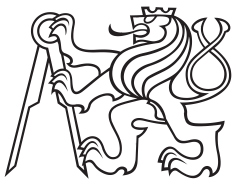
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra manažerství a humanitních věd

Využiti aspektově orientovaného přístupu a lokálních dat ve vývoji adaptivního uživatelského rozhraní pro Android

Tamara Titova

Květen 2016
Vedoucí práce: Ing. Jiří Šebek

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra ekonomiky, manažerství a humanitních věd

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Titova Tamara**

Studijní program: Softwarové technologie a management
Obor: Manažerská informatika

Název tématu:

Využiti aspektově orientovaného přístupu a lokálních dat ve vývoji adaptivního uživatelského rozhraní pro Android

Pokyny pro vypracování:
1. Prostudujte existující nástroje pro podporu aspektově orientovaného programování (AOP).
2. Seznamte se s vývojem aplikací pro mobilní zařízení s platformou Android.
3. Zpracujte rešerši ohledně vývoje aplikací, které využívají hardware mobilního zařízení (akcelerometr, senzor okolního osvětlení, fotoaparát a jiné).
4. Navrhněte a implementujte framework, který využije tyto informace při generování adaptivního uživatelského rozhraní (AUI) s ohledem na AOP.
5. Výsledná implementace musí být snadno rozšiřitelná. Implementaci otestujte na demonstrační aplikaci.
6. Zhodnoťte výhody a možná omezení řešení.

Seznam odborné literatury:
1. ŠEBEK, J. and K. RICHTA. Aspect-oriented User Interface Design for Android Applications [online]. In: DATESO 2015. Databases, Texts, Specifications, and Objects 2015, Nepřívěc u Sobotky, Jičín, 2015-04-14/2015-04-16. Praha: MATFYZPRESS, vydavatelství Matematicko-fyzikální fakulty UK, 2015, pp. 121-130. CEUR Workshop Proceedings. vol. 1343. Available from: http://www.cs.vsb.cz/dateso/2015/
2. Oficiální dokumentace: http://developer.android.com
3. http://mobilenet.cz/clanky/techbox-vas-telefon-je-prospikovany-senzory-12496
4. ŠEBEK, J., M. TRNKA, and T. ČERNÝ. On Aspect-Oriented Programming in Adaptive User Interfaces. In: Proceedings of the 2nd International Conference on Information Science and Security. The 2nd International Conference on Information Science and Security, Seoul, 2015-12-14/2015-12-16. Piscataway: IEEE, 2015, pp. 147-151.

Vedoucí bakalářské práce: Ing. Jiří Šebek

Platnost zadání: do konce letního semestru 2016/2017

L.S.

Prof.Ing. Jaroslav Knápek, CSc. — vedoucí katedry

Prof.Ing. Pavel Ripka, CSc. — děkan

V Praze dne 10.2.2016

Poděkování / Prohlášení

Chtěla bych poděkovat Ing. Jiřímu Šebkovi za vedení mé bakalářské práce, cenné rady, ochotu a vstřícný přístup. Velké poděkování náleží celé mé rodině a manželovi za podporu a trpělivost.

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 27. 05. 2016

.....

Abstrakt / Abstract

Tato bakalářská práce se zabývá problematikou vývoje softwaru závislém na kontextu a jeho využití k možnosti adaptivity uživatelského rozhraní na základě získaného kontextu. Výsledkem práce je framework, který slouží k vývoji adaptivního uživatelského rozhraní s využitím kontextu z hardwaru mobilního zařízení pro OS Android a dodržuje styl aspektově orientovaného programování. Framework je založen na principu modulární architektury. Jeden z hlavních modulů poskytuje programátorovi snadný a rychlý přístup ke kontextu senzorů mobilního zařízení. Další důležitý modul zajišťuje přístup k aspektům, založeným na specifické formě statických metadat - anotací. Aspekty jsou pomocným nástrojem k automatickému generování adaptivního rozhraní v době runtimu. Navržené postupy frameworku byly porovnány s konvenčním vývojem aplikací pro OS Android. Otestováním každého z navržených aspektů bylo prokázáno, že framework poskytuje čitelný kód a výrazně snižuje počet řádků kódu, což pomáhá vývojáři urychlit vytvoření aplikace.

Klíčová slova: kontext, aspektově orientované programování, Android, senzory, adaptivní uživatelské rozhraní

This thesis deals with design issues for software dependent on context and possibilities of adaptation of a user interface based on the obtained context. As a result of this work, a framework has been implemented to be used to develop an adaptive user interface. The framework uses the context of mobile device hardware for Android OS and follows the style of the aspect-oriented programming. The framework is based on a modular architecture. One of the main modules provides the developer with easy and quick access to the context of mobile device sensors. Another important module gives access to aspects, which are based on the particular form of static metadata called annotations. The aspects are useful tools for automatic generation of adaptive interfaces during runtime. The proposed framework procedures were compared with the conventional approach of developing applications for Android OS. By testing each of the implemented aspects it has been demonstrated that the framework provides a readable code and reduces the number of lines of the code, which helps to a developer to create application rapidly.

Keywords: context, aspect-oriented programming, Android, sensors, adaptive user interface

Title translation: Utilization aspect oriented approach and local data in the development of adaptive user interface for Android

Obsah /

1 Úvod 1
1.1 Motivace 1
1.2 Cíle 2
2 Rešerše 3
2.1 Uživatelské rozhraní 3
2.2 Adaptivní uživatelské rozhraní 3
2.3 Kontext 4
2.4 Senzory Androidu 5
2.4.1 Akcelerometr(Gravitační senzor) 6
2.4.2 Barometr 7
2.4.3 Gyroskop 7
2.4.4 Senzor okolního osvětlení 8
2.4.5 Magnetometr 9
2.4.6 Geolokační senzor GPS 10
2.4.7 Ostatní senzory 10
2.5 Aspektově orientované programování AOP 11
2.5.1 Základní pojetí AOP[1] 12
2.5.2 Výhody AOP 12
2.6 Životní cyklus aktivit v Android aplikacích 13
2.6.1 Aktivita 13
3 Související práce 15
3.1 LILOLE framework 15
3.2 Aspect Faces Java EE 15
3.3 Framework pro Android 16
3.4 AOP nastroje 16
3.4.1 AspectJ 16
3.4.2 Spring AOP 17
4 Analýza a design frameworku 18
4.1 Package diagram 18
4.2 Use Case Diagram 20
4.3 Sekvenční diagram 21
5 Implementace 23
5.1 Struktura projektu Android Studio 23
5.2 Rozhraní XML vs. Rozhraní JAVA 25
5.3 Aspekty 26
5.4 USE-CASE: @Country a @Address 27
5.5 USE-CASE: @AutoColor a @BackgroundAutoColor 29
5.6 Průřezové body aspektů @AutoColor a @BackgroundAutoColor 30
5.7 Use-Case @TextSize 31
5.8 Use-Case @ShakeSound 32
5.9 Use-Case @Brightness 33
5.10 Přístup k senzorům 34
6 Porovnání 36
6.1 Framework vs Konvenční přístup: Přístup k senzorům 36
6.2 Framework vs Konvenční přístup: Posluchač senzorů 37
6.3 Framework vs. Konvenční přístup: @Country 37
6.4 Framework vs. Konvenční přístup: @Address 38
6.5 Framework vs Konvenční přístup: @AutoColor 38
6.6 Framework vs. Konvenční přístup: @BackgroundAutoColor 39
6.7 Framework vs. Konvenční přístup: @TextSize 39
6.8 Framework vs. Konvenční přístup: @ShakeSound 40
6.9 Framework vs Konvenční přístup: Úsudek 40
7 Testování 41
7.1 Testování času spuštění aplikace 41
7.2 Tabulka regresivních testů 43
8 Instalace 44
8.1 Operační systém a Vývojové prostředí 44
8.2 Postup instalace 44
9 Závěr 45
Literatura 46
A Zadání bakalářské práce 49
B Příklady kódu 50
C Obsah přiloženého CD 51
D Použité zkratky 53

Tabulky / Obrázky

5.1. Empirické nalezení hodnoty zrychlení 33
6.1. Framework vs Konvenční přístup: Přistup k senzorům 36
6.2. Framework vs Konvenční přístup: Posluchač senzorů..... 37
6.3. Framework vs Konvenční přístup: @Country 37
6.4. Framework vs Konvenční přístup: @Address..... 38
6.5. Framework vs Konvenční přístup: @AutoColor 38
6.6. Framework vs. Konvenční přístup: @BackgraoundAutoColor 39
6.7. Framework vs. Konvenční přístup: @TextSize 39
6.8. Framework vs. Konvenční přístup: @ShakeSound 40
7.1. Testování času spuštění aplikace..... 41
7.2. Tabulka regresivních testů..... 43

2.1. Struktura čipu akcelerometru ...6
2.2. Intenzita osvětlení.....8
2.3. Globální polohovací systém.... 10
2.4. AOP Integrátor 11
2.5. Životní cyklus aktivity 14
4.1. Package Diagram..... 19
4.2. Use Case Diagram 20
4.3. Sequence Diagram 21
5.1. Struktura projektu v Android Studio. 23
5.2. Příklad rozhraní XML s použitím res/layoutu 25
5.3. Příklad na vytváření anotace .. 26
5.4. Anotace @Brightness 26
5.5. Zobrazení státu na obrazovce.. 28
5.6. Experiment nalezení adresy. ... 28
5.7. Struktura anotace @AutoColor..... 29
5.8. Příklad obrazovky při využití @AutoColor 30
5.9. Příklad obrazovky při využití @TextSize 32
5.10. Struktura @ShakeSound. 32
5.11. Příklad změny jasu obrazovky..... 34
5.12. Přístup k senzorům využitím SensorManageru..... 34
5.13. Příklad na získání kontextu gyroskopu..... 34
5.14. FactoryContext příklad 35
5.15. Zobrazení kontextu senzorů na obrazovce..... 35
8.1. Import projektu 44

Kapitola 1
Úvod

V současné době se počet lidí, kteří nevlastní smartfon, stále snižuje. Většina lidí nosí téměř každý den telefon v kapse: při cestě do zaměstnání, na dovolené, v obchodě, doma - všude jsou v blízkosti telefonu, a tato skutečnost může být použita v zájmu vývojáře. Proces vývoje zaměřený na uživatele poskytuje určitou míru souladu s jeho požadavky.

Adaptivní druh myšlení vývojáře pomáhá najít řešení, které může výrazně zlepšit interakci uživatele s okolním prostředím pomocí adaptivního rozhraní. Obdržením dat ze senzorů telefonu vývojář získává možnost analyzovat činnost uživatele, předvídat jeho budoucí chování a na základě těchto informací vytvářet adaptivní rozhraní, které bude přizpůsobeno individuálním potřebám uživatele.

1.1 Motivace

Kontext je základem lidské komunikace - v životě obvykle stačí říci několik slov, aby byl sdělen smysl zprávy. Za účelem rychlého pochopení zpráv člověk podvědomě načítá kontext prostředí, uloží jej a používá na "dešifrování" přicházejících zpráv od jiných osob. Tedy jde o kontext dějin, kontext vlastních zkušeností nějaké osoby, kontext, který se dá získat pomocí smyslových orgánů. Nicméně, přístroje nejsou lidé, ale mají také své „smyslové orgány", které zajišťují jejich schopnost vnímání a pamatování si kontextu. Je nutné využívat všechny možnosti zařízení, protože to zvýší úroveň komunikace s uživatelem. Velkým přínosem je schopnost zařízení přizpůsobovat se potřebám uživatele. Používání takového "chytrého" zařízeni je pohodlnější a příjemnější.

Adaptivní rozhraní je jedním z možných nástrojů adaptivity. K rozpracování adaptivního rozhraní se vyžaduje hluboké porozumění kontextu, které je možné získat například prostřednictvím senzorů mobilního zařízení. Hlavní výhodou adaptivity je schopnost odstranit zbytečné informace pro uživatele na základě přijatých kontextových charakteristik. Rozhraní se upraví takovým způsobem, aby uživatel nemusel provádět nadbytečnou práci a neztrácel čas na věci, které zařízení může provést samostatně.

Hlavní motivací této práce je:

- prozkoumat možnosti, které poskytuje kontext senzorů
- navrhnout framework, zaměřený na získaní kontextu z hardwaru mobilního zařízeni pro OS Android a využit princip aspektově orientovaného programování
- jako součásti frameworku navrhnout moduly na vytvářeni adaptivního rozhraní, běžícího na základě kontextově-závisle aplikace

V současné době existuje pouze jeden framework pro Android [2], který je zaměřen na vytvářeni adaptivního rozhrání na principu AOP. Tento framework je inspiraci pro napsání této práce. Framework [2] pro Android nevyužívá kontext ze senzorů, proto tento obor vývoje poskytuje hodně možnosti na hledaní a nalézání zajímavých řešení.

1.2 Cíle

Cílem této práce je nastudovat existující nástroje pro podporu aspektově orientovaného programování, seznámit se s vývojem aplikací pro mobilní zařízení s platformou Android a nastudovat informace o kontextu, kontextově-závislých aplikacích. Využít získané informace pro napsání vlastního frameworku zaměřeného na generování adaptivního uživatelského rozhraní(AUI) s ohledem na AOP. Dalším cílem je navrhnout framework, který bude efektivní a příznivý pro budoucího vývojáře.

Aby framework byl vhodný pro použití, musí splňovat následující vlastností:

- zajišťuje jednotný styl psaní kódu
- zajišťuje strukturovanou architekturu
- obsahuje čitelný kód
- snižuje duplicitu kódu
- zajišťuje adaptivitu na základě kontextu senzorů
- zajišťuje snadnou rozšiřitelnost
- poskytuje několik příkladů pro prezentace
- poskytuje co nejvíce hotových řešení pro generování UI

Kapitola 2
Rešerše

2.1 Uživatelské rozhraní

Uživatelské rozhraní je nástroj na komunikaci mezi zařízením a uživatelem. Účelem rozhraní je zjednodušení interakce uživatele s různými funkcemi programu, skrytí zbytečné informace uživateli, umožnění přehledné a pochopitelné práce s programem.

Uživatelské rozhraní je rozděleno do dvou typů[3]:

- **Statické**(fixní) - rozhraní, které zachovává neměnné, stálé schéma chování ve vztahu k uživateli. Neměnícím se schématem je například inicializace a ověřování uživatelů, informace ukončení seance („shut down“), informace „O programu“, - tyto operace nemohou být upravené ani libovolnou skupinou uživatelů, ani jednotlivě.

- **Adaptivní**(dynamické) - rozhraní , které má celou řadu scénářů chování vzhledem k uživateli. Adaptivní část je založena na neměnné části, která dává podporu při dynamické práci. Tento druh rozhraní působí na uživatele a dynamicky se přizpůsobuje uživatelskému modelu chování. Adaptivní součástí rozhraní je vše, co je spojeno s vytvořením „modelu uživatele“, nástrojem funkčních, podporujících a designérských aspektů interakce.

2.2 Adaptivní uživatelské rozhraní

Adaptivní uživatelské rozhraní je automaticky přizpůsobitelné rozhraní, které mění svoje chování v závislosti na jednotlivých uživatelských úkolech. Adaptivní uživatelské rozhraní je flexibilní systém, zahrnující soubor programových a technických prostředků, který provádí logické uspořádání dat o koncovém uživateli, předpovídá jeho chování a přizpůsobuje se jeho preferencím a zájmům.

Úkoly adaptací systému k uživateli:

- detekce a korekce chyb, kterých se dopustil uživatel při práci se systémem
- změna složitosti rozhraní v souladu se získanými daty o uživateli
- přizpůsobení rozhraní cílům uživatele
- výběr optimálního způsobu prezentace zdrojů informací pro uživatele
- integrace poskytnutých informací pro uživatele

2.3 Kontext

Existuje interdisciplinární věda „Teorie interakce člověka a počítače“ (angl. *Human Computer Interaction*) nebo zkráceně - **HCI**. HCI[4] je věda, která se zabývá zkoumáním metod a způsobů interakce mezi uživateli a počítači. Tato věda je souhrnem různorodých oblastí, jako jsou: informatika, programování, inženýrství, psychologie, design a další. Kruh odborníků podílejících se na různých aspektech interakce člověk-počítač je dostatečně velký, proto každým rokem věda Human Computer Interaction přiláká více a více odborníků z mnoha dalších oborů, přijímá nové koncepce a postupy návrhu.

