()));
saveToPreferences(latLonLocation, true);
return latLonLocation;
}
else { return "No coordinates found"; }
}
```

7.4.3 Pravidelné aktualizace polohy

Pokud chceme dostávat pravitelné aktualizace naší polohy, které budou probíhat v předem určeném intervalu, je nutné vytvořit nový *LocationRequest*. Hodnoty *Interval* a *FastestInterval* lze nastavit příslušnými *metodami*, přičemž tato hodnota znamená časový interval udávaný v milisekundách.

```
public void createLocationRequest() {
mLocationRequest = new LocationRequest();
mLocationRequest.setInterval(POSITION_UPDATES_INTERVAL);
mLocationRequest.setFastestInterval(
        POSITION_UPDATES_FASTEST_INTERVAL);
mLocationRequest.setPriority(
        LocationRequest.PRIORITY_HIGH_ACCURACY);
}
```

Pokud již máme vytvořený a připojený *Google API Client*, pomocí postupu popsaného dříve, a *metodou createLocationRequest* vytvořený *LocationRequest*, stačí již zavolat následující *metodu startLocationUpdates*. Pokud si následně budeme přát tyto aktualizace odstranit, stačí *zavolat metodu stopLocationUpdates*. *Metoda* pro start získávání aktualizací polohy vypadá takto:

```
public void startLocationUpdates() {
LocationServices.FusedLocationApi.requestLocationUpdates(
                mGoogleApiClient, mLocationRequest, this);
}
```

Dále se zde nachází *překrytá metoda onLocationChanged*, která v pravidelných intervalech sleduje polohu uživatele a následně ji zobrazí do *TextView*, který byl předán v *konstruktoru*. Pokud byl však *zavolán konstruktor* bez parametru *TextView*, nedělá tato *metoda* zdánlivě nic, pouze aktualizuje hodnotu současné lokace v proměnné *mCurrentLocation*.

7.4.4 Získání adresy

Doposud byla uživateli současná adresa vždy zobrazována jako prosté souřadnice zeměpisné šířky a zeměpisné délky. Pokud však chceme zobrazit konkrétní slovní popis adresy, na níž se uživatel právě nachází, použijeme k tomu *metodu getAddress*. V té je nejprve kontrolováno, zda uživatel vlastní zařízení se systémem Android verze *Gingerbread* či vyšší a zda je v zařízení dostupný tzv. *Geocoder*, jenž umožňuje právě převod ze souřadnic zeměpisné šířky a délky na adresu. Další z podmínek je připojení k Internetu, protože *metoda* pro získání adresy toto vyžaduje. Pokud jsou tyto podmínky splněny, je vyvolán proces, který běží na pozadí aplikace. Jeho úkolem je, z naposledy získané adresy zadané souřadnicemi, získat slovní popis adresy. Pokud během tohoto procesu nenastane některá z jeho *vyjímek*, je tímto procesem navrácena adresa v následujícím formátu:

název ulice, město, země

Pokud nelze některý z těchto údajů získat, je místo něj navrácena hodnota *null*. Po dokončení zmíněného procesu na pozadí je následně *zavolána metoda onPostExecute*. V ní je získaná adresa uložena do sdíleného úložiště zařízení pod svoje universální označení a pokud byla tato *třída* vytvořena

konstruktorem obsahujícím *TextView*, je získaná adresa zobrazena i v něm.

Na konci této *třídy* najdeme opět *metody* , které nastavují či získávají hodnoty příslušných proměnných.

7.5 DeviceSensors

Tato *třída* poskytuje uživateli data z několika vybraných senzorů. Vychází z poznatků a příkladů v kapitole 4.2 na straně 13. Použití senzorů krokoměru bylo naprogramováno na základě zdroje [9]. Pro správnou funkcionalitu této třídy je nutné, aby implementovala *rozhraní SensorEventListener* a *překryla* jeho nezbytné *metody*.

V této *třídě* jsou uživateli nabídnuty tři *konstruktory*. První z nich slouží pouze k vytvoření instance pro načtení všech dostupných senzroů na zařízení. Zbylé dva jsou již složitější a obsahují několik parametrů. Oba se od sebe liší parametrem pro předání zdroje zvuku.

Metoda selectAndRunSensor má za úkol registrovat zvolený senzor. Pokud se tento senzor v zařízení nenachází, vrací tato *metoda* hodnotu *false*. Opačnou *metodou* je *unregisterSensor*, která dříve registrovaný senzor odregistruje.

Metoda showAvailableSensorsList při jejím zavolání vrátí *List* všech dostupných senzorů. *Metoda showAvailableSensorsString* funguje podobným způsobem, vrací však již řetězec pod sebou jmenovitě vypsaných dostupných senzorů zařízení.

Překrytá metoda onSensorChanged poskytuje, na základě konstruktorem zvoleného senzoru, příslušná data. Ta jsou zobrazována do předaného *TextView* při každém spuštění události senzoru. V případě akcelerometru je uživateli navíc dána možnost, zda chce zobrazovat „surová“ data, či jejich „předvypočítanou“ variantu, ze které je odstraněna gravitační konstanta působící na senzor. Tato volba je volena opět příslušnou položkou v *konstruktoru*.

Na konec následuje už jen *metoda onAccuracyChanged*, která zde nemá prakticky žádný význam, a *metody*, kterými lze získat, respektive nastavit, hodnotu potřebného zrychlení, které je nutné vyvinout při zvednutí zařízení, aby se ozval výstražný zvuk.

8 Použití v aplikaci

Tato kapitola ukazuje praktické využití *knihovny* na tvorbě jednoduché aplikace, která tak zároveň demonstruje její funkcionalitu.

Při spuštění aplikace je uživateli zobrazeno jednoduché menu se třemi tlačítky. Každé z nich spouští *aktivitu* příslušnou jeho názvu. Rozdělení *aktivit* je tedy následující:

8.1 Práce se senzory

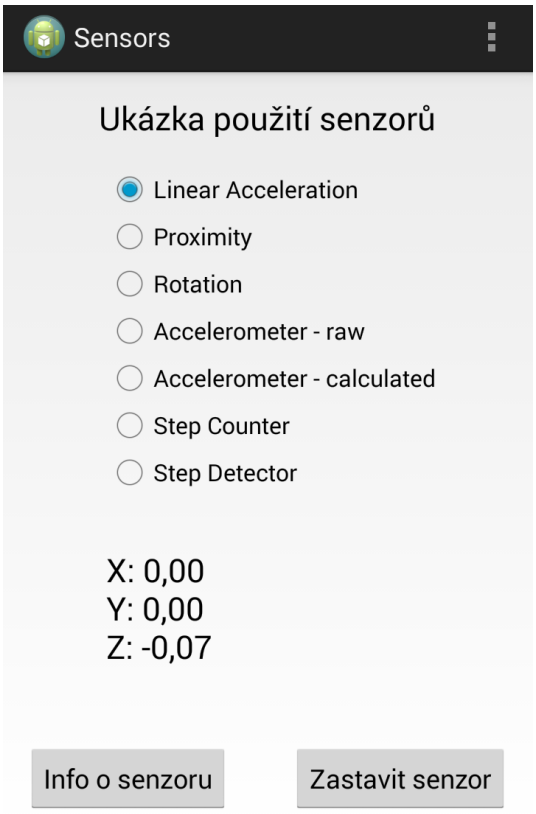
V této *aktivitě* je demonstrováno využití části *knihovny*, která se zabývá získáváním dat ze senzorů. Funguje následovně:

8.1.1 Spuštění

Při spuštění je uživateli zobrazen výčet několika vybraných senzorů, v podobě skupiny tlačítek, ze kterých lze vždy vybrat jen jeden zároveň. Jeho výběrem dojde k zaregistrování příslušného senzoru, respektive zavoláním *metody selectAndRunSensor* z testované *knihovny*. Pokud je tento senzor na zařízení dostupný, jsou data z něj získaná okamžitě zobrazována do příslušného textového prvku, který je nutné předat při volání *konstruktoru*. Výsledek si lze prohlédnou na obrázku 8.1.

8.1.2 Zastavení senzoru

Dále se zde nachází tlačítko **Zastavit senzor**, které je dostupné pouze pokud je některý ze senzorů aktivní, a jeho účelem je zavolání *metody unregisterSensor*, která zastaví a odregistruje spuštěný senzor.



Obrázek 8.1: Spuštění senzoru

8.1.3 Informace o senzoru

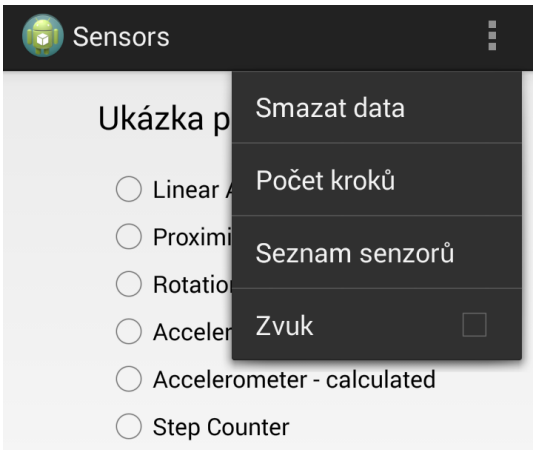
Tlačítko **Info o senzoru** slouží pro zobrazení základních informací o právě zvoleném senzoru. Informace jsou následně pomocí *metody getSensorInfo* zobrazeny do samostatného okna. Jak vypadá informace o senzoru lineární akcelerace ukazuje obrázek 8.2.

8.1.4 Menu

Do *aktivity* bylo taktéž přidáno jednoduché menu, které nabídne uživateli několik funkcí. Jako první je zde volba **Smazat data**, která, po potvrzení uživatelem, provede vynulování dat zaznamenaných senzorem krokoměru. Volbou **Počet kroků** je v dialogu uživateli zobrazen počet kroků, který byl zaznamenán senzorem krokoměru, včetně data prvního spuštění krokoměru. Dále

Obrázek 8.2: Informace o senzoru

je zde možnost nechat si zobrazit seznam všech dostupných senzorů na zařízení, stisknutím tlačítka **Seznam senzorů**. A na závěr zaškrtávací tlačítko **Zvuk**, jenž povolí, respektive zakáže, zvuk pro funkci rychlého zvednutí zařízení. Ukázku nabídky menu zobrazuje obr. 8.3.



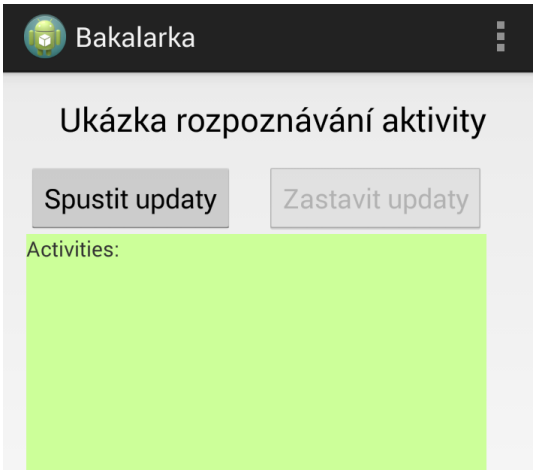
Obrázek 8.3: Menu pro práci se senzory

8.2 Rozpoznávání aktivity

Tato část aplikace se zabývá rozpoznáváním aktivity z navržené *knihovny* podrobně popsané v kapitole 7.1 na straně 25. Použití *knihovny* je zde následující:

8.2.1 onCreate

V *metodě onCreate* nejprve získáme odkazy na všechna potřebná tlačítka a textové prvky. Následně vytvoříme instance potřebných *tříd knihovny*, které budeme následně využívat k volání *metod* těchto *tříd*. Dále je zde vytvořena instance *SimpleDateFormat*, která následně umožní získat aktuální čas, jenž je taktéž zapisován spolu s updaty aktivity. Jak vypadá aplikace po spuštění ukazuje obrázek 8.4.



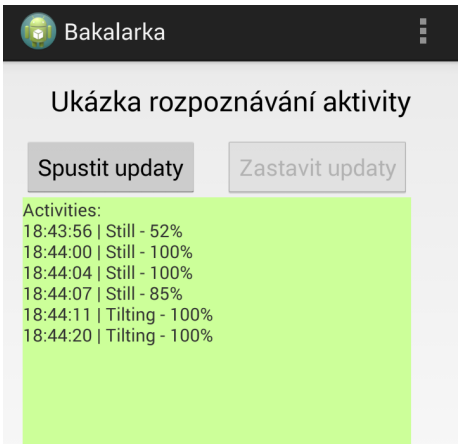
Obrázek 8.4: Rozpoznávání aktivity v okamžik spuštění

8.2.2 Spuštění updatů

Pro zahájení rozpoznávání aktivity uživatele slouží tlačítko **Spustit updaty**. Po jeho stisknutí dojde k jeho deaktivování, aby ho uživatel nemohl stisknout vícekrát, což je nežádoucí. Aby bylo možné v reálném čase sledovat aktualizace aktivit, je nutné v aplikaci vytvořit *broadcast receiver*, stejně jako tomu bylo v případě *klientské třídy*, která je popsána v kapitole 7.1 na straně 25.

Jeho vytvoření a registrace je tedy součástí funkcionality tohoto tlačítka. Aktualizace aktivity jsou vypisovány do textového prvku *Activities*, který je vytvořen tak, že při přesáhnutí textu přes okraj obrazovky jím lze rolovat. Ke každé této aktualizaci je zároveň zaznamenán čas, kdy daná aktivita začala probíhat.

Stiskem je dále vytvořen *mediaPlayer* s předaným zvukem „zvonku“, který opakovaně zazní po nastaveném čase, pokud je uživatel ve stavu *Still*. Nakonec je volána *metoda startActivityUpdates* ze *třídy ActivityRecognition*, která „připojí“ vše potřebné k získávání aktualizací aktivit. Zastavení aktualizací proběhne po stisknutí tlačítka **Stop updates**. Výpis informací po obdržení několika aktualizací ukazuje obrázek 8.5.