Všudypřítomná výpočetní technika, jinak *Ubiquitous computing*(**ubicomp**)[5] (nebo *pervasive computing*) je model, který popisuje pronikání počítačů do různých sfér lidské činnosti. Jeden z modelů HCI je osobní počítač PC. Ubicomp je možné si představovat jako následující model HCI po PC. Hlavní důležitou odlišností modelu Ubiquitous computing je přítomnost množství subjektů a objektů, které jsou sjednoceny do jednoho společného systému pro efektivní práci s informací. Ubicomp označuje éru chytrých strojových zařízení vstupujících do lidského prostředí.

Strojní zařízení musejí umět samostatně získat informace o okolním prostředí, ve kterém se nachází uživatel, „p“ocítit jeho zájmy a předpovědět úmysly tj. musí mít schopnost rozpoznat potřebný kontext a správně tento kontext interpretovat. Je možné vytvářet zařízení a systémy, které budou brát v úvahu kumulativní znalosti a zkušenosti. Tímto způsobem je možné získat inteligentní systémy, které se dokážou přizpůsobit předem nenaprogramovaným situacím, vnímat změny v okolním prostředí a komunikovat s uživateli.

Kontext je velice rozsáhlý pojem, který nemá přesně specifikovanou definici. V hlavních rysech jsou kontext data, popisující prostředí nebo okolí, v němž se pohybuje uživatel.

Day a Abowd [6] zadefinovali kontext mobilního zařízení a všudypřítomné výpočetní techniky jako:

„Kontext je informace, která může být použita k charakterizaci situace, ve které se nachází osoba, místo nebo objekt, který je považován za relativní pro interakce mezi uživatelem a aplikací, včetně uživatele a aplikace samostatně.“.

Zde je uveden názorný příklad ze života pro vysvětlení pojmu „kontext“: Představme si, že chceme jít na procházku. Podíváme se z okna, jaké je počasí. Vidíme šedé nebe a slyšíme, jak fouká vítr. Teploměr ukazuje +15 stupňů Celsia. Dále předpokládáme, že bude pršet a podle našeho úsudku si s sebou vezmeme deštník.

Co nám příklad ukazuje? Shromáždili jsme různé údaje, které popisovaly vnější prostředí za oknem(kontext) a následně jsme vytvořili předpověď budoucího počasí. Na základě předpovědi jsme nastavili určité chování a přizpůsobili jsme se naší prognóze(čili vzali jsme si deštník). Stejně by mohl smartfon nabídnout uživateli v souladu s nasbíraným kontextem pomocí senzorů, aby si s sebou vzal deštník - tímto způsobem by komunikoval s uživatelem a vypadal chytřejší.

Většinou uživatel pracuje s jakýmkoliv zařízením pomocí rozhraní. Rozhraní musí být vytvořeno na základě takového systému, který nebude pro uživatele zaplněn zbytečnými daty a funkcemi. Proto je vývoj adaptivního rozhraní velmi důležitým procesem.

Mobilní zařízení je schopno přijímat maximální množství údajů o svém vlastníkovi, analyzovat je v reálném čase a přizpůsobovat se jeho chování a požadavkům. Tímto způsobem se uživatel nemusí orientovat ve funkcích rozhraní, o které nemá zájem. Adaptivní rozhraní samo o sobě přijme vzhled, vhodný pro jeho uživatele.

Existuje zvláštní přístup k rozvoji různých typů informačních systémů, který bere v úvahu všechny vlastnosti okolního prostředí a je určen pro kompenzaci nedostatků výpočetní techniky, které nemohou hromadit znalosti o kontextu. Tento přístup se nazývá *Context-Aware Computing*(**CAC**)[5]. V roli kontextově-závislého přístupu ve sféře mobilních zařízení vystupuje adaptivita, která zprostředkuje informaci o aktuálním kontextu, sloužící k zajištění vhodné interakce mezi uživatelem a vizuálně představenou informací vzhledem k současné situaci.

Kontextově-závislé systémy patří ke kategorii *Ubiquitous computing*. Hlavním zdrojem informací pro kontextově-závislé systémy jsou:

- umístění, poloha uživatele
- sociální prostředí(dav, rodina, přátelé, kolegové), kdo je v blízkosti uživatele
- technologické prostředí(počítače, telefony, tablety, servery atd., přístup k Internetu)
- fyzické prostředí (počasí, tlak, úroveň osvětlení, hluk, úroveň šumu)

Moderní mobilní zařízení mají širokou škálu komunikačních schopností(bezdrátové počítačové sítě, mobilní telefonie) a „smyslové orgány“, tedy senzory. V závislosti na funkčních možnostech poskytovaných smartfonem, mohou být použity tyto senzory: akcelerometr, barometr, senzor okolního osvětlení, magnetometr, teploměr, senzor měření šumové hladiny, senzor měření kvality mobilní komunikace a jiné. Univerzální zařízení(mobilní telefony) poskytují hodně možností sbírání kontextu nutného pro optimalizaci HCI systému.

Bez ohledu na to, že nyní existuje mnoho hotových řešení chytrých telefonů, schopnosti mobilních zařízení v stylu CAC disponují unikátními a podivuhodnými vlastnostmi, které poskytují široké pole působnosti pro rozpracování „inteligentních“ mobilních zařízení vývojáři.

2.4 Senzory Androidu

Android umožňuje získat přístup k senzorům pomocí knihovny android.hardware. Platforma Android podporuje senzory následujících tří kategorií[7]:

- **Senzory pohybu (Motion sensors)**

Tyto senzory měří síly zrychlení a rotační síly kolem tří os. Do této kategorie patří akcelerometry, gravitační senzory, gyroskopy a rotační vektorové senzory.

- **Senzory okolního prostředí (Environmental sensors)**

Tyto senzory měří různé parametry prostředí, jako je teplota, tlak, osvětlení a vlhkost. Do této kategorie patří barometry, fotometry a teploměry.

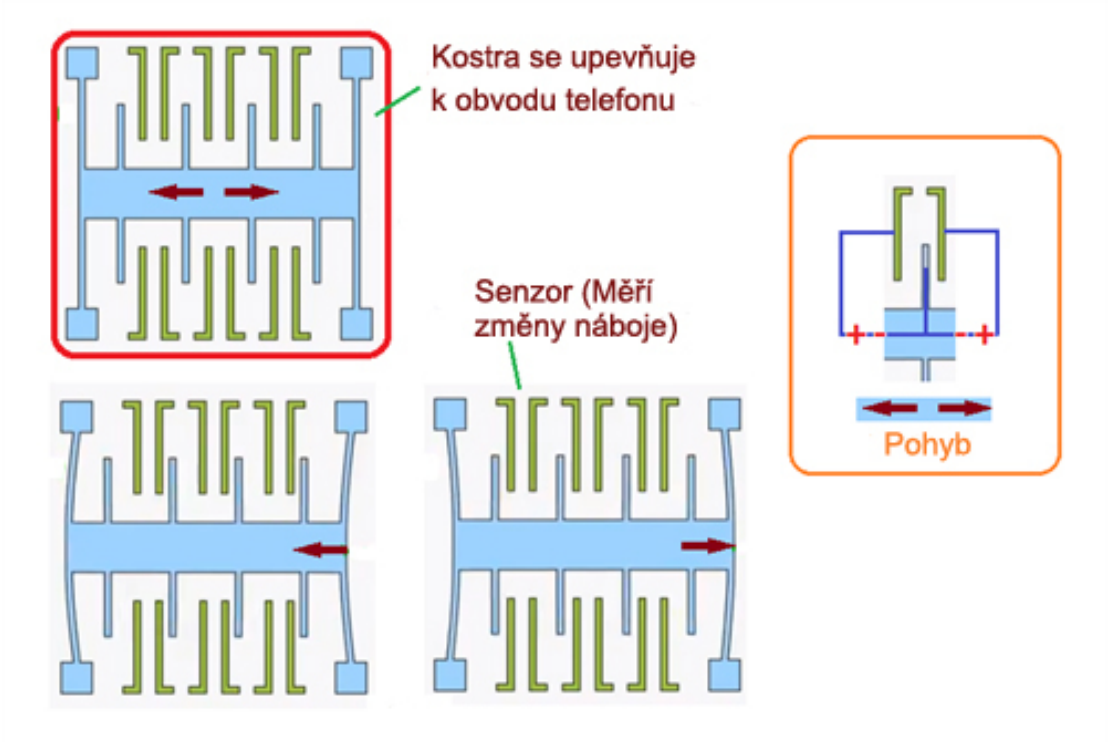
- **Senzory polohy (Position sensors)**

Tyto senzory měří fyzickou pozici zařízení. Do této kategorie patří orientační senzory a magnetometr.

2.4.1 Akcelerometr(Gravitační senzor)

Akcelerometr je snímač, sloužící k měření síly zrychlení v měřicích jednotkách $[m/s^2]$. V Androidu se aplikuje na zařízení ve všech třech fyzikálních osách(x, y, z) včetně gravitační síly.

Veškeré prvky akcelerometru se nachází v jednom čipu. Schéma vypadá následujícím způsobem:



Obrázek 2.1. Struktura čipu akcelerometru. Převzato z [8].

Akcelerometr [9] je zabudován do obvodů telefonu. K pevné kostře s pružnými držáky, které umožňují pohyb v rámcích určité meze, se připevňuje přepážka z vodičů. Tyto odvody jsou umístěny mezi kontakty, které dovolují získávat naměřená data. Během pohybu odvodů, napětí pole kolem kontaktů mění svoje vlastnosti, a to umožňuje provádět měření.

Akcelerometry v mobilních telefonech fungují na principu schopnosti krystalu generovat elektrické napětí při jeho deformování. Vytvořený elektrický náboj je tedy úměrný síle, která ho vygenerovala. Elektrický náboj je přes speciální tranzistor konvertován na napěťový výstup s nízkou impedancí, která je dobře měřitelná běžnými měřicími přístroji.

Původně se akcelerometr v telefonu využíval k měření množství kroků(krokoměr), tato funkce byla velmi užitečná pro ty, kteří mají rádi sportovní životní styl. Možnosti využití akcelerometrů v smartfonu se poměrně zvyšovaly a počet aplikací využívajících akcelerometr narůstal. Později byl akcelerometr použit jako vodováha a také k vykonání pohybu ve hrách.

Akcelerometr se také používá k ochraně hardwaru mobilního zařízení při pádu. V případě pádu telefonu, akcelerometr okamžitě posílá příkaz vnitřním důležitým zařízením zaujmout bezpečnou polohu. To pomáhá snížit míru poškození vnitřních detailů telefonu a díky tomu se snižuje procento ztracených a poškozených dat.

Aplikace na Androidu, využívající akcelometr[10]:

- „Accelerometr Monitor“ - je aplikace pro měření vibrací v reálném čase. Poskytuje možnost nalezení zemětřesení.
- Aplikace, kde se stává nějaká akce při zatřepání telefonu, skoro všude je možné využit úroveň třesení:
  - „AppShaker“ (otevirá se předem vybraný program)
  - „ShakeCall“ (přijetí a zakončení volání)
  - „Shaker Unlocker“ (odblokování klávesnici)
  - „ShakeSMS“ (otevřít přijatou sms)
- „Tilt Scroll“ - je aplikace, která poskytuje možnost provádět scroll obrazovky pomocí náklonu telefonu na určitou stranu. Náklon dolů - text se spustí dolů.

2.4.2 Barometr

Barometr je přístroj na měření atmosférického tlaku. Uvnitř telefonu se instaluje digitální barometr, který využívá elektronické náboje na měření tlaku. Elektronický snímač pracuje na principu změny odporu v době deformace tenzometrického převodníku, který je připevněn na pružném elementu. Tento element se deformuje pod vlivem tlaku. Potom se ve snímači změní tlak na napětí.

Barometr se používá na předpověď počasí, přibližujících se cyklonů a anticyklonů, měření nadmořské výšky. Barometr slouží jako doplněk k GPS navigaci, poskytuje informace při vypočítávání souřadnic výšky a délky, což pomáhá zařízení rychleji stanovit polohu. V současné době ještě neexistuje hodně modelů smartfonů, využívajících barometr.

Aplikace na Androidu, využívající barometr[10]:

- „SyPressure“
- „vBarometer“
- „Barometer Monitor“

2.4.3 Gyroskop

Gyroskop [11] je zařízení, které se používá ke stanovení orientace zařízení v prostoru. Tento snímač je schopen reagovat na změny úhlů orientace těla ve třech fyzikálních osách x,y a z, což slouží ke sledování pohybu zařízení. Měřítkem je rad/s.

Bez ohledu na to, že funkčnosti akcelerometru a gyroskopu jsou podobné, princip jejich práce je odlišný. Gyroskop vypočítává úhel naklonění vzhledem k zemi, zachycuje polohu těla v prostoru vzhledem k vlastní „gravitaci“. Akcelerometr pak počítá vlastní zrychlení spolu s použitím gravitačního zrychlení planety. V praxi funkčnosti těchto dvou zařízení mohou být navzájem nahrazované, anebo doplňované. Většinou smartfony mají jak akcelerometr, tak i gyroskop.

Kombinace těchto dvou senzorů umožňuje sledovat pohyb v trojrozměrném prostoru a zvyšuje citlivost zařízení k jakémukoli naklonění, natáčení a dalším pohybům.

Jejich hlavním úkolem je zvýšení kvality her. Hráč může ovládat hry pomocí naklonění, setřásání, natáčení a dalších možností.

Existuje další možnost použití gyroskopu. Aplikuje se pro nalezení a určení směru pohybu při použití GPS navigace. Mapa bude zobrazena pomocí gyroskopu v souladu s tím, jak uživatel stojí, tj. na obrazovce se zobrazí schéma té místnosti, na kterou se

dívá uživatel v určitém čase. Zobrazení mapy na obrazovce téměř vždy odpovídá směru pohledu uživatele.

Existuje také další možnost použití gyroskopu. Aplikuje se pro nalezení a určení směru pohybu při použití GPS navigace. Mapa bude zobrazena pomocí gyroskopu v souladu s tím, jak uživatel stojí, tj. na obrazovce se zobrazí schéma místnosti, na kterou se uživatel v určitém čase dívá. Zobrazení mapy na obrazovce téměř vždy odpovídá směru pohledu uživatele.

Aplikace na Androidu, využívající gyroskop[10]:

- „Gyrophone“
- „3D Gyro Compass“
- „Planet Finder“ - je aplikace, poskytující rychlou orientaci viditelných planet na noční obloze. Je to astronomický kompas, který zobrazuje polohu planet, Slunce, Měsíce a Pluta, má 3D zobrazení Merkuru, Venuše, Měsíce, Marsu, Uranu, Neptunu, Pluta, většiny satelitů sluneční soustavy. Tato aplikace používá nejen gyroskop, ale také akcelerometr a magnetometr.

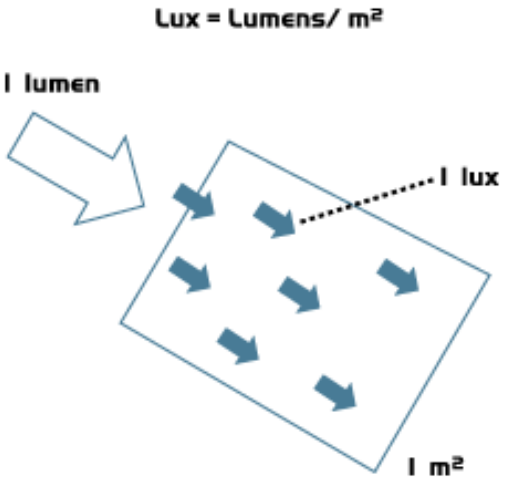
2.4.4 Senzor okolního osvětlení

Senzor okolního osvětlení se používá v systému na získání úrovně vnějšího osvětlení prostředí. Zařízení interpretuje informace ze senzoru pomocí speciálního programového zabezpečení a používá je na další automatické nástroje jasu podsvícení displeje a nástroje teploty barev obrazu.

Například, senzor okolního osvětlení adaptuje obrazovku na lepší viditelnost, když svítí slunce nebo naopak přizpůsobí jas a barvy obrazovky v temnotě. Používá se na vypínání podsvícení klávesnice v denním světle.

Díky automatickému ladění pomocí senzoru, vzniká úspora energie. Při vyhodnocení, že se uživatel nachází v temnějším prostředí, než tomu bylo dříve, se automaticky sníží jas obrazovky. Tím se dosáhne také snížení příkonu a zároveň se tím optimalizuje spotřeba energie.

Intenzita osvětlení[12] je množství světelné energie, dopadající na jednotku plochy, nazývané světelný tok (tj. Lumen) na metr čtvereční. Osvětlení se měří v luxech a měření bude obsahovat všechna světla dopadající na bod měření od 180 stupňů polokoule.