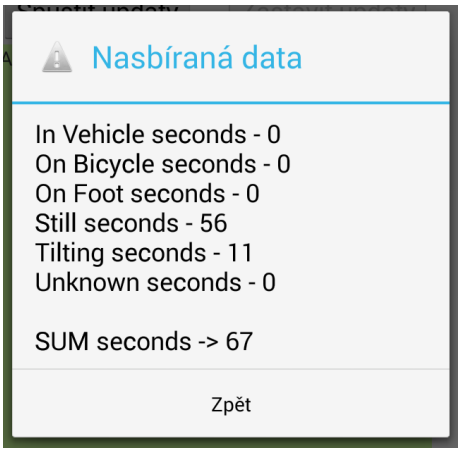


Obrázek 8.5: Aplikace po obdržení několika aktualizací

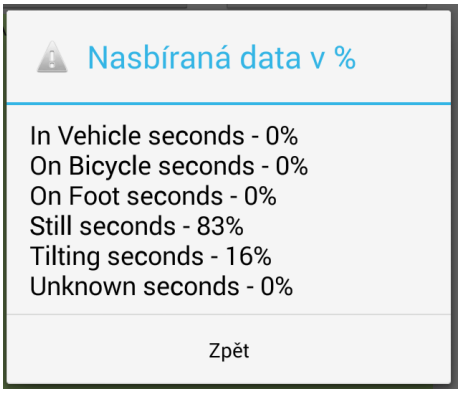
8.2.3 Výpis dat

Po „nasbírání“ určitého vzorku dat si lze tato data jednoduše zobrazit skrze menu aplikace (to si lze prohlédnout na obrázku 8.8). Kliknutím na volbu **Zobrazit data**, dojde k získání uložených dat z úložiště díky *metodě* ze *třídy DataStatistics*. Získaná data jsou následně uživateli zobrazena ve zvláštním okně. Výpis těchto dat ukazuje obrázek 8.6.

Testováním aplikace zde vyvstal „problém“, že aktivity nejsou vždy rozpoznány zcela dobře. Aplikace například může rozpoznat pohyb ve vozidle, i když jsou tato data měřena „v klidu“ u pracovního stolu. Stalo se tak i přes to, že bylo nastaveno zaznamenávání aktivit s jistotou 60% a vyšší. Někdy jednoduše stačí zařízením pohybovat specifickým způsobem, čímž dojde

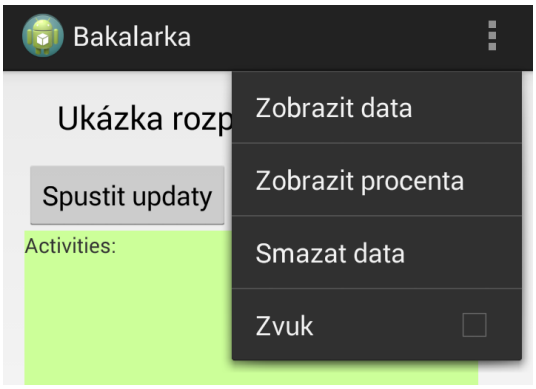


Obrázek 8.6: Nasbíraná data



Obrázek 8.7: Data v procentech

k „oklamání“ senzorů a získáním špatných dat. Toto lze zredukovat pouze nastavením akceptace při vyšší jistotě rozpoznání aktivity. Menší problém představuje také samotná rychlost rozpoznání nové aktivity – ne vždy reaguje použité *API* okamžitě a může díky tomu také dojít ke zkreslení dat, konkrétně v zaznamenané době běhu jednotlivých aktivit.



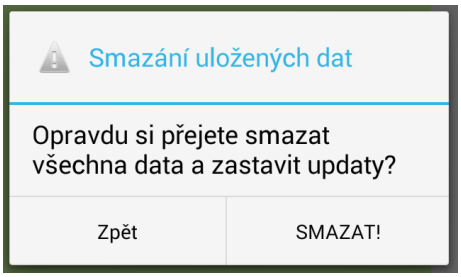
Obrázek 8.8: Menu rozpoznávání aktivity

8.2.4 Další tlačítka menu

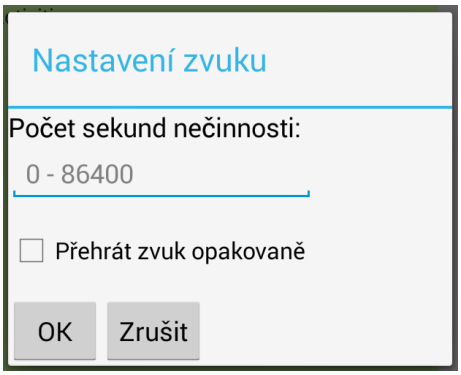
Dalším tlačítkem je v menu aplikace tlačítko **Zobrazit procenta**. Jeho stisknutím dostaneme údaje v procentech – kolik procent z celkového součtu času zaznamenávání jednotlivých aktivit tvořily které konkrétní aktivity. To ukazuje obrázek 8.7, kde procenta správně korespondují s daty na obrázku 8.6.

Následujícím tlačítkem v menu je **Smazat data**. Po jeho stisknutí je zobrazeno okno, ve kterém je uživatel dotázán, zda si opravdu přeje vynulovat všechna doposud získaná data z rozpoznávání aktivity. Pokud uživatel tuto akci potvrdí stisknutím **SMAZAT!**, dojde k přepsání všech příslušných dat v úložišti zařízení na nulové hodnoty. V opačném případě nenastane žádná akce a uživatel je navrácen zpět do aplikace v původním stavu. Způsob dotázání uživatele ukazuje obrázek 8.9.

Poslední volbou menu je **Zvuk**. Jeho stiskem se zobrazí okno, ve kterém může uživatel nastavit počet sekund, po jejichž uplynutí zazní z reproduktorů zařízení zvuk, který uživatele upozorní, že je příliš dlouho v nečinnosti. Zároveň si zde lze nastavit, zda bude tento zvuk přehráván opakovaně, dokud uživatel nezmění svou aktivitu, nebo zda se přehraje pouze jednou. Jak toto okno vypadá, ukazuje obrázek 8.10.



Obrázek 8.9: Smazání dat



Obrázek 8.10: Zapnutí zvuku aplikace

8.2.5 Překryté metody

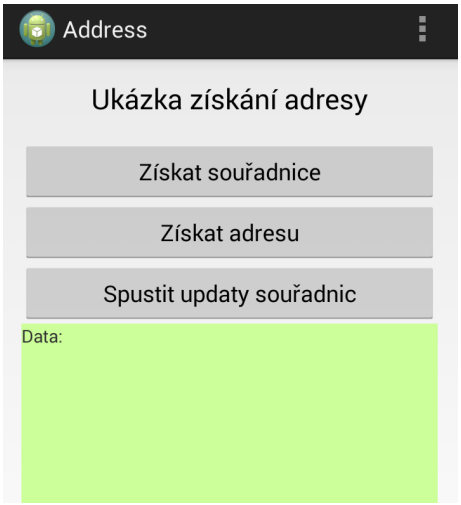
Na závěr se zde nachází ještě překrytá *metoda onDestroy*. Jejím účelem je „úklid“ po skončení aplikace. V tomto případě je jsou zde navíc *volány* všechny potřebné *metody* k tomu, aby byla všechna naslouchání a registrované intstance ukončeny. Tato implementace však již záleží pouze na uživateli *knihovny*, pokud dojde k ukončení všech naslouchání v této *metodě*, je nutné je opět po otevření aplikace znovu navazovat a vytvářet. Pozor – *metoda onDestroy* je *volána* i při pouhé změně orientace displeje zařízení. Proto je v testovací aplikaci změna orientace displeje při otáčení zakázána, aby nebylo nutné vždy po otočení displeje znovu klikat na požadovaná tlačítka.

8.3 Získání adresy

Tato část kapitoly popisuje využití navržené *knihovny* v oblasti získávání adresy.

8.3.1 onCreate

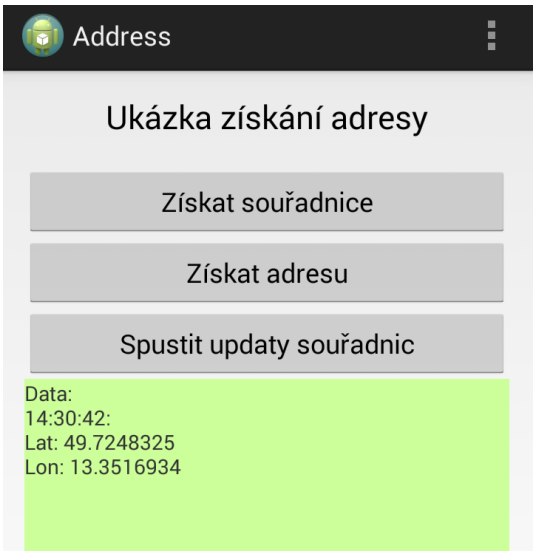
V *metodě onCreate* dojde k vytvoření instance *knihovní třídy LocationClass*. Následně je vytvořeno *Google API client* voláním *metody buildGoogleApiClient*. To je dále připojeno pomocí metody *connectApiClient*. Na konec je zavolána *metoda createLocationRequest*, která „vytvoří požadavek“ na pravidelné updaty polohy (ty je pak ještě nutné odstartovat). Jak vypadá nově spuštěná *aktivita* pro tuto část ukazuje obrázek 8.11.



Obrázek 8.11: Aktivita získání adresy

8.3.2 Získání souřadnic

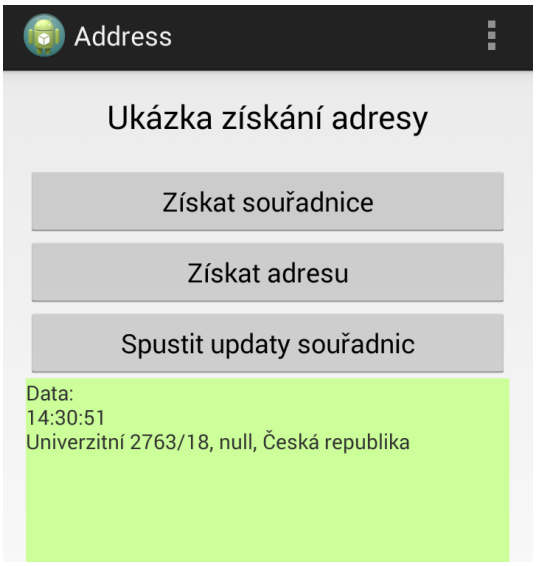
Stisknutím tlačítka **Získat souřadnice** dojde k zavolání *metody getLastLocationLatLon*, která získá poslední zaznamenané souřadnice zařízení a zobrazí je do příslušného textového prvku. To ukazuje obrázek 8.12.



Obrázek 8.12: Získání aktuálních souřadnic

8.3.3 Získání adresy

Pomocí tlačítka **Získat adresu** je volána *metoda getAddress*, která do textového prvku zobrazí slovy popsanou poslední zaznamenanou adresu zařízení, jak ukazuje obrázek 8.13.



Obrázek 8.13: Získání aktuální adresy

8.3.4 Pravidelné updaty souřadnic

Posledním tlačítkem, **Spustit updaty souřadnic**, jsou spuštěny pravidelné updaty souřadnic. Ty jsou následně v určeném časovém intervalu vypisovány do textového prvku *aktivity*. Po spuštění těchto updatů je název tlačítka změněn na **Zastavit updaty souřadnic**, které dříve spuštěné pravidelné updaty zastaví. Výpis několika automatických aktualizací si lze prohlédnou na obrázku 8.14.

Obrázek 8.14: Automatické aktualizace polohy

8.3.5 Menu

Na závěr obsahuje tato *aktivita* ještě menu o jedné položce. Tou je **Smazat data**. Jejím stisknutím je uživateli zobrazen dialog, při jehož potvrzení dojde ke smazání naposledy zaznamenaných souřadnic a polohy z úložiště zařízení.

9 Ověření funkcionality a možná rozšíření knihovny

V této kapitole bude popsán způsob, jakým byla výsledná aplikace, využívající navrženou *knihovnu*, testována, včetně různých testovacích případů. Dále zde bude uvedeno několik návrhů, jak lze výslednou *knihovnu* rozšířit.

9.1 Ověření funkcionality

Výsledná *knihovna* byla využita v samostatné aplikaci (jak podrobněji popisuje kapitola 8 na straně 38), která ověřila funkcionalitu všech *metod* poskytovaných *knihovnou*. Tato aplikace byla navíc zkoušena na několika různých mobilních zařízeních (splňujících hlavní předpoklady pro správný chod aplikace).

Testování aplikace na emulátoru přímo v PC bylo záměrně vynecháno, protože standardní emulátor, který je obsažen v instalačním balíku spolu s vývojovým prostředím Eclipse, neumožňuje simulování senzorů zařízení (což uvádí i zdroj [7]). Je zde sice možnost si toto simulování senzorů pomocí různých programů „přidat“, avšak testování na reálných zařízeních bylo shledáno jako naprosto dostačující.

9.1.1 Testování senzorů

V této části textu bude popsáno několik různých případů, jak byla testována část *knihovny* zabývající se senzory zařízení.

Proximity sensor

Tento senzor se na zařízení obvykle nachází v jeho horní části, poblíž reproduktoru, ke kterému je při telefonním hovoru přikládáno ucho. To má svůj význam v tom, že pokud senzor zaznamená ve své blízkosti nějaký objekt, zhasne při hovoru obrazovku, aby nedocházelo k náhodnému mačkání na displej zařízení.

Samotné testování probíhalo spuštěním senzoru a sledováním jeho chování. Pokud není senzor ničím zakryt, poskytuje (na testovacím zařízení Sony Xperia Z1 Compact) hodnotu přibližně 5. To znamená 5 cm, což je maximální rozsah tohoto senzoru. Pokud je před něj položena nějaká překážka, v menší vzdálenosti než zmíněných 5 cm, je zobrazovaná hodnota okamžitě změněna na 0. Většina těchto senzorů totiž funguje způsobem, že pouze zaznamená překážku a výsledkem je ano/ne, tedy jestli už se v oné vzdálenosti 0 až 5 cm nějaká překážka nachází či nikoliv. U některých zařízení může senzor fungovat ve více krocích, kdy neposkytne pouze hodnotu ano/ne, avšak ani jedno z testovacích zařízení takový senzor neobsahovalo. Samotné zakrytí senzoru je možné provést čímkoliv – senzor detekoval jak zakrytí dlaní, tak tenkým listem papíru či kovovým příborem.

Step detector/Step counter

Tyto dva senzory slouží pro počítání ušlých kroků a jejich funkcionalita je podobná. Je zde pouze ten rozdíl, že Step counter funguje o něco přesněji (a vyžaduje více energie k jeho chodu).

Testování těchto dvou senzorů probíhalo při chůzi po chodbě se zapnutým krokoměrem. Telefon byl držen v ruce ve vodorovné poloze před tělem. Chůze byla zaznamenána po pár krocích a po projití chodby bylo zaznamenáno celkem 22 kroků jedním směrem. Druhým směrem bylo zaznamenáno kroků 25. Toto měření bylo opakováno několikrát, vždy s podobnými výsledky. Proto lze závěrem konstatovat, že při chůzi tam a zpátky po té samé trase naměří detektory určitý počet kroků s jistou odchylkou. Ta může být způsobena například při otáčení se na konci chodby či prostou nepřesností senzoru, výsledek měření je však uspokojivý. Celkový počet kroků je určen přibližně správně.