Obrázek 2.2. Intenzita osvětlení. Převzato z [12].

Neexistuje přesné definování pojmů „světlého“ a „tmavého“, ale existují zdroje, které popisují kolik světla je obvykle ve standardním pokoji, na ulici při svítícím slunci nebo měsíci. Z toho se dá přibližně získat představa, kolik luxů je „světlé“ a kolik je „tmavé“.

Venkovní intenzita světla[13] je přibližně 10000 luxů za jasného dne. V budově v oblasti nejblíže k oknům může být úroveň osvětlení snížena na přibližně 1000 luxů. Ve standartní místnosti může být intenzita docela nízká: kolem 25 až 50 luxů. V současné době je běžně intenzita světla v rozmezí 500 - 1000 luxů - v závislosti na aktivitě.

Aplikace na Androidu, využívající senzor okolního osvětlení[10]:

Aplikace slouží pro měření osvětlení v luxech, kandelách. Obsahují nástroje na měření hodnot osvětlení.

- „Lighting calculations“ - umožňuje měření a převádění mezi různými jednotkami: lumen - lux, lumen - watt, lumen - kandela. Měří barevnou teplotu, může zjistit typ lampy.

  Aplikace na měření dopadajícího světla zastupuje funkce expozimetru, což je užitečné pro fotografy. To pomůže nastavit expozici (clonu, čas osvětlení a citlivost (ISO)).
- „beeCam Light Meter“ - tato aplikace může měřit expozici bez vlivu odrazivosti objektu. Tato aplikace může také vypočítat f-stop nebo rychlost závěrky pomocí ručně přivedeného osvětlení nebo EV(Exposure Value).
- „Orchid Care“ - tato aplikace ukazuje má-li orchidej dostatek světla a kolik světla potřebují různé druhy orchidejí.

2.4.5 Magnetometr

Magnetometr - senzor, který dovoluje měřit sílu magnetického pole podél os x, y a z, stejně jako magnetické vlastnosti materiálů.

Magnetometry používají astronomové k měření dopadu slunečního větru na magnetické pole Země, ke zkoumání fenoménu polární záře. Armáda je používá k hledání ponorek. Archeologové používají magnetometry k vyhledávání trosek potopených lodí nebo stop starověkých civilizací. Pomocí magnetometru je možné najít minerál obsahující železo.

Mobilní telefon má velmi malý magnetometr, který nestačí na vojenské nebo archeologické účely, ale magnetometr v systému Android je velmi užitečný pro implementaci funkcí kompasu. Takže magnetometr v telefonu může být použit i jako hledač kovů.

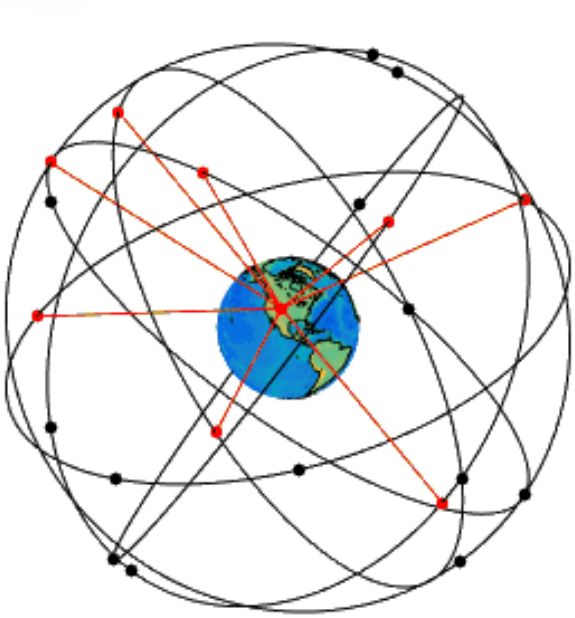
Aplikace na Androidu, využívající magnetometr[10]:

- „Gauss Meter“ - aplikace na měření hustoty magnetického toku(B) v jednotkách Gauss nebo Tesla
- „Metal Detector“ - tato aplikace může pomoci najít kovy
- „Compass 4D“- aplikace, která představuje panoramatický kompas, ukazuje postavení planet, zobrazí nejbližší města bez připojení na internet. Tato aplikace používá magnetometr, gyroskop a akcelometr.

2.4.6 Geolokační senzor GPS

GPS je Global Positioning System což znamená globální polohovací systém. Tento systém zajišťuje nalezení polohy zařízením odkudkoliv. Poloha se určuje pomocí signálu vysílaného z družic, které obíhají kolem Země. Družice komunikují jak mezi sebou, tak i s pozemní službou, proto vždy vědí své souřadnice v prostoru a čase.

Jednotlivá družice vysílá signál, obsahující čas odeslání signálu a své souřadnice. Přijímač ze signálu detekuje kód družice a z rozdílu času mezi odesláním a přijetím kódu a známé rychlosti, vypočítá vzdálenost od družice. Provedením měření u několika družic současně vysílajících signál, se jejich dráhy protnou a takovým způsobem přijímač zařízení získá polohu.



Obrázek 2.3. Získání polohy pomoci signálu z družic. Převzato z [14].

GPS používají vědci a výzkumníci jako zdroj přesného času. Vnitřní hodiny přijímače jsou neustále synchronizované s hodinami, umístěnými na družicích. Tím je zajištěna přesnost měření času z mikro až do nanosekund.

GPS výrazně snižuje náklady spojené s průzkumnou prací a výrazně zkracuje dobu provádění záchranných operací.

V současné době je GPS v mobilních telefonech velmi populární. GPS používají turisté, myslivci, rybáři, cyklisté a všichni, kteří chtějí vědět, kde se nacházejí, odkud přišli, jak rychle se pohybují nebo jak se dostat k cílovému bodu.

2.4.7 Ostatní senzory

Kromě základních senzorů existují i další různé a zajímavé senzory[15]:

1. **Senzor přiblížení**(proximity sensor) je snímač, který je schopen detekovat přítomnost blízkých objektů, aniž by s nimi musel být ve fyzickém kontaktu. Například, při přiblížení telefonu k uchu zhasne displej - to má na starosti právě tento senzor.
2. **Teploměr** a **vlhkoměr** měří teplotu a vlhkost okolního vzduchu. Je možné říci, že je to malá meteostanice. Pokud má telefon teploměr a vlhkoměr, tak není potřeba používat internet na zjištění počasí.

3. **Senzor pro pohybová gesta** pomocí infračervených paprsků detekuje pohyby ruky, což umožňuje ovládat smartfon pomocí pohybových gest.
4. **RGB světelný senzor** měří podíl množství červeného, zeleného a modrého světla. Na základě naměřených hodnot upravuje barevné podání displeje, aby barvy co nejvíce odpovídaly barvám, které vidí člověk.
5. Každé mobilní zařízení má určitě **mikrofon**. Mikrofon se používá k přenosu hovoru nebo k nahrání zvukového souboru pomocí funkce diktafonu. V moderních smartfonech se často používá ještě druhý mikrofon. Ten vnímá celé zvukové spektrum, typicky v rozsahu od 100 Hz - 10000 Hz. Pomocí speciálního elektronického obvodu se hlavní signál a signál druhého mikrofonu spojují. Takovým způsobem lze nalézt šum škodící kvalitě hovoru. Šum se odečte od hlavního signálu a zvuk bude čistší.

2.5 Aspektově orientované programování AOP

Existují softwarové problémy, na jejichž řešení nestačí použít procedurální nebo objektově orientované techniky. Realizace některých konstrukčních částí kódu je rozptýlena v celém programu, což vede k obtížím při vývoji a další podpoře programu. Hlavní příčina obtíží vzniká kvůli neschopnosti přesně zapouzdřit funkčnost, která vede ke vzniku průřezové funkčnosti.

Průřezová funkčnost (angl. *scattered*, *tangled*) - funkčnost, kterou není možné oddělit jako samostatnou podstatu pomocí metod, modulů a tříd. Realizace průřezové podstaty je rozptýlena do různých softwarových modulů, což vede k obtížím při porozumění kódu.

Aspektově orientované programování AOP je programovací paradigma, která umožňuje jasně vyjádřit strukturu programu, včetně průřezových problémů. AOP je založeno na modulech, nazývaných aspekty, ve kterých se realizuje průřezová funkčnost. Realizace kódu pomocí aspektů získává potřebnou izolaci, logické složení. Kód na základě aspektů může být použit opakovaně.

Práce v rámci AOP se skládá ze tří samostatných kroků[16]:

Obrázek 2.4. AOP Integrátor

1. **Aspektová dekompozice.** V tomto kroku se definuje, k jakému druhu patří požadavky, jestli k běžnému nebo k průřezovému. V této fázi se určuje průřezová funkčnost a moduly, které budou tuto funkčnost provádět.
2. **Realizace funkčnosti.** Každý požadavek je implementován samostatně a jasně vázán na svůj modul.
3. **Složení aspektů.** V této fázi se definují pravidla pro vytváření aspektů. Výsledek určuje, jak bude vypadat cílový systém.

2.5.1 Základní pojetí AOP[1]

Join point (česky *bod připojení*) - je místo v programovacím kódu, které se používá k volání provedení činností, které jsou rozptylovány v celém programu, tz., je možné aplikovat průřezovou zodpovědnost.

Advice (česky *rada*) - způsob vyhotovení kódu, který musí být vyvolán z bodu připojení(join point). Advice rozšiřuje stávající model programu, poskytuje dodatečnou logiku. Advice může být proveden před, po, nebo namísto definovaného bodu provádění programu. Například to může být upozornění pro programátora o vzniklé chybě v době provádění nějaké metody.

Pointcut (česky *průřezový bod*) [17] - místo v programu, kde je potřeba použit průřezovou zodpovědnost. Je to množina pointcutů. Tento bod pomáhá určit, zda daný join point je vhodný pro tento advice.

Aspect (česky *aspekt*) je modul, který je výsledkem aspektové dekompozice, během tohoto procesu se zjistí nějaké události, koncepty, pojetí, které mohou být použity na skupinu komponent, získaných po dekompozici. Obecně je to modul nebo třída, který spojuje komponenty systému a realizuje průřezovou funkčnost. Aspect mění chování kódu tím, že aplikuje advice v join pointech, definovaných možným pointcutem.

Target (česky *cíl*) - objekt, na který se aplikují advices.

Introduction (česky *zavedení*) - změna struktury třídy nebo změna hierarchií dědičnosti pro přidání aspektové funkčnosti v cizorodý kód.

Weaving (česky *propletení*)) je proces propletení objektů s odpovídajícími aspekty (třeba v době kompilace, stažení nebo v době běhu programu).

2.5.2 Výhody AOP

1. **Zapouzdření funkčnosti**. Tato vlasnost se dosahuje tím, že se zlepší systém pomocí dekompozice a rozloží se do samostatných modulů. Každý požadavek je implementován samostatně s minimální vazbou. Moduly obsahují minimální množství duplicitního kódu.
2. **Jednoduchost**. Vzhledem k tomu, že každý požadavek je realizován v samostatném modulu, kód se stává jednodušším a čitelnějším. To umožní rychle vyvíjet správně fungující systém.
3. **Možnost rozšíření**. Na stávající systém je snadné přidávat nové funkce tím, že se vytvoří nové aspekty. Po přidávání nového modulu do systému, již existující aspekty začínají působit na nově přidaný modul bez dodatečného úsilí. Vždy je možné odložit řešení potenciálních požadavků a realizovat je později jako jednotlivé aspekty, což nemá vliv na stávající funkčnost.

2.6 Životní cyklus aktivit v Android aplikacích

2.6.1 Aktivita

Překlad slova „Activity“ z angličtiny do češtiny znamená činnost, akce. Aktivita je hlavní část jakékoli Android-aplikace, na kterou působí uživatel, aby dosáhl vykonání některých akcí(odeslat SMS, nahrát video, uložit nové telefonní číslo atd.). Aktivita je často prezentována uživateli jako zvětšené okno na celou obrazovku, plovoucí okna nebo také jedna aktivita může být vložena do jiné aktivity.

Hlavní je pravě aktivita, která byla spuštěna jako první. Z ní je možné spustit další aktivitu. Každá android-aplikace obsahuje alespoň jednu aktivitu. Všechny aktivity přebývají v různých stavech, které tvoří *životní cyklus aktivity*. Jednotlivá aktivita má svůj vlastní životní cyklus.

Životní cyklus aktivity má 4 stavy[7]:

1. **active** nebo **running** (běžící stav)
   Aktivitu je vidět na popředí obrazovky, uživatel může navzájem působit na aktivitu.

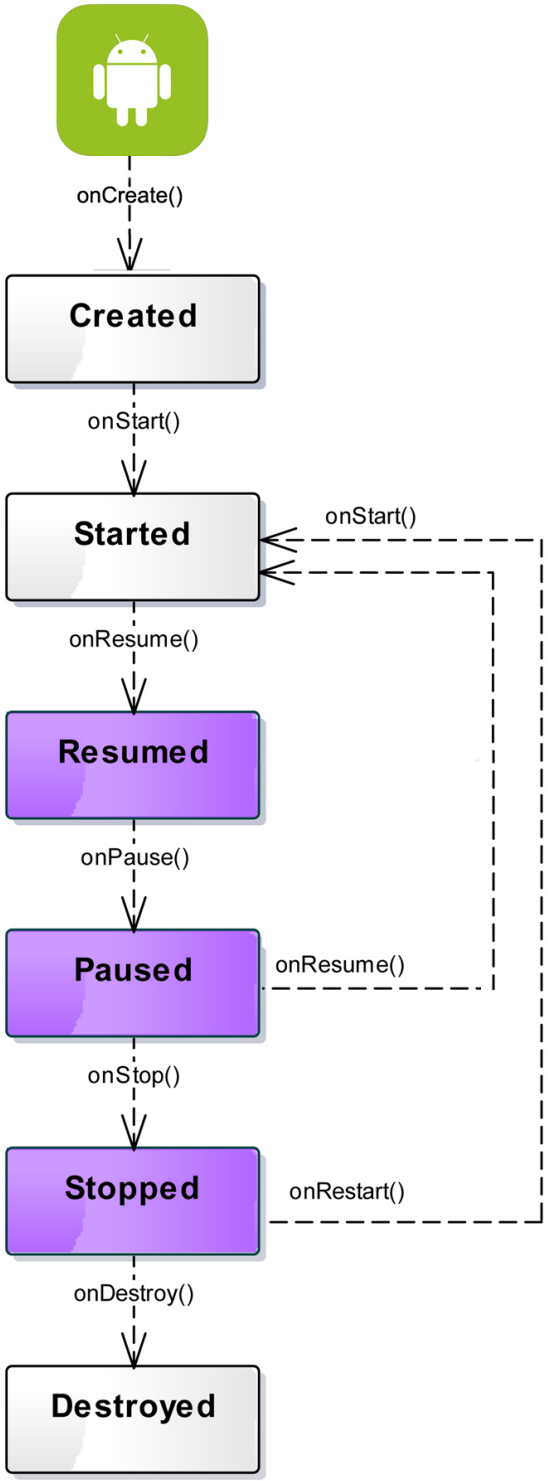
2. **paused** (přerušený stav)
   Uživatel a aktivita na sebe navzájem nemohou působit, ale aktivitu je vidět na obrazovce. Aktivita ztrácí fokus, ale je stále viditelná. To se stane v době, kdy aktivita byla částečně překryta jinou aktivitou, roztáhnutou na část obrazovky nebo mající transparentní strukturu. Pozastavená aktivita je „živá“, protože zachovává veškerou informaci a zůstává připojená se správcem oken(Window Manager). Může byt zničena systémem v extrémních situacích.

3. **stopped** (zastavený stav)
   Aktivitu není vidět, aktivita je zcela překryta jinou aktivitou. Uživatel nemůže navzájem působit na aktivitu. Ale aktivita si stále zachovává všechny informace uživatele, nicméně už není viditelná, správce oken je skrytý. Bude zničena systémem, až bude potřeba získat paměť za jinými účely.

4. **finished** nebo **killed** (zničený stav)
   Byla-li aktivita přerušena nebo zastavena, systém může smazat aktivitu z paměti buď zažádáním ji dokončit nebo zničením jejích procesů. Pro opětovné zobrazení uživateli, musí být kompletně restartována a obnovena do původního stavu. Když se aktivita přesune z jednoho stavu do druhého, android systém volá metody zpětného volání životního cyklu aktivity (*lifecykle callbacks*).

Další stavy aktivity jsou dočasné a systém se rychle přesouvá do dalšího stavu, což způsobuje další fáze životního cyklu.