Se špatným výsledkem se však setkal druhý pokus. Při něm bylo zařízení použito stejným způsobem, jako při předchozím pokusu, avšak jeho obrazovka byla zhasnuta. To mělo demostrovat využití krokoměru například při delším používání během dne. Po chůzi chodbou bylo zařízení opět „probuzeno“, avšak počet ušlých kroků zůstal roven nule. Pokus byl zopakován ještě jednou, celkem tedy dvakrát pro oba z testovaných senzorů. Výsledek byl vždy neuspokojivý, při „uspaném“ zařízení nebyl zaznamenán ani jeden krok. To je dáno tím, že tyto senzory patří do kategorie tzv. *non-wake-up* senzorů, které v rámci úspory baterie zařízení svou činnost jednoduše pozastaví, je-li zařízení „uspáno“. Řešení tohoto problému je možné použitím

programového „zámku obrazovky“ v aplikaci. To zamezí zhasnutí obrazovky, díky čemuž se tyto senzory „neuspí“.

Accelerometer

Accelerometer je senzor měřící akcelerační sílu. V jeho případě lze sledovat jak data poskytována přímo ze senzoru, tak data přepočítaná, která z dat přímých odstraňují působení gravitační síly Země.

Tento senzor byl následně testován s pomocí zvukové odezvy. V aplikaci byl tedy příslušným tlačítkem vybrán požadovaný senzor (s přepočítanými daty) a následně povolen zvuk. Po jeho zapnutí však došlo k neočekávanému chování – zvuk byl přehrán okamžitě, měl však být spuštěn až při prudším pohybu se zařízením. Následným zkoumáním bylo zjištěno, že se tak stalo proto, že okamžitě po spuštění udává senzor zkreslená data, která se „ustálí“ až po jisté době (přibližně 1 sekunda).

Na základě toho byl do knihovny přidán krátký časový interval, během kterého nebude senzor po jeho spuštění přehrávat požadovaný zvuk. Interval trvá přibližně zmíněnou jednu sekundu, po jeho uplynutí už senzor reaguje správně. Následné testování tedy probíhalo pomocí vizuální kontroly dat ze senzoru, zda korespondují s nastavenou hodnotou, při které má aplikace přehrát zvuk. Několika dalšími testy byla následně ověřena správná funkcionalita tohoto senzoru a *metody* pro přehrání zvuku.

Senzory Rotation a Linear Acceleration

Tyto zbylé dva senzory byly v aplikaci ověřeny vizuálně, zda udávají data korespondující s jejich očekávaným chováním. Senzor rotace správně udával náklon zařízení v prostoru a výstupy senzoru lineární akcelerace taktéž ukazovaly očekávané hodnoty. U senzoru lineární akcelerace byla navíc doplněna a testována funkce přehrání zvuku, jako v případě senzoru akcelerometru. Ten totiž funguje stejným způsobem jako akcelerometr s přepočítanými daty.

9.1.2 Testování rozpoznání aktivity

Tato část textu bude ověřovat hlavní funkcionalitu navržené *knihovny*, kterou je rozpoznávání aktivity uživatele. Rozpoznat lze celkem 6 druhů aktivit uživatele: jízdu ve vozidle, jízdu na kole, chůzi, naklánění zařízení, nečinnost a neznámou aktivitu.

Rozpoznávání aktivit

Testování rozpoznání aktivity probíhalo několikrát, vždy na stejné trase ve městě, po cestě z bytu do školy (cesta z lochotínských kolejí do jazykové učebny v budově ZČU v Sadech Pětatřicátníků v Plzni). Hned první den testování se však vyskytla záhadná chyba – aktivita přestala být po přibližně 8 minutách rozpoznávána. Po delším průzkumu, co a proč nefunguje, byl tento jev objasněn. Testovací zařízení (Sony Xperia Z1 Compact) nabízí zvláštní režim řízení spotřeby zařízení, tzv. režim *stamina*. Ten znatelně šetří baterii zařízení, jeho vedlejším efektem však je, že po oněch zmíněných 8 minutách uvede zařízení do „úplného spánku“, kdy zařízení sníží svou spotřebu na úplné minimum. To mělo za následek právě zastavení požadovaných updatů aktivity, což se ovšem při testování aplikace připojené k laptopu neprojevilo, protože při připojení zařízení do sítě (včetně USB portu na laptopu) je režim *stamina* automaticky deaktivován.

Při dalších testech byl tedy režim *stamina* deaktivován a testy již dopadly úspěšně. Celková doba rozpoznávání aktivity byla vždy přibližně 20 minut (v závislosti na MHD) a jednotlivé dílčí časy aktivit taktéž odpovídaly skutečnosti. Většina času zabrala chůze, následována nečinností (čekání na tramvaj) a jízdou ve vozidle (tramvaj – přibližně 7 minut). Nepatrný úsek času zabrala aktivita „otáčení přístroje“, která byla rozpoznána pravděpodobně při vkládání přístroje do kapsy kalhot. Při pár pokusech byla na krátkou dobu zaznamenáná i neznámá aktivita. Závěrem však lze konstatovat, že měření dopadla dle očekávání a vcelku přesně.

Delší neaktivita

Dále byla v rámci rozpoznávání aktivity testována další funkcionalita, kterou je upozornění uživatele, že se příliš dlouho nepohybuje. To lze využít například při „sedavé práci“, kdy zařízení uživatele zvoleným způsobem upozorní,

že by se měl začít pohybovat. To může prospívat uživatelově zdravotnímu stavu, pokud ono varování respektuje.

Testován byl případ „zazvonění“, kdy zařízení vydá zvukový signál po předem stanovené době. Nejprve tedy byl povolen zvuk (přehráván opakovaně) a následně nastaven čas (udávaný v sekundách), po kterém zařízení zazvoní. Poté bylo spuštěno rozpoznávání aktivity a zařízení položeno na stůl, aby nedošlo k přerušení „odpočtu“ při rozpoznání nečinnosti zařízení. Po uplynutí stanoveného času skutečně začalo zařízení zvukový signál opakovaně vydávat, coz potvrdilo správnou funkcionalitu.

Další test byl proveden stejným způsobem, pouze s tím rozdílem, že přehrávaný zvuk byl přehrán pouze jednou. Opět s kladným výsledkem. Dále bylo ještě testováno, zda bude zvuk přehrán po určité době, když bude nečinnost přerušena „uprostřed procesu“. Opět zde vše fungovalo dle očekávání.

Na závěr proběhl test „opakování“, tedy, zda bude zařízení vydávat zvuk i poté, pokud již jednou stanovený čas uplynul. Po zazvonění tak bylo zařízení nakláněno, aby byla rozpoznána aktivita „naklánění“, a opět položeno na stůl. Výsledkem tohoto pokusu bylo, že zařízení daný zvuk přehrálo znovu, dle očekávání.

9.1.3 Testování získání souřadnic a adresy

Posledním větším celkem navržené *knihovny* je získávání souřadnic, respektive adresy.

Získání souřadnic

Tento pokus probíhal následovně: po zapnutí aplikace byla vybrána příslušná kategorie a následně zmáčknuto příslušné tlačítko, které má za úkol zobrazit souřadnice. Nejprve bylo výsledkem prosté hlášení, že se nepodařilo získat souřadnice. To bylo způsobeno tím, že funkce získání souřadnic získává poslední známé souřadnice zařízení, nejprve je tedy nutné vůbec nějaké mít. Pro tento účel poslouží jakákoliv aplikace, která vyžaduje určení polohy uživatele. Stačí si tuto polohu nechat zaktualizovat a při dalších pokusech o získání souřadnic v testované aplikaci vše již funguje a opravdu vrací souřadnice aktuální polohy zařízení.

Získání adresy

U získání adresy je postup podobný tomu, jaký byl u získávání souřadnic. Pokud je v zařízení již načtena jeho poslední známá poloha, lze získat také přesnou adresu. Po stisku příslušného tlačítka byla ovšem opět výsledkem negativní hláška. To bylo způsobeno tím, že tato funkce potřebuje k určení přesné adresy i přístup k datové síti. Po připojení bylo tedy tlačítko opětovně zmáčknuto a výsledkem již byla slovy popsaná konkrétní adresa.

Jedinou drobností při testování bylo, že v případě získávání adresy v malé vesnici (která neobsahuje ulice), byla získána neúplná adresa, konkrétně tedy bez udání přesné ulice, místo které bylo vráceno prosté *null*. Při testech ve větším městě vše probíhalo již bez problémů, získána byla přesná adresa.

Pravidelné updaty

Poslední ověřovanou funkcionalitou bylo pravidelné získávání updatů souřadnic. Pro tento pokus je nutné na zařízení povolit ještě sledování pozice zařízení (pomocí systému GPS). Následně už jen stačilo zmáčknout příslušné tlačítko a poté začaly být vypisovány v pravidelných intervalech (nastavených na 10 sekund) aktuální souřadnice polohy zařízení.

9.2 Další možná rozšíření

Zhotovenou knihovnu je díky jejímu bohatě komentovaném kódu snadné dále rozšiřovat. Jednou z funkcionalit, kterou by bylo možné do knihovny přidat, je sledování pohybu uživatele po mapě. Například při každé změně aktivity uživatele je možné získat jeho aktuální polohu. Pokud by byla tato data v této souvislosti zaznamenávána, bylo by možné následně na mapě zobrazit, kudy a jakým způsobem se uživatel pohyboval. Díky tomu by nemělo být úplně složité zobrazit do mapy pohyb uživatele podobně, jako v případě výše jmenované aplikace GPS Tracker, jak naznačuje obrázek 3.1 na straně 8.

Rozšířit by šla například i o bezpečnostní prvek – bude sledován pohyb uživatele, například jízda na kole. Pokud by však náhle uživatel setrvával ve stavu nečinnosti, byl by upozorněn, aby svou nečinnost potvrdil. Pokud by se tak po určitou dobu nestalo, mohla by aplikace například odeslat SMS či

nějakým způsobem kontaktovat záchranné složky, že se uživateli pravděpodobně něco stalo.

Samostatná aplikace by šla také vystavět okolo, dnes velmi populárního, krokoměru. Stačilo by knihovnu dále rozšířit o zaznamenávání více dat ohledně krokoměru, jako například časy chůze či ušlou vzdálenost. Ze získaných údajů pak lze sestavit různé statistiky či například počet spálených kalorií chůzí samotnou. Pomocí využití ostatních senzorů by bylo možné například klasifikovat pohyb uživatele v budově, zda momentálně stoupá do schodů, v jakém patře se nachází či podobně.

10 Závěr

V rámci bakalářské práci bylo prozkoumáno využití senzorů mobilního operačního systémemu Android, které slouží pro detekci pohybu uživatele zařízení . V textu práce je podrobně popsáno, jak tyto senzory použít ve vlastní aplikaci a jaké chování od nich očekávat. Využito bylo několika vybraných senzorů a dále pak zkoumáno jejich chování. Mezi tyto senzory patří například akcelerometr, senzor rotace zařízení, senzory krokoměru či tzv. proximity senzor.

Dále se práce věnuje rozpoznávání konkrétního typ pohybu uživatele pomocí vybraných senzorů v mobilním zařízení. Shrnuje také praktické poznatky z testování těchto senzorů, které občas mohou být i překvapující. Tím je myšleno zejména nečekané chování senzorů (například nefunkční krokoměr při zhasnutí displeje zařízení) či jednoduše různé rozsahy a chování senzorů na různých mobilních zařízeních.

V rámci rozpoznávání pohybu poskytuje knihovna ještě možnost upozornit dlouho nečinného uživatele, že se příliš dlouho nehýbe. To knihovnu při jejím využití částečně řadí mezi ty, které se starají o zdraví uživatele.

Na základě získaných poznatků byla závěrem zhotovena knihovna, kterou lze využívat v budoucích aplikacích. Její podrobnější popis je uveden v textu práce. Hlavní přínos knihovny spočívá v tom, že ušetří jejímu uživateli psaní mnoho řádek kódu a času. Stačí pouze využívat metody, které poskytuje. Navíc se uživatel nemusí podrobněji seznamovat s prácí se senzory, respektive s použitím Google Play services API, což umožňuje využití těchto pokročilejších funkcí i začínajícím programátorům. Díky takto zjednodušenému přístupu se lze při tvorbě vlastní aplikace více věnovat například jejímu designu či další funkcionalitě.

Použité zkratky

API – Application Programming Interface

cm – centimetr

GPS – Globální polohovací systém

Inc. – Incorporation

MHD – městská hromadná doprava

OS – operační systém

PC – personal computer

SDK – Software development kit

SMS – Short message service

USB – Universal Serial Bus

Wi-Fi – wireless fidelity

ZČU – Západočeská univerzita v Plzni

Literatura

[1] *Dashboards* [online]. [cit. 18.4.2015]. Dostupné z: `https://developer.android.com/about/dashboards/index.html`.

[2] *Setting Up Google Play Services* [online]. [cit. 15.4.2015]. Dostupné z: `https://developer.android.com/google/play-services/setup.html`.

[3] *Making Your App Location-Aware* [online]. [cit. 22.3.2015]. Dostupné z: `https://developer.android.com/training/location/index.html`.

[4] *Smartphone OS Market Share, Q4 2014* [online]. [cit. 18.4.2015]. Dostupné z: `http://www.idc.com/prodserv/smartphone-os-market-share.jsp`.

[5] *Motion Sensors* [online]. [cit. 5.2.2015]. Dostupné z: `https://developer.android.com/guide/topics/sensors/sensors_motion.html`.

[6] *Position Sensors* [online]. [cit. 8.2.2015]. Dostupné z: `http://developer.android.com/guide/topics/sensors/sensors_position.html`.

[7] *Sensors Overview* [online]. [cit. 5.2.2015]. Dostupné z: `https://developer.android.com/guide/topics/sensors/sensors_overview.html`.

[8] *Storage Options* [online]. [cit. 26.2.2015]. Dostupné z: `http://developer.android.com/guide/topics/data/data-storage.html`.

[9] BAWA, H. *Create your own simple pedometer application using Android 4.4 and Nexus 5* [online]. [cit. 26.3.2015]. Dostupné z: `http://blog.bawa.com/2013/11/create-your-own-simple-pedometer.html`.

[10] KONEČNÝ, M. *Vyvíjíme pro Android: Notifikace, broadcast receivery a Internet* [online]. [cit. 12.3.2015]. Dostupné z: `http://www.zdrojak.cz/clanky/vyvijime-pro-android-notifikace-broadcast-receivery-a-internet/`.

[11] MURPHY, M. L. *The Busy Coder's Guide to Android Development*. CommonsWare, 2010. ISBN 978-0-9816780-0-9.

[12] PARMAR, K. *Android Activity Recognition Example* [online]. [cit. 19.2.2015]. Dostupné z: `http://kpbird.blogspot.cz/2013/07/android-activityrecognition-example.html`.