Z následujícího schématu je zřetelně vidět všechny přechodné stavy a metody používané pro tento účel:



Obrázek 2.5. Životní cyklus aktivity

Kapitola 3
Související práce

Tato bakalářská práce zahrnuje různé obsáhlé pojmy a oblasti, ke kterým tyto pojmy patří. Zmíněné pojmy jsou adaptivita a kontext, aspektově-orientované programování, senzory, uživatelská rozhraní. Existují frameworky, které jsou zaměřené na řešení úkolů v rámci své oblasti: některé využívají AOP přistup, některé řeší otázky generování uživatelských rozhraní, některé se zabývají adaptací systému. Část existujících frameworků jsou spojením několika oblasti najednou, příkladem jsou frameworky, které řeší otázky generovaní uživatelských rozhraní na základě kontextu. V současné době nejsou známé frameworky, které by využívali všechny výše popsané oblasti najednou. Ale určitě existují dobré nástroje v rámci svých oborů, jejichž popis bude představen níže.

3.1 LILOLE framework

LILOLE (*Lifelong Learning*)[18] je framework, který zajišťuje celoživotní učení na základě toku dat ze senzorů a slouží k prediktivnímu modelování uživatele. Framework usiluje o přístup, umožňující všudypřítomným systémům neustále se učit a přes proud na bázi aktivního učení využívat získané znalosti k adaptaci systému.

Základem realizace slouží Sens-ation platforma, která je založená na událostech (*event-based platform*) pro všudypřítomnou výpočetní techniku. Tato platforma poskytuje integrace senzorů, inferenční stroj, mající schopnost logického odvozování (porovnává nově vstupující a známá data) a pohony. Framework realizuje různé strategie v podobě logického výstupu pro Sens-ation platformu, které mohou být nakonfigurované v editoru.

3.2 Aspect Faces Java EE

Aspect Faces [19] – je nástroj, který slouží k automatickému vytváření částí uživatelského rozhraní. Základem AF je metamodel, který reprezentuje strukturu sestavení UI. AF implementuje techniku *Rich-Entity Aspect/Audit Design*(**READ**), která je založena na aspektově orientovaném principu programování. AF umožnuje generovat rozhraní jak staticky, tak i během runtimu(dynamicky). AF řeší problém průřezové funkcionality. Hlavními aspekty jsou například: vlastnosti třídy, jména tříd, datové typy, komponenty, které reprezentují data, security, validace. Aspect Faces je programátorsky příznivý, protože stačí popsat každý aspekt jen jednou, a tím se snižuje množství ručně vytvořeného zdrojového kódu aplikace.

3.3 Framework pro Android

Framework[2] [20] slouží ke tvorbě mobilních aplikací s operačním systémem Android. Framework je implementován s využitím aspektově-orientovaného přístupu. Podporuje následující aspekty: security(bezpečnost), layout, ověřování vstupu(input validation), data binding a presentation(prezentace).

Aspekt „**security**“ odpovídá za nastavení role uživatele, kterou lze přidělit pomocí anotací @UiUserRoles. Aspekt „security“ určuje podle dříve nastavené role, jakou informaci může vidět uživatel a která pro něj není dostupná.

Aspekt „**layout**“ umožňuje vývojáři použit šablonu layoutu, která je součástí frameworku. Vývojář může vytvořit svůj vlastní layout, ale pokud mu vyhovuje využít předem definovanou šablonu tohoto frameworku, tak si může vybrat tuto šablonu. Framework poskytuje na výběr dva layouty: jedno-sloupcový layout a dvou-sloupcový layout.

Aspekt „**input validation**“ odpovídá za korektní nastavení defaultních hodnot(například, @NotNull). Pokud nějaká hodnota není korektní, uživatel bude informován Toast zprávou a změní se barva pozadí na červenou. Tento aspekt také kontroluje provádění činností při změně hodnot a před odesláním dat.

Aspekt „**data binding**“ se řeší v runtimu pomoci java reflection. Demonstruje, jakým způsobem vývojář může nastavit defaultní hodnoty. Má dvě možnosti: první možnost je uvést hodnoty v konstruktoru, druhá možnost je nechat to na program, který dynamicky vytvoří hodnoty a zatím je propojí s odpovidajícími atributy.

Aspekt „**presentation**“ umožňuje znázornit data v závislosti na jejich typu. Framework poskytuje využití dat tři typů: String, int a password.

3.4 AOP nastroje

Většina nástrojů při implementaci programu na principech AOP využívají join pointy, pointcuty, rady(advices) a aspekty. Klíčovými rozdíly AOP nástrojů jsou způsoby deklarace aspektů. Zároveň nástroje mají odlišení v konvencích pojmenovaní, ale všude je využit stejný princip. Nástroje můžou být vytvořeny jako:

- framework nad programovacím jazykem;
- rozšířeni konkrétního programovacího jazyka;
- nový aspektově-orientovaný jazyk.

Tady jsou představeny některé nástroje, využívající aspektově-orientované principy[21].

3.4.1 AspectJ

AspectJ je rozšíření syntaxe a sémantiky jazyka Java, který umožňuje deklarovat aspekty přímo v kódu. AspectJ poskytuje statické AOP, tj. weaving probíhá při buildu aplikace. AspectJ je open sourcem, který je distribuován pod licencí Eclipse. Pro překlad program používá vlastní kompilátor – ajc, který vytváří všechny výchozí texty projektu, včetně java kódu. AspectJ poskytuje vlastní sadu klíčových slov pro práci s aspekty a obsahuje relativně velký objem podporovaných join pointů. Pro definování join pointů se používají klíčová slova a regulární výrazy.

Velkou výhodou AspectJ je zjednodušení vývoje programu, díky snížení množství kódu, než při vytváření klasické aplikace. Díky kratší délce kódu se potenciálně snižuje pravděpodobnost výskytu chyb.

3.4.2 Spring AOP

Spring AOP poskytuje jemně-laděnou konfiguraci a schéma propojení. Spring AOP nemění syntaxi programovacího jazyka Java a používá standardní Java kompilátor. Realizace aspektově-orientovaného programování v Spring je založena na využití objektů-prostředníků(*proxy*).

Aspekt v Spring AOP je deklarován pouze v XML. Převahou XML stylu je kontrola nad propojováním pointcut a advice, což poskytuje výhody při expanzi a konfiguraci aspektů.

Pokud programátor chce změnit pointcut, tak se přeorientuje z advice na odpovídající pointcut v XML-složce. Na jednu stranu tento postup pomáhá lokalizovat konfigurace poincatů a aspektů, na druhou editování velkého počtu textu vede ke zvýšení pravděpodobnosti výskytu chyb. Pravidelné přepínání mezi Java a XML-složkami může unavovat programátora.

Knihovny Spring AOP mají přednost v přenášení mezi aplikačními servery. Modularita prostředí ulehčuje proces nástroje pro AOP podporu.

Kapitola 4
Analýza a design frameworku

Framework byl vyvíjen v prostředí Android Studio a je určen pro vývoj aplikací pro mobilní zařízení s operačním systémem Android. Ke splnění požadavků univerzálnosti a praktického použití, framework má modulární strukturu, která umožňuje provádět nástroje jednotlivých komponent. Framework využívá princip aspektově orientovaného programování a slouží ke generování adaptivního rozhraní prostřednictvím kontextu, získaného ze senzorů mobilního zařízení.

4.1 Package diagram

Package diagram je diagram, který znázorňuje vazby mezi balíky, tvořící strukturu celého frameworku. Tento diagram ilustruje modularitu frameworku a popisuje funkčnost celého systému.

- Balík „**context**“ obsahuje třídy, které jsou určeny k napojení na různé typy senzorů a využití veškeré informace z poskytovaných senzorů.

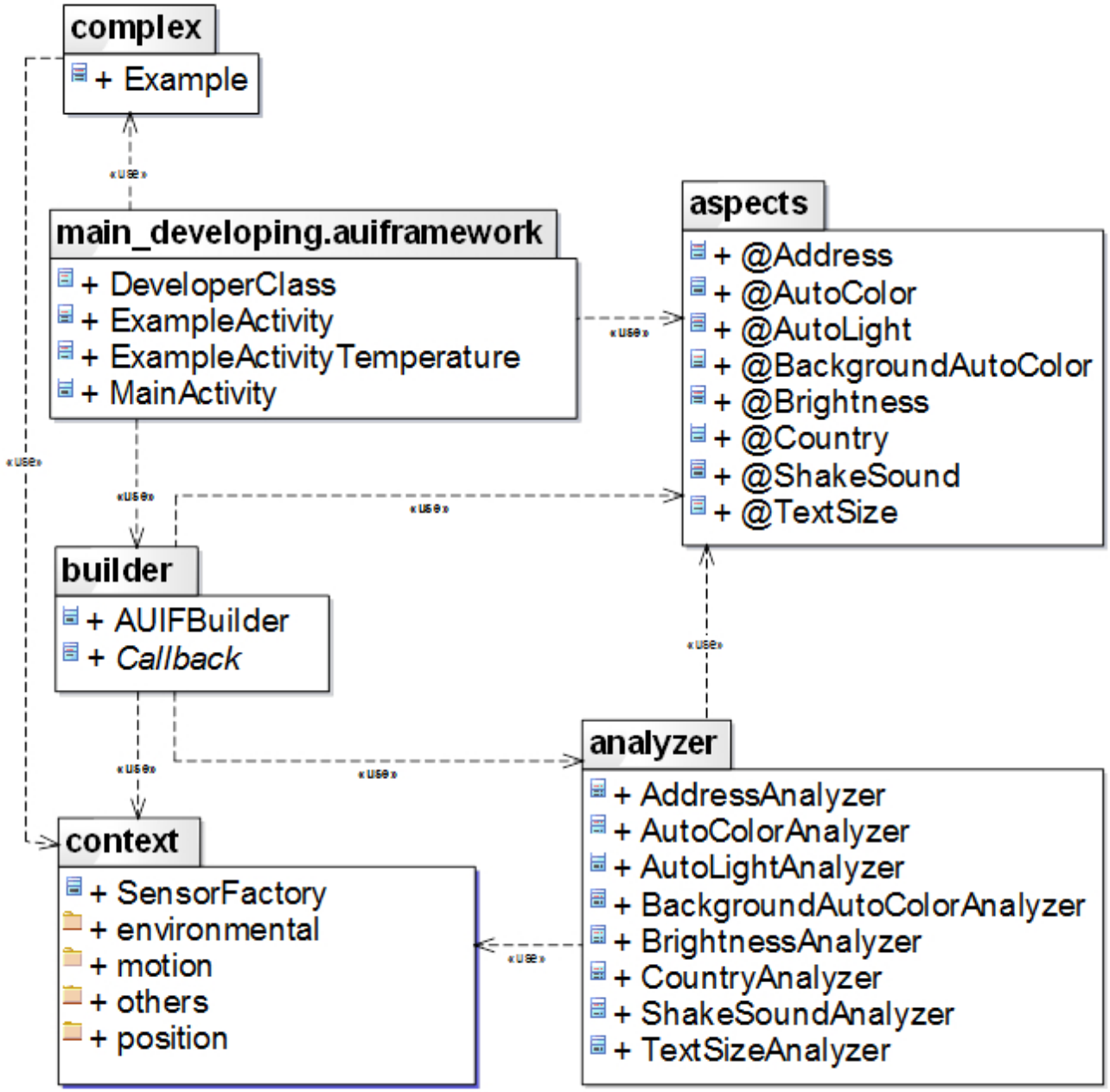
- Balík „**complex**“ patři k logické vrstvě. Tedy je umožňováno rozšíření na další funkčnosti, které potřebuje vývojář. Například, když vývojář bude chtít uskutečnit zvláštní funkci využitím několika senzorů(přesně napsat počítání rychlosti spojením kontextu akcelerometru, barometru, magnetometru a gyroskopu), tak na to může použít tento balík. V něm je příklad, jak se volají jednotlivě senzory.

- Balík „**aspects**“ realizuje aspekty, které podporují aspektový styl programování.

- Balík **„main_developing.auiframework“** je určen pro vývojáře. Vývojář může použít balík „**complex**“ také, pokud bude potřebovat navrhnout nějakou logickou funkcí, kterou neobsahuje tento framework.

- Balík „**builder**“ slouží ke spuštění frameworku. Jakmile programátor spustí aplikaci, tak hned bude vytvořena instance třídy AUIFBuilder z tohoto balíku. Tato třída načte veškerá data a anotace, které byly využity programátorem, nainicializuje potřebné údaje, získá a uloží kontext, který použije k adaptaci rozhraní na základě programátorem vybraných funkcí.

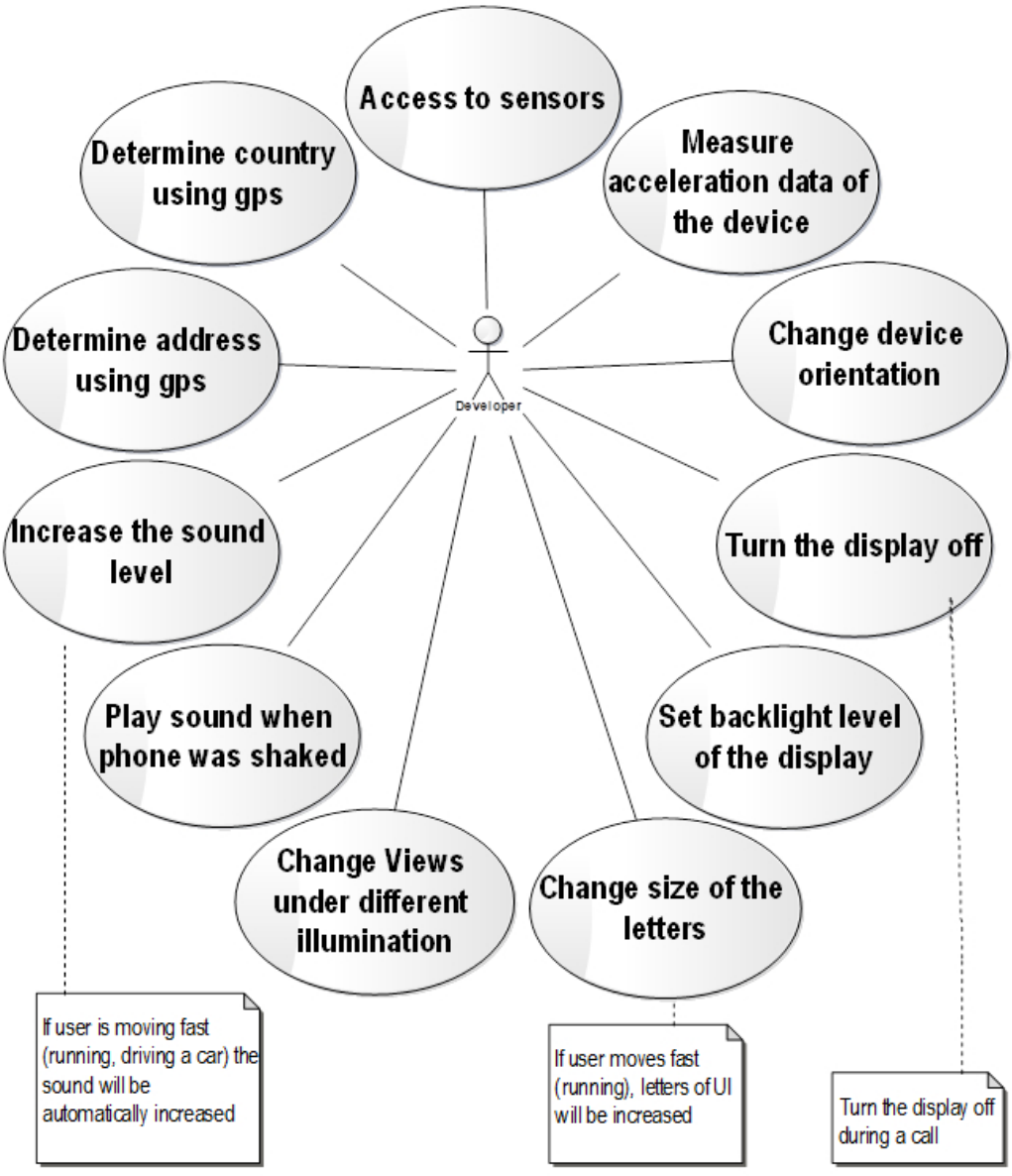
- Balík „**analyzer**“ se používá ve třídě AUIFBuilder, která se nachází v balíku „**builder**“. Jakmile bude aplikace spuštěna, objekt třídy AUIFBuilder nasbírá počáteční informace, získá všechna potřebná metadata a pošle získané data dál balíku „**analyzer**“ na další přesnější analýzu a provedeni akcí.