[13] PHILLIPS, B. – HARDY, B. *Android Programming: The Big Nerd Ranch Guide*. Big Nerd Ranch, Inc., 2013. ISBN 978-0321804334.

[14] RETO, M. *Professional Android 4 Application Development*. John Wiley & Sons, Inc., 2012. ISBN 978-1-118-26215-3 (ebk).

[15] SMITH, D. – FRIESEN, J. *Android Recipes: A Problem-Solution Approach*. Apress, 2012. ISBN 978-1430246145.

[16] VOGEL, L. *Creating libraries for Android applications - Tutorial* [online]. [cit. 19.3.2015]. Dostupné z: `http://www.vogella.com/tutorials/AndroidLibraryProjects/article.html`.

Seznam příloh

A Uživatelská příručka

Tato dokumentace slouží jako návod pro uživatele, jak nainstalovat a používat ukázkovou aplikaci. Ukázkové obrázky výsledné aplikace jsou uvedeny v různých částech kapitoly 8 začínající na straně 38.

A.1 Požadavky

Aby bylo možné aplikaci plně využívat, je nutné ji instalovat na zařízení s verzí operačního systému Android 4.4 (KitKat – API 19) a vyšší. Pro správné využívání všech dostupných funkcí je také nutné mít povolené sledování polohy zařízení a zapnuté připojení k datové síti (například Wi-Fi). Pokud uživateli nevadí absence senzorů krokoměru, je nejnižší nutná verze systému 2.2 (Froyo – API 8).

A.2 Instalace

Aplikace je publikována jako spustitelný soubor s názvem *Motion-library example* a příponou *.apk*. Tento soubor je nejprve nutné uložit do svého zařízení.

Následně musí být uživatelem povolena instalace aplikací z neznámých zdrojů. Umístění této možnosti bude u různých výrobců zařízení pravděpodobně individuální, v případě potíží s hledáním této možnosti tedy doporučuji její umístění vyhledat na Internetu. Zařízení Sony Xperia Z1 Compact nabízí tuto možnost v **Nastavení** pod záložkou **Zabezpečení**. Zde je tedy nutné zaškrtnout volbu **Neznámé zdroje**.

Po povolení instalace z neznámých zdrojů stačí vyhledat dříve uložený soubor ve svém zařízení a „ťuknutím“ na jeho ikonku zahájit instalaci. Při té je nutné povolit několik vyžadovaných funkcí, které aplikace potřebuje ke své plné funkcionalitě. Jde například o informace o datovém připojení, o získávání informací o aktivitě uživatele a podobně.

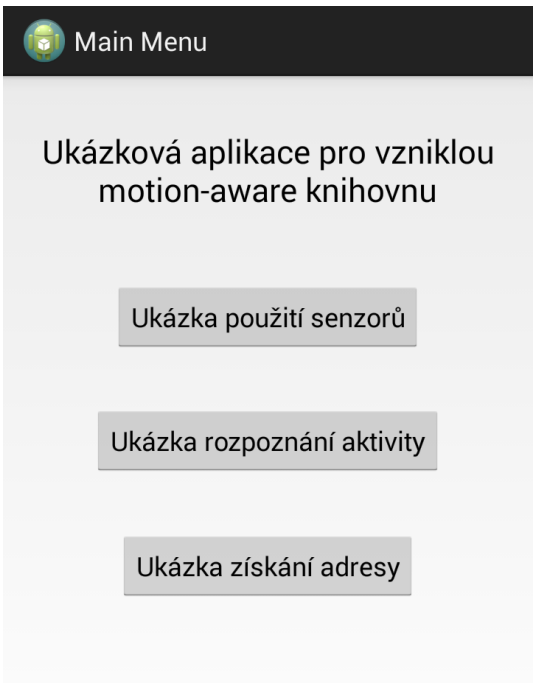
Když je tento krok odsouhlasen, aplikace je nainstalována.

A.3 Spuštění a práce s aplikací

Aplikaci spustíme jednoduše tak, že ji vyhledáme v seznamu nainstalovaných aplikací a „ťuknutím“ ji spustíme.

A.3.1 Menu

Tím se otevře menu aplikace, které nabízí tři volby – spustit část aplikace zabývající se senzory, část aplikace rozpoznávající druh pohybu uživatele a část aplikace získávající aktuální souřadnice a adresu uživatele. Jak vypadá toto menu ukazuje obrázek A.1.

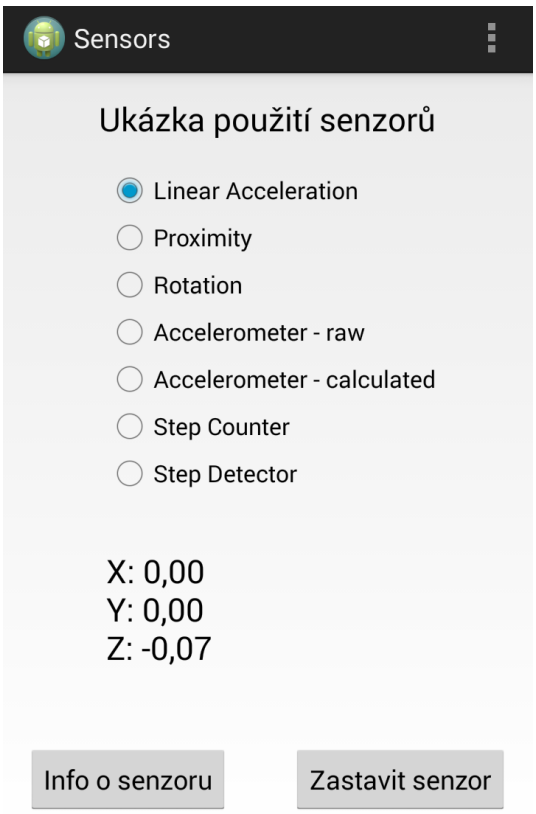


Obrázek A.1: Menu aplikace

A.3.2 Senzory

V této části se uprostřed obrazovky nachází výčet několika vybraných senzorů. Klepnutím na název některého z nich jsou následně pod tímto výčtem vypsána příslušná data (pokud je ovšem daný senzor přítomný v zařízení),

která tento senzor poskytuje. To ukazuje obrázek A.2. Pokud si uživatel přeje zjistit více informací o právě vybraném senzoru, stačí stisknout příslušné tlačítko pod seznamem senzorů. Pro získávání dat z jiného senzoru stačí jednoduše vybrat požadovaný senzor ze seznamu, není nutné před výběrem dalšího senzoru ten právě „běžící“ zastavovat. Pokud již uživatel nechce získávat data z tohoto senzoru, stiskne opět příslušné tlačítko vpravo pod seznamem senzorů.



Obrázek A.2: Spuštění lineárního akcelerometru

Dále se zde nachází jednoduché menu aplikace, pod kterým je několik voleb. Lze si zde například zobrazit počet ušlých kroků se zapnutým krokoměrem (pozor, při používání krokoměru je nutné, aby zařízení nepřešlo do režimu spánku!) nebo vynulování počtu těchto ušlých kroků.

Dalším tlačítkem v menu si lze nechat zobrazit seznam všech senzorů, které používané zařízení obsahuje.

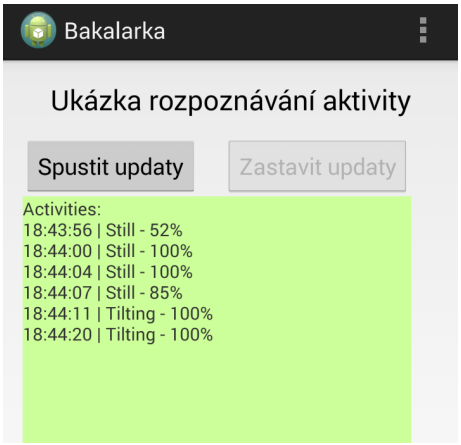
Poslední volbou lze zapnout zvuk. To se projeví tak, že po volbě senzoru

Linear Acceleration a následnému zapnutí zvuku dojde ke „zazvonění“ přístroje, pokud je jím pohnuto dostatečně velkou rychlostí. Stejnou funkci poskytuje i senzor *Accelerometer - calculated*, u kterého je však před „prudkými pohyby zařízením“ nutné několik vteřin počkat (než se data ze senzoru „stabilizují“).

A.3.3 Rozpoznávání aktivity

Tato část vypadá na první pohled velice prostě. Jsou zde pouze dvě tlačítka **Spustit updaty** a **Zastavit updaty** a zeleně podbarvená část aplikace pro výpis textu.

Prvním z tlačítek dojde k zahájení rozpoznávání aktivity. Aktivitou se zde rozumí pohyb uživatele. Rozpoznat je možné několik typů pohybu, konkrétně jízda ve vozidle, jízda na kole, chůze, nečinnost a naklápění zařízení, případně může být zobrazena nerozpoznaná aktivita. Po určité době by mělo dojít k výpisům právě rozpoznané aktivity do zeleně podbarvené části aplikace. Aktivita je akceptována s určitou pravděpodobností a ne vždy je rozpoznána naprosto správně. Pokud není zařízení schopno akceptovat rozpoznanou aktivitu, nebude jednoduše vypisováno nic. V některých případech může dojít i k akceptování špatně rozpoznané aktivity, jelikož lze uživatelem senzory jednoduše „oklamat“, pokud je zařízením pohybováno specifickým způsobem pro daný typ aktivity. Zastavení rozpoznávání aktivity je provedeno příslušným tlačítkem umístěným vpravo. Aplikaci po krátkém rozpoznávání aktivity ukazuje obrázek A.3.



Obrázek A.3: Rozpoznávání aktivity

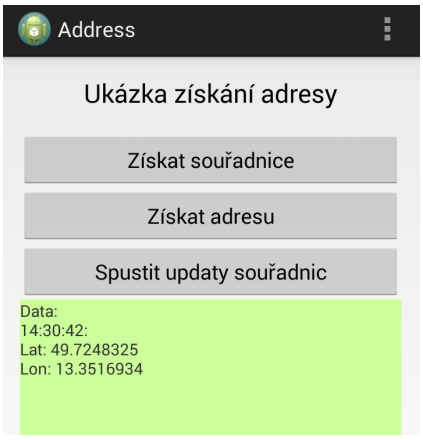
V menu aplikace je také na výběr z několika voleb. Příslušnými tlačítky si lze nechat zobrazit doposud nasbíraná data při rozpoznávání aktivity. Následně lze tato data dalším z tlačítek menu vynulovat.

Poslední volbou je zapnutí zvuku. Po zvolení této volby je vyžadováno zadání počtu sekund a případná volba, zda bude zvuk přehráván opakovaně. Tato volba má za následek to, že pokud je rozpoznána aktivita jako nečinnost a uživatel se tedy po dříve stanovený počet sekund nepohybuje, začne zařízení zvonit. A to buď opakovaně či pouze jednou, náležitě dle volby při zapnutí zvuku. To může sloužit například jako upozornění pro uživatele, že příliš dlouho sedí na jednom místě a měl by se nějakým způsobem „protáhnout“.

A.3.4 Získání souřadnic a adresy

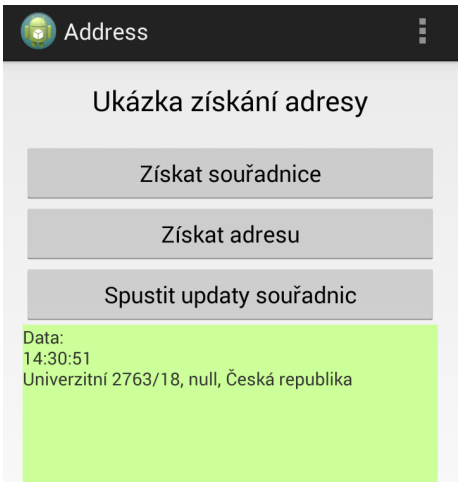
Po zvolení této části aplikace se uživateli naskytně pohled na celkem 3 tlačítka a zeleně podbarvenou část aplikace určenou pro výpis souřadnic.

Horním tlačítkem proběhne pokus o získání posledních známých souřadnic polohy zařízení. V případě, že nejsou žádné zaznamenány, je tato skutečnost vypsána na obrazovku. V tomto případě je doporučeno si například přes aplikaci Mapy (kterou by mělo zařízení nativně obsahovat) nechat určit svou aktuální polohu (k tomu bude nutné mít povoleno sledování zařízení a mít zařízení připojené k datové síti). Ta by pak již měla být načtena a v aplikaci po dalším stisku tlačítka již správně zobrazena v podobě souřadnic. Získání přesných souřadnic ukazuje obrázek A.4.



Obrázek A.4: Získání aktuálních souřadnic

Prostřední tlačítko dokáže získat (opět na základě poslední známé polohy zařízení) přesnou adresu zařízení, popsanou slovně. K tomu je však nutné být připojen k datové síti (například pomocí Wi-Fi). Získání neúplné adresy ukazuje obrázek A.5.



Obrázek A.5: Získání neúplné adresy

Poslední z tlačítek spustí pravidelné aktualizace souřadnic zařízení v určitém intervalu (v této aplikaci nastavených na 10 sekund). U této operace je vyžadováno mít povolené sledování zařízení, což lze dohledat v **Nastavení** zařízení pod položkou **Poloha**. Opětovným stisknutím tohoto tlačítka budou pravidelné aktualizace zastaveny.

V menu se pak nachází jediná položka, kterou je vynulování posledních zaznamenaných souřadnic a adresy.

B Programátorská příručka

Tato dokumentace slouží pro programátory, kteří chtěji vytvořenou *knihovnu* používat či dále rozvíjet.

B.1 Případy užití a seznam tříd

Obrázky 5.1 (strana 19) a 6.2 (strana 23) ukazují, jakým způsobem je *knihovna* v rámci aplikace využívána a které *třídy* mezi sebou navzájem komunikují. Druhý jmenovaný obrázek (6.2) zobrazuje přehled *tříd knihovny* a stručný výpis jejich nejdůležitějších *metod*. Pro podrobné informace je doporučeno nahlédnout do přiložené dokumentace vygenerované pomocí utility *Javadoc*. V té se nachází komentovaný popis každé *třídy* a jejích *metod*. Samotný kód je taktéž bohatě komentován, neměl by tak být problém se v něm snadno zorientovat.

B.2 Použití knihovny

Pro využívání *knihovny* ve vlastním projektu je nejprve nutné tuto *knihovnu* do daného projektu připojit. Poté už stačí jednoduše vytvořit instance potřebných *tříd* z *knihovny* a pomocí těchto instancí volat jejich *metody*.

Pro případ jakýchkoliv nejasností byla zhotovena testovací aplikace, která využívá všech funkcí navržené *knihovny*. Ta je, jako *knihovna* samotná, bohatě komentována a mělo by z ní být hned patrné, jak potřebné funkce *knihovny* využívat.