Obrázek 4.1. Package diagram

4.2 Use Case Diagram

Use Case Diagram je diagram případů užití. Tento digram odráží vztah mezi budoucím vývojářem a funkcemi, které stanoví framework. Diagram ukazuje vývojáři, jaké funkce frameworku jsou pro něho k dispozici.

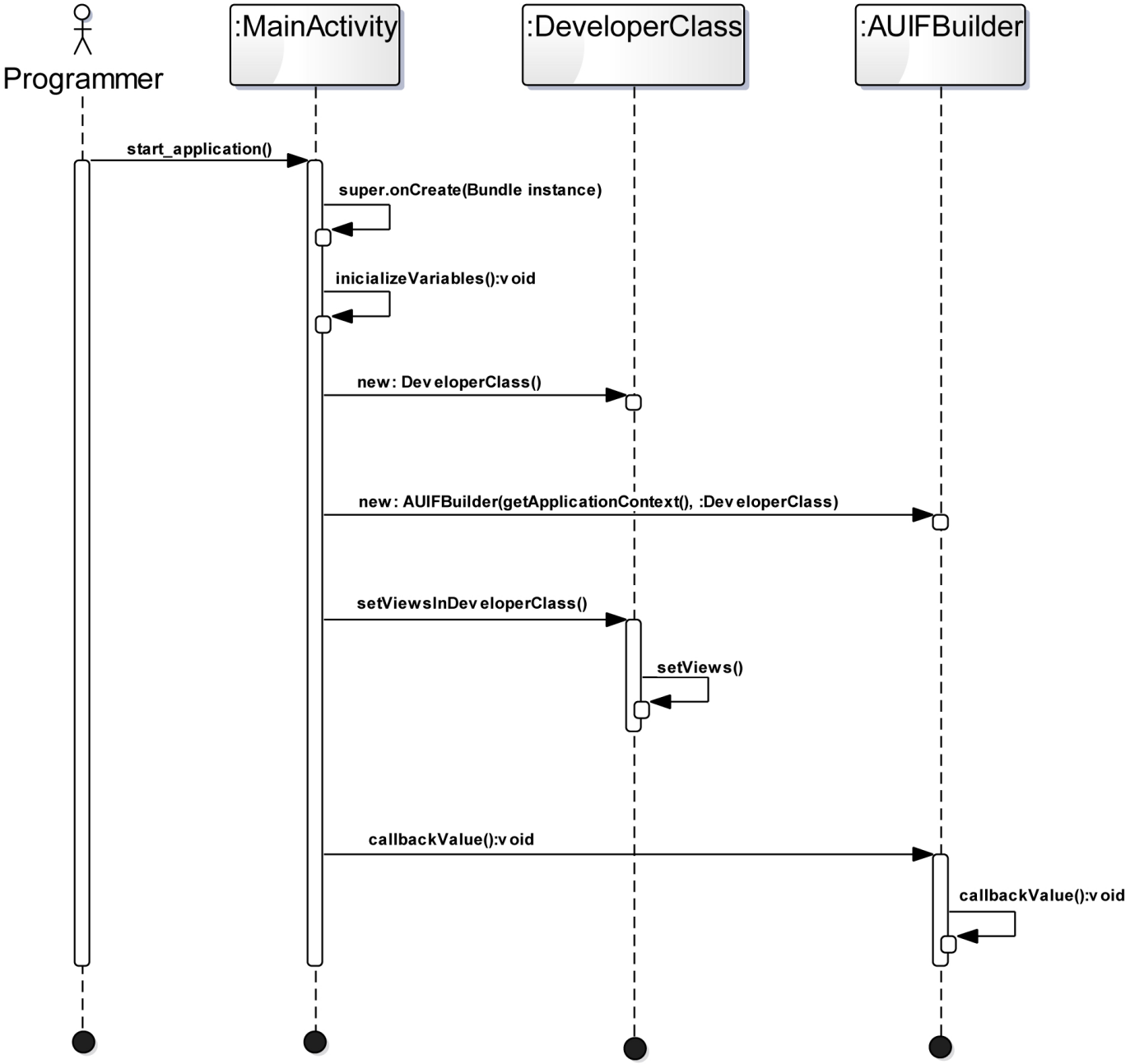


Obrázek 4.2. Use Case Diagram

4.3 Sekvenční diagram

Sekvenční diagram je vhodným nástrojem pro definování pořadí po sobě jdoucích různých zpráv, pomocí kterých objekty na sebe vzájemně působí. Sekvenční diagram odráží dynamiku chování objektů, tj. zachycuje pořadí přicházejících událostí. V sekvenčním diagramu jsou zobrazeny pouze ty objekty, které jsou přímo zapojené do interakce.

V době, kdy programátor spustí aplikaci, bude jako první volána metoda *onCreate*(), která je ve třídě MainActivity. Pomocí diagramu níže, je popsán způsob volání frameworku v metodě *onCreate*().



Obrázek 4.3. Sekvenční diagram

Objekt se zobrazuje jako obdélník umístěný v horní části své linie života (*lifeline*). Uvnitř obdélníku je zaznamenán název objektu a název třídy, oddělené dvojtečkou. V diagramu jsou znázorněné tři objekty: MainActivity, DeveloperClass a AUIFBuilder. V metodě $onCreate()$, která je volaná v MainActivity jsou vytvořené dva objekty: „clazz:DeveloprClass“ a „builder:AUIFBuilder“.

Lifeline - svislá čára, která slouží k označení běhu časového období, během něhož objekt existuje v systému a potenciálně může se zúčastnit všech interakcí.

Actor je účastník, který je vzhledem k vyvíjenému systému vnější. V diagramu je označován jako „Programmer“.

Zpráva self - zpráva, kterou odesílatel posílá sám sobě. V diagramu to jsou metody:

- $super.OnCreate(Bundleinstance)$ - nastavuje parametry pro činnost hlavní aktivity.
- $inicialazeVariables()$ - inicializuje všechny TextViews, které jsou popsané v xml layoutu

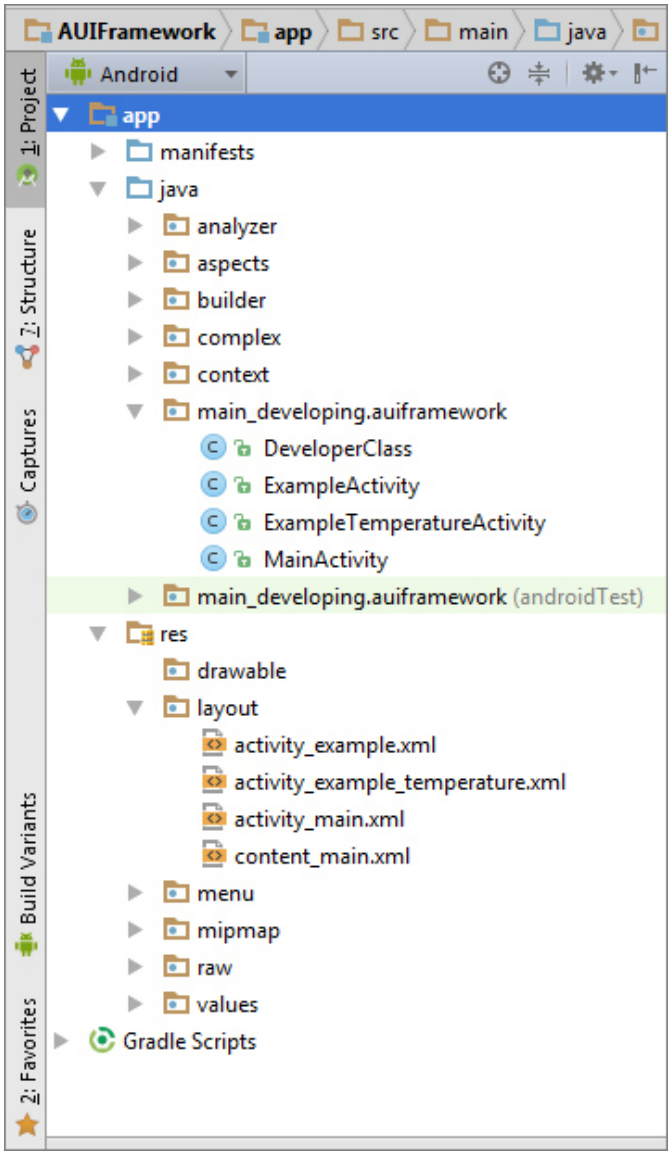
Asynchronní událost označuje horizontální šipka, která indikuje odesílání zprávy nevyžadující odpověď od příjemce. Po odeslání zprávy odesílatel hned pokračuje ve své práci. Objekt, který přijal zprávu, začíná samostatně kontrolovat provádění operací.

- $newDeveloperClass()$ - vytváří instance třídy DeveloperClass
- $newAUIFBuilder(getApplicationContext(), developerClass)$ - vytváří instance třídy s aktuálním kontextem, která spustí celý framework
- $setViewsInDeveloperClass()$ - nainicializuje všechny textView, které bude chtít vývojář adaptovat
- $callbackValue()$ - spustí načtení kontextu, aplikace se spustí na emulátoru nebo v reálném smartfonu a rozhraní začne se adaptovat

Kapitola 5
Implementace

Pro implementace frameworku bylo využito vývojové prostředí AndroidStudio firmy JetBrains, které je založeno na IntelliJ IDEA. Programovací kód frameworku je popsán v jazyce Java. Pro prezentace funkčností frameworku je klasicky využit rozšiřitelný značkovací jazyk XML.

5.1 Struktura projektu Android Studio



Obrázek 5.1. Struktura projektu v Android Studio

Stejně jako jakýkoli jiný projekt Java, základní projekt v Android Studio má stromovou strukturu. Hlavní částí této struktury jsou:

- **manifests**
  Manifest nebo konfigurační soubor aplikace — XML-soubor, který má vždy název AndroidManifest.xml. Manifest zadává konfiguraci a popisuje základní vlastnosti aplikace: deklaruje aplikační komponenty, uvádí externí knihovny, které využívá aplikace, deklaruje povolení, která jsou nezbytná pro fungování aplikace (například povolení pro přístup na internet).

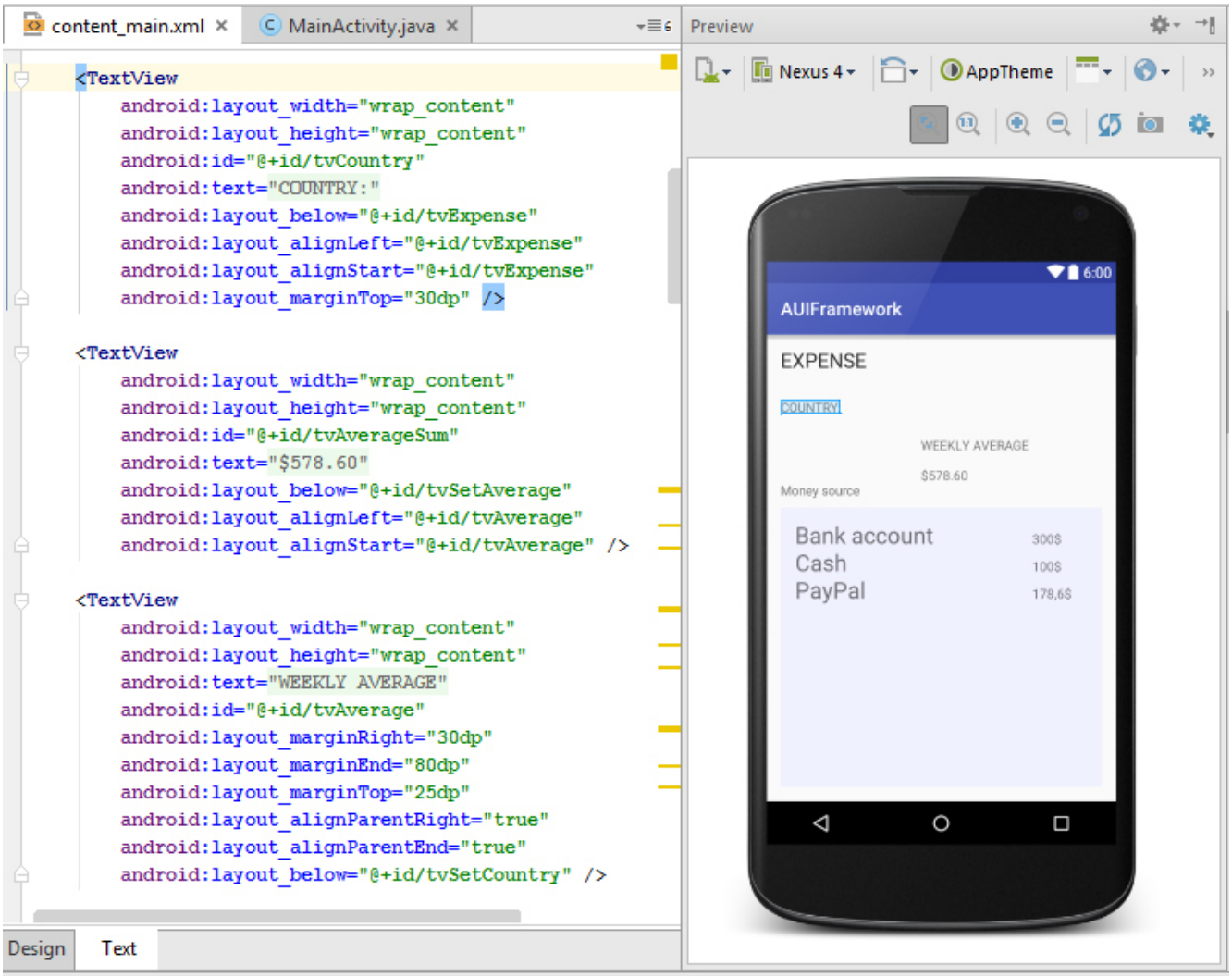
- **java**
  Balík „java“ obsahuje veškerý kód frameworku. Při vytváření projektu v jednotlivé složce v balíku „java“ se vytváří třída MainActivity, což je defaultní třída, která je spuštěná při startu aplikace.

- **res**
  Do res adresáře se ukládají pro aplikace užitečné statické soubory zdrojů: obrázky, řetězy, atd. Většinou jsou zdroje uloženy v XML souborech. Systém automaticky vytváří identifikátory zdrojů a využívá je v souboru R.java. Třída R obsahuje odkazy na všechny zdroje projektu. Tento postup umožňuje odkazovat na zdroje v rámci programového kódu. Statická třída R je generována automaticky na základě definovaných zdrojů a je vytvořena při kompilaci projektu. Některé z podadresáře res jsou generovány při prvotním vytváření projektu, jiné se přidávají samostatně. Typicky res zahrnují následující podadresáře:

  - **res/drawable/** - podadresář je určen k ukládání obrázků použitých v aplikaci
  - **res/layout/** - podadresář je určen k ukládání XML-souborů, které definují grafické rozhraní aplikace
  - **res/menu/** - obsahuje různé XML soubory definující menu aplikací
  - **res/values/** - používá se pro uschovávání zdrojů konstant různých typů (String, Color atd.)

5.2 Rozhraní XML vs. Rozhraní JAVA

Existují různé způsoby, jak lze implementovat uživatelské rozhraní aplikace v Androidu. Jeden ze způsobů je vytvoření View-komponenty programově přímo v Java kódu. Druhý způsob je vytvoření aplikace s použitím res/layoutu. V nově vytvořeném layoutu se zformují makety obrazovky, v xml souboru se popisují View-komponenty uživatelského rozhraní aplikace. Příkladem je následující obrázek:



Obrázek 5.2. Příklad rozhraní XML s použitím res/layoutu

Při vytváření frameworku byl preferován druhý způsob. Odůvodnění jsou:

- xml soubor ve frameworku je názorným příkladem, jak lze využívat framework. Vytváření obrazovky programově je méně názorný způsob, neboť je možné se podívat na výsledek, jak vypadá obrazovka, až bude spuštěna aplikace. Z těchto důvodů se využívá xml způsob, aby vývojář názorně viděl, jaké komponenty jsou použity a jak se budou adaptovat po spuštění aplikace.
- většinou jsou příklady na internetu nebo v knihách uvedeny více klasickým způsobem, což je xml prezentace komponent rozhraní.

- ve frameworku není vytvářena žádná šablona, aby měl vývojář prostor pro generování vlastního rozhraní v závislosti na jeho vkusu a preferencích. Adaptují se většinou TextViews podle nastavení vývojáře. Pokud vývojář nebude chtít využívat xml, tak může psát kód programově a zatím v určené třídě DeveloperClass označit, jaké TextView chce adaptovat pomoci jednotlivých aspektů.

5.3 Aspekty

Pomocným nástrojem pro adaptaci jednotlivých prvků rozhraní je anotace. Aspekty frameworku jsou tvořeny anotacemi. *Anotace* je zvláštní forma metadat, která obsahuje informaci o programovacím kódu, ale není součástí programu. Anotace nevykonává žádnou činnost, ale je ukazatelem k provedení požadovaných akcí. Anotace poskytuje informace kompilátoru, jaké události musí provést. Níže je uveden obrázek pro představu, jak se vytváří jednotlivé anotace:

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface Brightness {

    String morning() default "1F";
    String day() default "0.5F";
    String night() default "0.2F";

}
```

Obrázek 5.3. Příklad na vytváření anotace

- `@interface` je klíčové slovo k zavedení nové anotace a má definované pole, které mohou být nastavené jako defaultní

- `@Retention` určuje, jak bude anotace uložena
  - RetentionPolicy.*SOURCE* — pouze ve zdrojovém kódu
  - RetentionPolicy.*CLASS* — ve zdrojovém kódu a .class souboru
  - RetentionPolicy.*RUNTIME* — ve zdrojovém kódu a .class souboru a v době runtimu. Anotace @Brightness bude k dispozici během runtimu.

- `@Target` - definuje typ, ke kterému se uplatní anotace @Brightness.
  - ElementType.*FIELD* - definuje se nad poli. Anotace @Brightness může být nastavena jenom nad polem. Například:

```
@Brightness(day = "0.5F",night = "0.2F")
public Window window;
```

Obrázek 5.4. Nastavení anotace @Brightness nad polem.

  - ElementType.*ANNOTATION_TYPE* - definuje se nad anotacemi
  - ElementType.*LOCAL_VARIABLE* - definuje se nad lokálními proměnnými

- ElementType.*METHOD* - definuje se nad metodami
- ElementType.*TYPE* - definuje se nad třídami, interfacy, anotacemi a enumy

Pomocí anotací jsou ve frameworku tvořené následující aspekty, uskutečňující dříve označené use-casy:

- @Address
- @AutoColor
- @BackgroundAutoColor
- @Brightness
- @Country
- @ShakeSound
- @TextSize

5.4 USE-CASE: @Country a @Address

Tento use-case popisuje získání kontextu pomocí existující funkce gps-senzoru. Tato funkce určuje stát, ve kterém se nachází uživatel a stát se zobrazuje na obrazovce smartfonu. Je vhodné tuto funkci využít, když uživatel vyplňuje okno registrace. Stát se automaticky nastaví a uživatel ho nebude muset vpisovat do textového pole.

K získání GPS-kontextu ve frameworku slouží instance třídy GPSContext. V rámcích této třídy se k nalezení gps-kontextu používá instance třídy LocationManager.

Třída LocationManager je vlastní třídou Androidu. Poskytuje přístup ke službám systému určení polohy. Tyto služby umožňují aplikacím dostávat pravidelné aktualizace geografické polohy mobilního zařízení. Na místo instance této třídy se volá kontext servisu systému, jak je ukázáno níže:

```
locationManager =
(LocationManager) context.getSystemService(Context.LOCATION_SERVICE);
```

Příklad kódu 1: Inicializace objektu LocationManager

Aby funkce třídy LocationManager byly přístupné je potřeba v AndroidManifest.xml ukázat povolení:

```
<uses-permission android:name =
„android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name =
„android.permission.ACCESS_COARSE_LOCATION"/>
```

Příklad kódu 2: Povolení v AndroidManifest.xml

Dále se využívá interface posluchače LocationListner, který je napojen na poskytovatele služby. Poskytovatelem služby může být buď GPS, což jsou data z GPS-družic nebo Network, což jsou souřadnice polohy, které se dají získat pomocí mobilního spojení nebo WIFI.

Jedna z metod rozhraní LocationListner je metoda, která je volána, pokud došlo ke změně umístění:

```
@Override
public void onLocationChanged(Location location)
```

Příklad kódu 3: Metoda určení změny umístění v LocationListner

Využitím parametru *location* ve výše uvedené metodě se dají získat zeměpisná délka a šířka. Tyto dvě hodnoty se konvertují do jejich „slovního vyjádření“, například:

šířka: 50.0880400
délka: 14.4207600
slovní vyjádření: Czech Republic

Konverze se dělá pomocí třídy *Geocoder*, která obdří jako parametr zeměpisnou šířku a délku, zatimco využitím internetu zkonverzuje data do „slovního vyadření“.

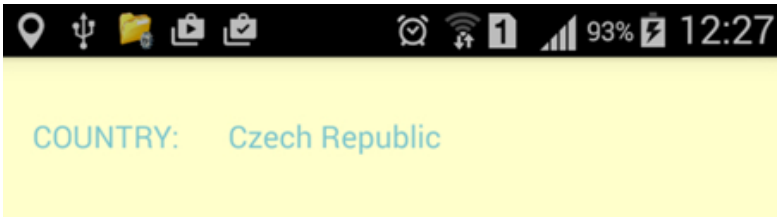
V balíku *„main_developing.auiframework“*, který slouží pro vlastní třídy programátora, programátor vytvoří svou třídu, ve které označí anotací `@Country`, že chce do proměnné typu „String“ přidat stát.

```
@Country
public String country;
```

Příklad kódu 4: Příklad použití anotace @Country

Dále programátor potřebuje přidat svou třídu do AUIFBuilderu a spustit framework. Při spuštění frameworku se rychlé nainicializují data ve třídě frameworku ContextGPS a zobrazí se na obrazovce.

Názorný příklad poskytuje následující obrázek:



Obrázek 5.5. Zobrazení státu na obrazovce

Postup na nalezení adresy je stejný jako na nalezení státu. K určení adresy je potřeba využít anotaci `@Address`.

Existuje omezení, které spočívá v tom, že není možné ve všech případech určit přesnou adresu. Byl proveden experiment, ve kterém byla třikrát za sebou změřena adresa při nacházení se na stejném místě, ve stejném pokoji. Výsledkem je:

Obrázek 5.6. Experiment nalezení adresy

Aktuální adresa polohy byla *Chaloupeckého 1915/7*. Družice poskytují adresu blízko adresy aktuální, ale není žádná garance, že adresa bude určena přesně, což se zjistilo empirickým způsobem pomocí experimentu.

5.5 USE-CASE: @AutoColor a @BackgroundAutoColor

Tyto dva aspekty slouží ke změně barvy různých textových polí TextView a barvy pozadí. Jsou to poměrně populární funkce, například v navigátorech. Jakmile auto zajede do tunelu, změní se barva cesty a písmen na žlutou a pozadí bude tmavé. Při výjezdu z tunelu bude barva pozadí světlá a písmena budou napsána tmavými barvami.

Za účelem získání kontextu se používá třída *LightContext*. Snímač pomáhá změřit osvětlení kolem uživatele a na základě naměřených hodnot framework nastavuje barvu textu v TextView a barvu pozadí. Ve frameworku k tomu slouží anotace `@AutoColor` a `@BackgroundAutoColor`. Tyto anotace jsou stejné za výjimkou pole, nad kterým se dají definovat. @AutoColor se umisťuje nad $List < TextView >$ a @BackgroundAutoColor se musí nastavit nad *Window*. Pro oba tyto atributy ve třídě DeveloperClass je nutné napsat setter.

Struktura anotace @AutoColor je:

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface AutoColor {

    String morning() default "black";
    String day() default "grey";
    String night() default "white";
}
```

Obrázek 5.7. Struktura anotace @AutoColor

Nastavené defaultní barvy jsou:

- *morning*() - označuje barvu textu TextView nebo pozadí v době, kdy je hodně světlo ($>$ 50 luxů). Pokud je kolem uživatele dostatečné světlo, bude defaultní barva textu černá, defaultní barva pozadí bude bílá.

- *day*() - označuje střední světlo, které je nastaveno na 5-50 luxů. Pokud je kolem uživatele osvětlení jako ve statisticky průměrném pokoji, bude defaultní barva textu šedá a pozadí - světle žluté.

- *night*() - označuje, že kolem uživatele je tma ($<$ 5 lux). Pokud bude kolem uživatele hodně šero, jako když venku svítí měsíc, tak bude defaultní barva textu bílá a barva pozadí tmavě fialová.

5.6 Průřezové body aspektů @AutoColor a @BackgroundAutoColor

Programátor má možnost nastavit jakoukoliv barvu podle svých preferencí. Existuje několik způsobů, jak se dá nastavit barva:

```
@AutoColor(day=„grey", night=„#7ac5cd")
List<TextView> list = new ArrayList<>();

@AutoColor (day=„rgb(122,197,205)", morning =„blue", night=„#6a2876")
List<TextView> list2 = new ArrayList<>();
```

Příklad kódu 5: Příklad použití anotace @AutoColor

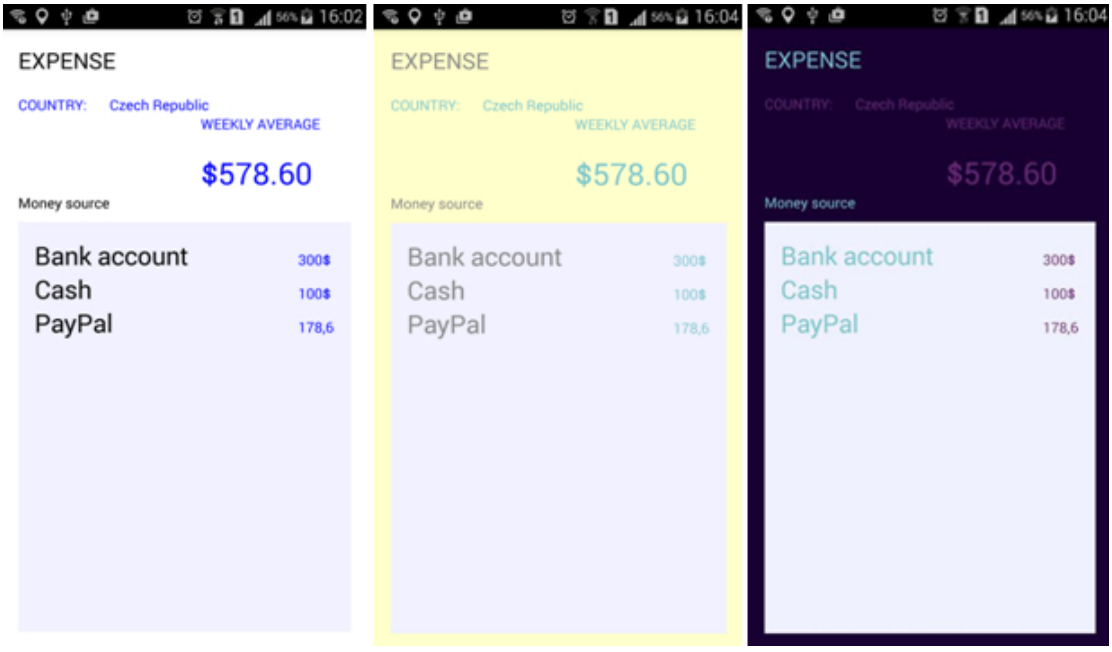
Pokud @AutoColor nemá parametry uvnitř závorek, tak bude text TextView nastaven na defaultní hodnoty.

Programátor může využít následující nástroje barev:

- **klasický způsob**: „white“, „grey“, „black“ a další, které jsou používané ve standardní java třídě *java.awt.Color*
- **rgb způsob**: nastavit hodnoty červené(red), zelené(green), modré(blue) barvy od 0 do 255. Tímto způsobem se umožňuje získání velkého počtu různých odstínů
- **hexadecimální způsob**: nastavení barvy vždy začíná znakem **#**. Také tento způsob poskytuje velký počet různých barev a jejich odstínů

Při spouštění frameworku se třída AUIFBuilder dozví, že je aktualizovaná anotace @AutoColor. Následně bude vytvořena instance třídy LightSenzorContext a využije se metoda této třídy *getIlluminance*(), která zajistí měření osvětlení. Podle naměřených hodnot osvětlení, texty TextView nabydou barev, které dříve naznačil programátor.

Příklad změny barvy:



Obrázek 5.8. Příklad obrazovky při využití @AutoColor

5.7 Use-Case @TextSize

Tento use-case určuje velikost písmen v závislosti na rychlosti, kterou se pohybuje uživatel zařízení.

Představa, co je „rychlé“ a co je „pomalejší“, se v různých případech může lišit. Pokud je aplikace určena pro uživatele, který chodí pešky, tak střední rychlost pohybu bude 5-7km/hod. Pokud je aplikace určena pro uživatele, kteří jezdí na kole, tak se označení střední rychlosti výrazně zvyšuje. Proto aspekt @TextSize umožňuje zadefinovat pointcaty, pomocí kterých vývojář určí, jakou rychlost lze chápat jako „rychlou“ a jakou jako „pomalejší“. Na to využije:

- **speedSlow**
- **speedMiddle**
- **speedFast**

Samotný aspekt adaptuje velikost písmen v závislosti na rychlosti. Vývojář musí přidat velikost v bodech:

- **sizeSlow** - velikost TextView, která se nastaví, když se uživatel bude pohybovat pomalu
- **sizeMiddle** - velikost TextView, která se nastaví, pokud se uživatel bude pohybovat střední rychlostí
- **sizeFast** - velikost TextView, která se nastaví, jakmile se uživatel bude pohybovat rychle

Taky lze měnit barvu v závislosti na rychlosti:

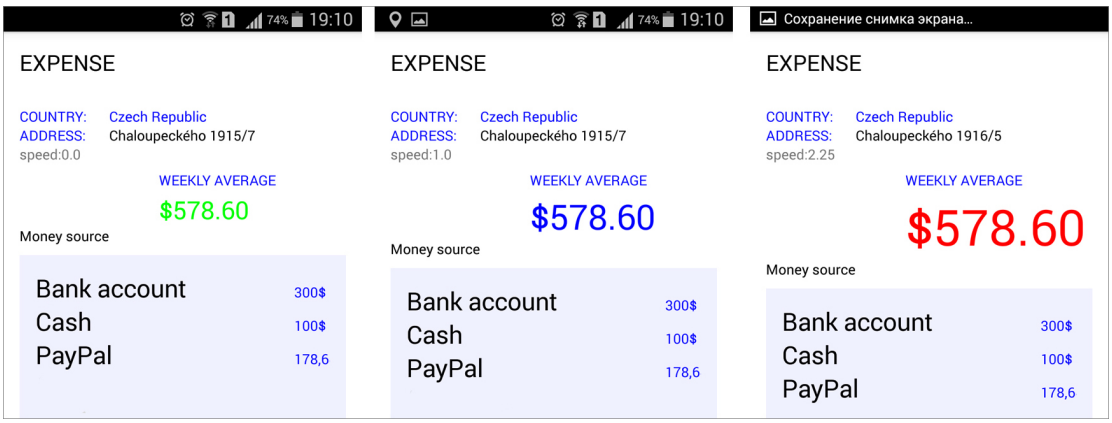
- **colorSlow** - barva TextView, která se nastaví, když se uživatel bude pohybovat pomalu
- **colorMiddle** - barva TextView, která se nastaví, když se uživatel bude pohybovat střední rychlostí
- **colorFast** - barva TextView, která se nastaví, pokud se uživatel bude pohybovat rychle

Důležitou částí tohoto aspektu je to, že vývojář musí zadefinovat v MainActivity(nebo libovolné jiné třídě, která bude směřovaná k adaptaci rozhraní) konkrétní TextView, u kterého chce zvětšit písmena. Poté musí přidat dané TextView do setteru DeveloperClass.

Jinými slovy strategie lze popsat následujícím způsobem:

1. Zadefinovat v DeveloperClass objekt TextView a napsat setter.
2. Najít v MainActivity potřebný objekt TextView podle id klasickým způsobem a přidat objekt TextView do setteru DeveloperClass.
3. Spustit framework.
4. Objekt TextView mění velikost písmen vzhledem k rychlosti pohybu.

Výsledkem je:



Obrázek 5.9. Příklad obrazovky při využití @TextSize

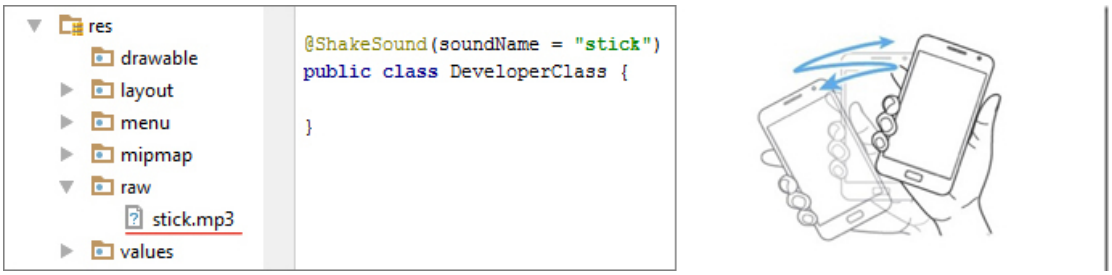
Existuje taky omezení na počítání rychlosti pomocí GPS:

1. GPS smartfonů neposkytuje přesné vypočítaní rychlostí;
2. rychlost se ukazuje na obrazovce s velkým zpožděním.

Funkce gps senzoru na získání rychlosti se dá upřesnit pomocí různých filtrů a využitím pomocných senzorů pohybu. Nicméně libovolná metodika nebude poskytovat přesný výsledek. V současné době přesné gps data jsou k dispozici pouze na specializovaných drahých geodetických zařízeních, které se většinou používají za vojenskými, vědeckými nebo komerčními účely (geodézie, kartografie atd.).

5.8 Use-Case @ShakeSound

@ShakeSound je aspekt, sloužící k nahrávání zvuku, jakmile zatřeseme smartfonem. Je to jediná anotace v frameworku, která se umisťuje nad třídu, ale ne nad polem. Vývojář si může vybrat libovolný hudební úryvek či krátký zvuk a přidat ho do zdrojů "raw". V DeveloperClass napíše anotace @ShakeSound a v závorkách přidá název skladby, kterou chce nahrát při třesení.



Obrázek 5.10. Struktura @ShakeSound. Převzato z [22].

Při implementaci funkce třesení byl vyjmut kontext z akcelerometru, což je zrychlení v třech osách x, y, z. Každá z těchto hodnot obsahuje značení gravitace, kterou je možné odfiltrovat a tím získat hodnoty, které by ukázal lineární akcelerometr. Potom je využita formule na kalkulace zrychlení ze třech os.

Lineární zrychlení ve frameworku se používá pro ošetření třesení. Pokud hodnota zrychlení bude větší než 19, to znamená, že došlo ke třesu smartfonu. A nahraje se hudba pomocí vytvořené instanci třídy MediaPlayer.

```
if(accelerometrContext.getLinearAcceleration()>19){
        player.start();
}
```

Příklad kódu 6: Příklad metody nahrání zvuku

Hodnota $19m/s^2$ byla získaná empiricky a ukázala nejlepší výsledek. Níže je uvedena tabulka měření:

Hodnota$[m/s^2]$	Výsledek
10	Hodně citlivý k třesení, začíná přehrávat zvuk každopádně i při malých třesech.
15	Už je méně citlivý než při 10, ale je potřeba citlivost ještě trochu zmenšit.
19	Zvuk nehraje při úplně malých třesech, ale nahrává se při výraznějších třesech.
23	Zvuk se nahraje, pokud smartfon bude mít velký třes.
27	Zvuk se nahraje, pokud smartfon bude mít hodně velký třes.
30	Zvuk nehraje.

Tabulka 5.1. Empirické nalezení hodnoty zrychlení

5.9 Use-Case @Brightness

```
@Brightness(day = „0.5F", night = „0.2F")
public Window window;

public void setWindow(Window window){
    this.window = window;
}
```

Příklad kódu 7: Příklad použití anotace @Brightness

Window je abstraktní základní třída pro vzhled displeje na nejvyšší vrstvě. Poskytuje standardní zásady uživatelského rozhraní, jako je pozadí, titulní plocha atd.[7]

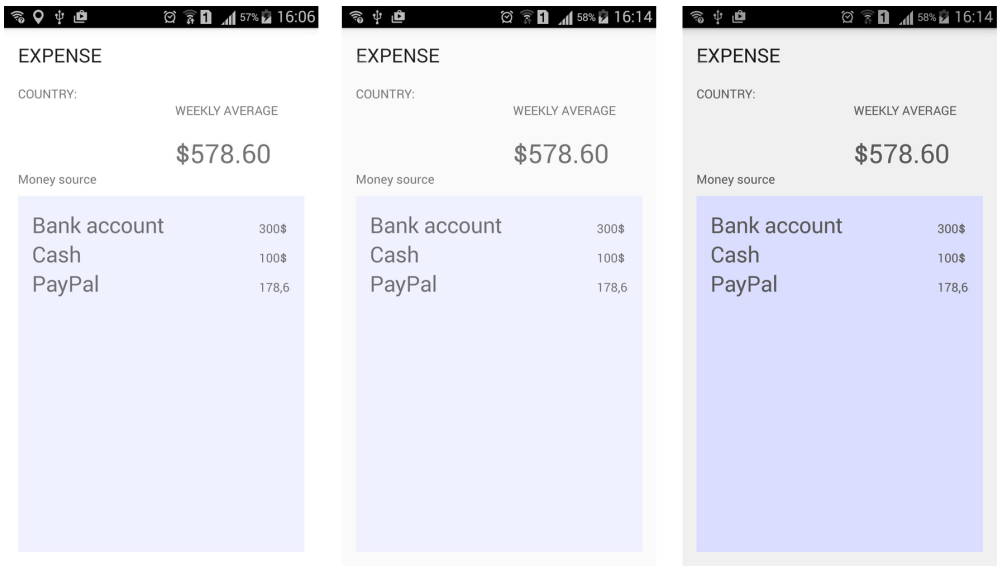
Jas obrazovky v Androidu je představen proměnnou typu float a má hodnoty od 0F do 1F.

- **0F** - nejtmavší
- **1F** - nejsvětlejší

Tyto hodnoty se definuji, jako:

- **day** - jakost obrazovky při průměrném světle 50 lux
- **night**- - jakost obrazovky při průměrném světle $<$ 50 lux
- **morning** - jakost obrazovky při průměrném světle $>$ 50 lux

Výsledek je:



Obrázek 5.11. Příklad změny jasu obrazovky.

5.10 Přístup k senzorům

Framework zapouzdřuje klasický přístup k senzorům, poté poskytuje svůj vlastní přístup. V konvenčním postupu pro přístup se využívá *SensorManager*. Například standardní přístup k senzoru okolního osvětlení a navěšení posluchače (listeneru) na něho vypadá následujícím způsobem:

```
sensorManager = (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);
sensor = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
sensorManager.registerListener(this, sensor, SensorManager.SENSOR_DELAY_NORMAL);
```

Obrázek 5.12. Přístup k senzorům využitím SensorManageru

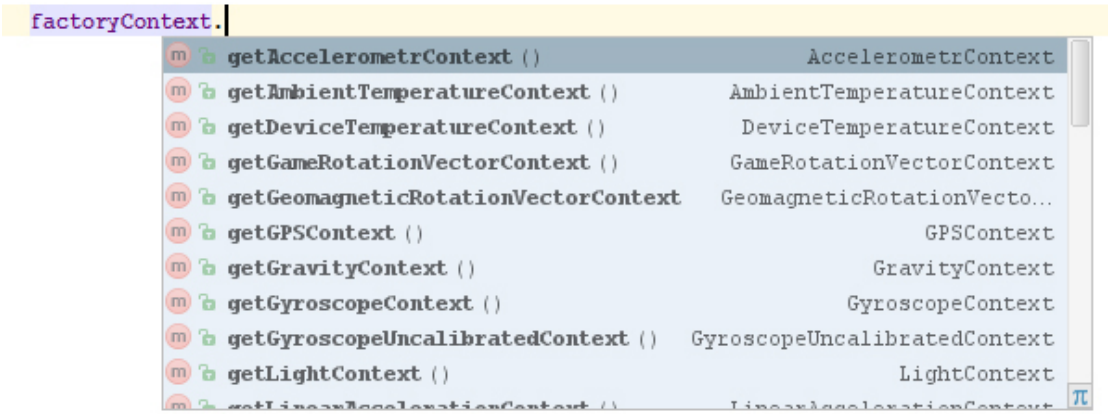
Kontext z každého jednotlivého senzoru se dá získat využitím objektu *event*, který je instancí třídy SensorEvent. Třída SensorEvent reprezentuje události senzoru a drží informace, jako jsou typ senzoru, časové intervaly, přesnost dat a data senzorů. SensorEvent zahrnuje otevřené(public) pole typu float, nazývané *SensorEvent.values*. V něm jsou uloženy aktuální hodnoty pro každý senzor. Aktuální hodnotu ze tří os gyroskopu si lze představit následujícím způsobem:

```
// called when sensor values have changed
@Override
public void onSensorChanged(SensorEvent event) {
    xAxisGyroscope = event.values[0];//Rate of rotation around the x axis. [rad/s]
    yAxisGyroscope = event.values[1];//Rate of rotation around the y axis. [rad/s]
    zAxisGyroscope = event.values[2];//Rate of rotation around the z axis. [rad/s]
}
```

Obrázek 5.13. Příklad na získání kontextu gyroskopu

Účelem třídy *SensorFactory*, která už je součástí frameworku, je poskytnutí jednoduššího přístupu k libovolnému senzoru. Názvy všech senzorů odpovídají názvům

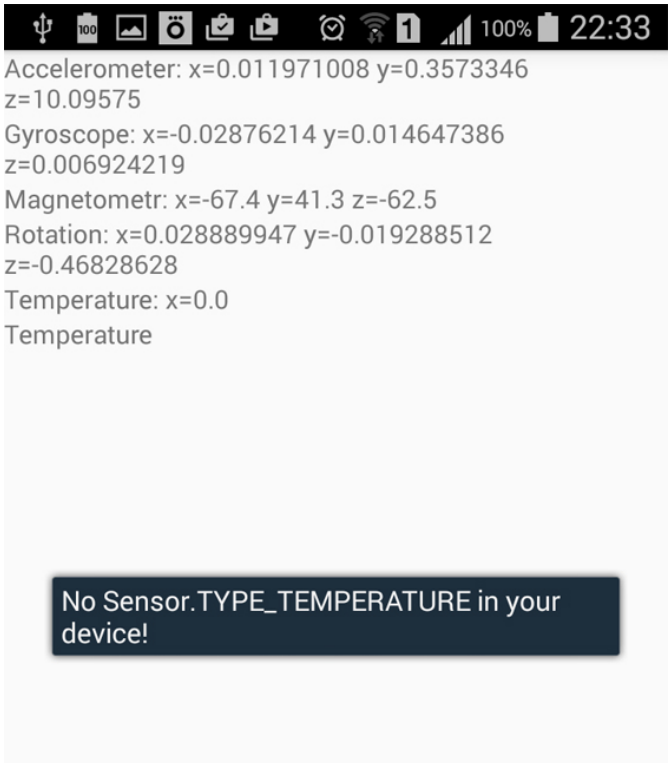
definovaným v Androidu, s výjimkou toho, že mají na konci názvu slovo „Context“. Takovýmto způsobem jsou vidět všechny senzory a jejich typ:



Obrázek 5.14. Příklad získání kontextu pomoci FactoryContext

Pro každý senzor existuje ošetření, zda volaný senzor ve smartfonu je nebo není. Pokud není, tak se na displeji nebo emulátoru bude zobrazovat okno Toast, které upozorní: „*No _TYPE_ in your device!*“.

V balíku „complex“ je vytvořen příklad Example, který ukazuje způsob přístupu k senzorům. Tato třída představuje logickou vrstvu. V balíku „main_developing.auiframework“ je třída ExampleActivity, která popisuje, jak zavolat logiku, kterou poskytuje Example a jak jí znázornit na obrazovce. V příkladu je volán kontext akcelerometru, magnetometru, gyroskopu, gravity a rotation senzorů. Poté se měnící kontext ukáže na displeji smartfonu. Výsledek je následující:



Obrázek 5.15. Zobrazení kontextu senzorů na obrazovce

Kapitola 6
Porovnání

6.1 Framework vs Konvenční přístup: Přístup k senzorům

Funkce přístupu k senzorům ve frameworku se liší od standardního postupu. Tento návrh má své výhody i nevýhody. Nevýhoda je v tom, že není moc snížen počet nutných řádků kódu a není možné najít vhodné příklady na internetu, které by využívaly způsob frameworku AUIF. Programátor se musí přizpůsobit jinému přístupu k senzorům, na což je potřeba čas. Na druhou stranu, nový způsob poskytuje přehlednější názvy. Také framework v balíku „context“ obsahuje šablony s konvenčním přístupem ke všem existujícím senzorům Android. Pokud vývojář nebude chtít použít přístup k senzorům, poskytovaný frameworkem, může okopírovat a použít konvenční přistup ze tříd balíku „context“.

```
private SensorManager manager;
private Sensor sensor;
...
manager=(SensorManager)context.getSystemService(Context.SENSOR_SERVICE);
    if (manager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){
        //Success! There is a magnetometer.
    sensor = manager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
    } else {
        //Failure! No magnetometer.
    }
```

Příklad kódu 8: Konvenční přístup k magnetometru

```
FactoryContext fc;
MagneticFieldContext magneticContext;
...
fc = new FactoryContext(context, this);
magneticContext = fc.getMagneticFieldContext();
```

Příklad kódu 9: AUIF framework přístup k magnetometru

Kritérium	AUIF framework	Konvenční přístup
Počet řádků kódu	4	8
Čitelný kód	ano	ne
Opětovné využití	ano	ano
Jednodušší možnost rozšíření	ano	ne

Tabulka 6.1. Porovnání dvou přístupu: Přistup k senzorům

6.2 Framework vs Konvenční přístup: Posluchač senzorů

V konvenčním přístupu se implementuje interface SensorEventListner, který umožňuje získávat a obnovovat kontext senzorů v metodě *onSensorChanged(SensorEvent event)* . Ve frameworku má stejný účel interface Callback s metodou *callbackValue()*.

Rozdílem je to, že konvenční strategie v metodě onSensorChanged je parametr event, pomocí kterého lze nalézt příslušný kontext. Ve frameworku se k tomu používá už konkrétní, vybraný senzor, který byl dříve zadefinován využitím FactoryContext.

Z event.values[0] není vidět, z jakého senzoru se čte kontext, a co to vlastně je za kontext. Postup ve frameworku je více přehledný díky názvům kontextů. Zároveň využitím frameworku nikdy nevznikne chyba *ArrayIndexOutOfBoundsException*, která může se objevit kvůli event.values poli.

Kritérium	AUIF framework	Konvenční přístup
Počet řádků kódu	13	21
Čitelný kód	ano	ne
Opětovné využití	ano	ne
Jednodušší možnost rozšíření	ano	ne

Tabulka 6.2. Porovnání dvou přístupu: Posluchač senzorů

6.3 Framework vs. Konvenční přístup: @Country

Výhodou AUIF frameworku je to, že počet řádků, které by měl naimplementovat vývojář je snížen téměř 10-krát. Programátor se nemusí zabývat tím, jakým způsobem je nalezen stát, potřebuje vědět jen anotaci @Country, která slouží k nalezení státu, a musí umět spustit framework. To je mnohem jednodušší, než psát tuto funkci od začátku. Data kontextu jsou zapouzdřena, vývojář nemusí přímo volat GPSContext, o volání kontextu se postará AUIFBuilder, jakmile programátor spustí aplikaci. Hlavní je, aby vývojář nezapomněl přidat třídu, kde se vyskytuje anotace @Country do AUIFBuilderu.

Nevýhoda je v tom, že po spuštění frameworku nelze získat stát přímo v *onCreate()* metodě. Po spuštění má stát hodnotu null, a je nutné počkat dobu po kterou gps získává šířku a délku polohy umístění zařízení, Geocoder zkonvertuje hodnoty do názvu státu a ten název se uloží do String proměnné. Jinými slovy, pokud programátor zavolá proměnnou stát v *onCreate()* metodě hned po spuštění, tak bude v String proměnné ležet null. V době, kdy se null hodnota uloží do proměnné typu TextView, vznikne chyba *NullPointerException*.

Kritérium	AUIF framework	Konvenční přístup
Počet řádků kódu	9	min 83
Čitelný kód	ano	ne
Opětovné využití	ano	ne
Jednodušší možnost rozšíření	ano	ne

Tabulka 6.3. Porovnání dvou přístupu: @Country

6.4 Framework vs. Konvenční přístup: @Address

Postupy v této metodě jsou stejné jako v Kapitole 6.3.

Kritérium	AUIF framework	Konvenční přístup
Počet řádků kódu	9	min 83
Čitelný kód	ano	ne
Opětovné využití	ano	ne
Jednodušší možnost rozšíření	ano	ne

Tabulka 6.4. Porovnání dvou přístupu: @Address

6.5 Framework vs Konvenční přístup: @AutoColor

Zmenšení počtu řádků kódu jednoznačně vytváří více čitelný kód. Například pokud programátor bude chtít najít, při jakých hodnotách osvětlení se nastavují barvy a jaké jsou tyto barvy, musí hledat tyto informace v minimálně 139 řádcích. I pokud má programátor vyhledávač, srozumitelně nastavené parametry metod, má průhledné dobré názvy, musí ztrácet čas, aby našel informace o barvách v kódu. Nastavení anotace s parametry nevyžaduje žádný čas na hledání. Hned je vidět, že v noci je nastavena barva X, ráno barva Y a jakého jsou typu tato barvy: klasického, rgb nebo hexadecimální.

Nevýhodou je to, že neposkytuje možnost programátorovi nastavit i lux hodnoty. Uvnitř frameworku je napsáno, že 30 luxů je střední hodnota, míně než 5 je tma, víc než 30 je hodně světla. Pokud programátor bude chtít změnit 30 luxů například na 20 luxů, tak takovou možnost nemá. Je možné to udělat v logice AUIFBuilderu, ale na to je potřeba čas, aby našel tyto nastavení a změnil hodnoty.

Kritérium	AUIF framework	Konvenční přístup
Počet řádků kódu	9	139
Čitelný kód	ano	ne
Opětovné využití	ano	ne
Jednodušší možnost rozšíření	ano	ne

Tabulka 6.5. Porovnání dvou přístupu: @AutoColor

6.6 Framework vs. Konvenční přístup: @BackgroundAutoColor

Anotaci @BackgroundAutoColor vývojář využije pro možnost změnit barvu pozadí aplikace. Počet potřebných řádků kódu je značně snížen, díky tomu, že stačí využit anotace @BackgroundAutoColor, napsat setter a spustit framework. Ten postup šetří čas programátora při vývoji dané funkčnosti. Nevýhodou je to, že ve většině ostatních anotací se využívají objekty TextView, které se budou měnit. V případě této anotace, programátor musí využít instance třídy Window, která odpovídá za změnu pozadí, což může způsobit potíže programátorovi.

Kritérium	AUIF framework	Konvenční přístup
Počet řádků kódu	9	139
Čitelný kód	ano	ne
Opětovné využití	ano	ne
Jednodušší možnost rozšíření	ano	ne

Tabulka 6.6. Porovnání dvou přístupu: @BackgroundAutoColor

6.7 Framework vs. Konvenční přístup: @TextSize

Hlavní výhodou AUIF frameworku při implementování této funkci oproti konvenčnímu přístupu je výrazné sníženi počtu řádků kodu. Při využití frameworku programátor potřebuje vědět jen název anotaci odpovídající za zvětšeni písmen textu. Při změně rychlosti objekty TextView začnou automatické měnit velikost písmen. Programátor může také měnit barvy prvků v závislosti na rychlosti pohybu uživatele. Parametry se ukazují v atributech anotaci @TextSize. Při implementaci této funkce konvenčním postupem, programátor potřebuje navrhnout kód na získaní gps souřadnic, spočítat si měnicí rychlost, zadefinovat změnu barev a velikostí. Celý tento proces zpomalí vývoj.

Kritérium	AUIF framework	Konvenční přístup
Počet řádků kódu	10	195
Čitelný kód	ano	ne
Opětovné využití	ano	ne
Jednodušší možnost rozšíření	ano	ne

Tabulka 6.7. Porovnání dvou přístupu: @TextSize

6.8 Framework vs. Konvenční přístup: @ShakeSound

K vykonání této funkce konvekčním způsobem vývojář potřebuje vytvořit přístup ke kontextu akcelerometru, získat kontext, najít formuli na počítání změny zrychleni, najít, jak funguje třída MediaPlayer, která odpovídá za nahrání zvuku: což je poměrně hodně informací potřebných k tomu, aby byl nahrán zvuk při zatřepání smartfonem. AUIF framework poskytuje mnohem jednodušší způsob implementace této funkcionality: přidat zvuk do raw souboru projektu, napsat anotaci a přidat tam název zvuku, stejný jako v raw souboru.

Nevýhodou je, že zvuk musí být krátký. V případě, že by došlo k opakovanému zatřepání smartfonem v době, kdy zvuk stále ještě hraje, se nic nestane, zvuk bude nahráván, dokud neskončí kompozice. Není implementována metoda, která by umožnila zastavit nahrávání zvuku.

Kritérium	AUIF framework	Konvenční přístup
Počet řádků kódu	5	90
Čitelný kód	ano	ne
Opětovné využití	ano	ne
Jednodušší možnost rozšíření	ano	ne

Tabulka 6.8. Porovnání dvou přístupu: @ShakeSound

6.9 Framework vs Konvenční přístup: Úsudek

Využití frameworku je výhodné, protože poskytuje jednoduchý princip přístupu k senzorům, vývojář se nepotřebuje zabývat tím, jak se k senzoru dostat. Stačí vědět, z jakého senzoru potřebuje získat informace a využít určitou třídu, která pomůže získat hledaný kontext, jakmile bude mít přístup k datům požádaného senzoru. Tento přístup k senzorům nesnižuje počet řádků kódu oproti konvenčnímu přístupu, ale má čitelnější názvy a předpřipravené šablony pro všechny existující senzory ve smartfonech s OS Android.

Další výhodou je, že framework poskytuje několik připravených řešení pro adaptaci uživatelského rozhraní. Například užíváním GPS senzoru je možné definovat stát, kde se nachází uživatel v této chvíli a automaticky vyplnit textové pole, které je součástí rozhraní.

Hlavní výhodou je snížení řádku kódu, kromě přístupu k senzorům. Díky využívání aspektů a modularitě, je kód čitelný a snadno rozšiřitelný. Zmenšením počtu řádků se snižuje i čas, potřebný na rozpracování kódu.

Budoucí vývojář nemusí sledovat přísná pravidla a používat pravě ty metody, které poskytuje daný framework. K rozšiřování funkčnosti vývojář potřebuje zavést nový aspekt. Pomocí snadného přístupu ke kontextu může navrhnout v logice jakoukoli zvláštní funkčnost, například spočítat zrychlení letadla nebo spočítat srdeční rytmus a cokoliv dalšího a poté generovat UI s využitím svých vlastních výpočtů.

Kapitola 7
Testování

K otestovaní frameworku byl využit smartfon Samsung GALAXY S3 s operačním systémem Android verze 4.4.4(KitKat).

7.1 Testování času spuštění aplikace

Tabulka níže ukazuje čas potřebný ke spuštění aplikace. Čas je naměřen u:

- AUIF frameworku;
- u aplikace, která nalezne stát pomoci gps konvenčním způsobem
- u aplikace, která nalezne kontext ze dvou senzorů: magnetometru a akcelerometru

Pro všechny případy byl čas změřen 10-krát, neboť se nejedná o stálou hodnotu. Pro každé spuštění se naměřená hodnota může se lišit.

Krok	AUIF Framework $[ms]$	GPS konvenční $[ms]$	Přístup k akcelerometru a magnetometru konvenční $[ms]$
1.	148	98	117
2.	199	100	147
3.	137	106	150
4.	145	94	100
5.	145	105	119
6.	186	74	96
7.	176	129	96
8.	139	95	113
9.	139	108	93
10.	135	109	93

Tabulka 7.1. Testování času spuštění aplikace

Využitím vzorce aritmetického průměru získáme střední čas spuštění:

auif framework $= (148 + 199 + ... + 139 + 135)/10 = 154,9$

gps konvenční $= (98 + 100 + ...108 + 109)/10 = 101,8$

akcelerometru a magnetometru konvenční $= (117 + 147 + ...93 + 93)/10 = 112,4$

Z těchto spočítaných dat je vidět, že střední čas na spuštění frameworku je vyšší, než v případech spuštění aplikace klasickým způsobem. Je to způsobeno tím, že AUIF framework nahrává veškerý kontext: získává kontext ze senzorů, vytváří objekty tříd Window, MediaPlayer, inicializuje mnoho různých objektů třídy TextView, zatímco ukázky aplikací spouští pouze jednu funkci: pro "GPS konvenční" je to získání názvu

státu, pro “přístup k akcelerometru a magnetometru konvenční” pouze přístup k těmto oběma senzorům. Lze předpokládat, že pokud budou pomocí konvenčního přístupu popsané všechny funkce, které zahrnuje a uskutečňuje framework, tak čas spuštění bude skoro stejný, protože jak konvenční postup tak i framework používají xml prezentace rozhraní. Pokud by framework využíval prezentace programovým způsobem, tak by čas spuštění aplikace byl značně snížen. Důvody k použití xml představy v této práci byly popsané v Kapitole 5.2. Uskutečnění funkcí, které poskytují framework, jsou založené na klasických postupech, proto není důvod si myslet, že čas spuštění frameworku bude delší.

7.2 Tabulka regresivních testů

Pojmy:

- developerClass - jakákoliv třída, která je vytvořena vývojářem. Vstupuje se do parametru builder.buildCache(developerClass) a má typ Object
- testovaná funkce - říká o funkčnostech frameworku na základě anotaci

Testovaná funkce	Vstup	Výsledek
Nalezeni státu pomocí anotace @Country TextView country;	developerClass	Do String country se uloží stát. Aplikace se testuje v České Republice, proto do country se ukládá Czech Republic.
Nalezení adresy pomoci anotace @Address public TextView address;	developerClass	Do String address se uloží adresa, která mírně se může lišit od aktuální adresy.
@AutoColor(day=„grey“) TextView tv;	developerClass	Proměnná tv při osvětlení od 5LUX do 50LUX má šedou barvu.
@AutoColor(night=„7ac5cd“) TextView tv;	developerClass	Proměnná tv při osvětlení min než 5LUX má barvu mořské zeleni
@AutoColor(morning = `rgb(122,197,205)`) TextView tv;	developerClass	Proměnná tv při osvětlení vice než 50LUX má modrou barvu.
@BackgroundAutoColor public Window window;	developerClass	Barva pozadí se mění v závislosti na osvětlení a nastavují se defaultní barvy: tmavé osvětlení - tmavě-fialová barva, střední osvětlení - žlutá barva, světlo - bílá barva.
@ShakeSound(soundName = `stick`)	developerClass	Při třesení se nahrává zvuk biče.
@Brightness(day = `0.5F`, night = `0.2F`) public Window window;	developerClass	Mění se jas displeje: kde je tmavě tak jas výrazně se zmenšuje na 0.2F, při denním světle jas displeje se zvýší, při slunečním světle nastavuje se defaultní hodnota 1F.
@TextSize TextView tv;	developerClass	Mění se velikost písmena v závislostí na rychlostí pohybu.

Tabulka 7.2. Regresivní testování

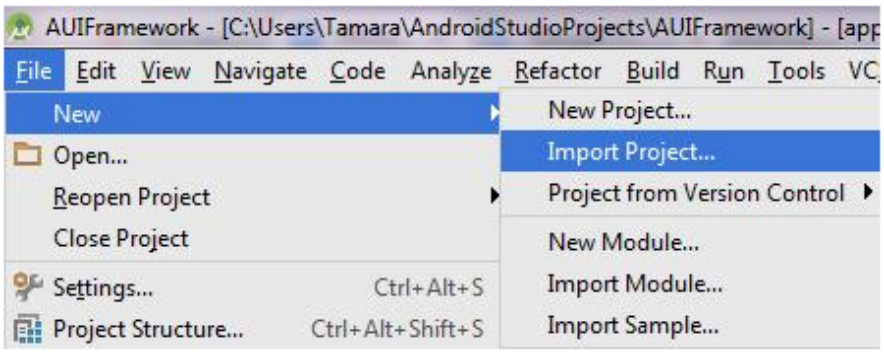
Kapitola 8
Instalace

8.1 Operační systém a Vývojové prostředí

Doporučuje se používat framework s operačním systémem Windows 7 a vývojovým prostředím Android Studio, které je založeno na IntelliJ IDEA. IntelliJ IDEA je k dispozici pro Windows, Mac OS X a Linux. Framework by taky měl fungovat na těchto OS, ale testování nebylo provedeno.

8.2 Postup instalace

1. Vložte instalační CD do CD mechaniky
2. Rozbalte AUIFramework_code.zip
3. Obsah uložte na disk
4. Otevřete Android Studio
5. Importujte projekt



Obrázek 8.1. Import projektu

Kapitola 9
Závěr

Vylepšení uživatelského rozhraní smartfonů je aktuálním úkolem, jehož řešení je důležitou podmínkou k vytvoření chytřejších telefonů, které budou pomáhat uživateli zvládat tok příchozích informací, přizpůsobí se jeho chování, náladě a okolnímu prostředí. V současné době high-tech technologií se musí telefony stávat inteligentnějšími, aby odpovídaly požadavkům rychle se vyvíjejícím trendům. Při úvahách o adaptaci uživatelského rozhraní smartfonů je jedním ze zásadních nástrojů použití senzorů, pomocí kterých je možné získat přehled o potřebách uživatele.

První část bakalářské práce je věnována posouzení problematiky. Byly prozkoumány pojmy jako je adaptivní rozhraní, kontext, aspektově-orientované programování a zároveň byly popsané senzory smartfonů a základy práce s OS Android.

Jako součást praktického výzkumu byl navrhnut framework pro adaptaci uživatelského rozhraní. Struktura frameworku byla stanovena s přihlédnutím k požadavku na využití aspektově-orientovaného principu programování. Za splněním tohoto účelu byla navrhnuta speciální forma syntaktických metadat, nazývaná anotace. Byly rozpracované následující anotace: @Country, @Address, @ShakeSound, @AutoColor, @BackgroundAutoColor, @Brightness, @TextSize, - každá z nich odpovídá za určitou adaptaci jednotlivých elementů uživatelského rozhraní.

Následující část této práce je zaměřená na testování a srovnání postupů navrhnutého frameworku s postupy, patřícími ke konvenčnímu vývoji kódu. Na rozdíl od konvenčních přístupů, framework poskytuje možnost rychlejšího vývoje díky výraznému snížení řádků kódu. Strategie využití anotací snižuje množství opakujícího se kódu. Rozdělením jednotlivých funkcí do aspektů se kód frameworku stal čitelným a snadno rozšiřitelným.

V navazující práci se předpokládá:

- implementace use-casů, které nebyly vyvinuty v rámci této práce;
- rozšíření počtů realizovaných požadavků na adaptaci uživatelského rozhraní;
- integrace nových a více složitých mechanizmů se zaměřením na fyziku procesů protékajících senzory smartfonů.

Každá z navrhnutých funkcí v této bakalářské práci byla otestována (Kapitola 7). Provedené testy potvrzují, že framework je funkční a všechny aspekty jsou úspěšně vyřešeny v souladu se stanovenými podmínkami této práce. AUIF framework je k dispozici na CD disku, který je součástí této bakalářské práce.

Literatura

[1] KREST'JANINOV, Mihail. Introduction to AOP: Paradigms of programming. In: *Habrahabr* [online]. 2011 [cit. 2016-05-18]. Dostupné z: https://habrahabr.ru/post/114649/

[2] ŠEBEK, Jiří. *Aspect-oriented user interface design for Android applications*. Prague, 2013. Master's thesis. Czech Technical University in Prague. Vedoucí práce Ing. Tomáš Černý.

[3] HODAKOV, Viktor. Adaptive User Interface: Problems Of Building. *Automation Electrotechnical Complexes and Systems*. 2003, 1(11), 45-57. ISBN 5-7763-8361-7.

[4] (ed.). Human Computer Interaction - brief intro. SOEGAARD, Mads, Rikke Friis DAM a John M. CARROLL. *The Encyclopedia of Human-Computer Interaction, 2nd Ed.* [online]. 2013. Denmark: The Interaction Design Foundation, 2013, s. 21-62 [cit. 2016-04-16]. ISBN 978-87-92964-00-7. Dostupné z: http://www.interaction-design.org/books/hci.html

[5] *Open Systems: DBMS* [online]. Moscow: Publishing house „Open systems", 2012(3) [cit. 2016-03-20]. ISSN 1028-7493. Dostupné z: www.osmag.ru

[6] ABOWD, Gregory D., Anind K. DEY, Peter J. BROWN, Nigel DAVIES, Mark SMITH a Pete STEGGLES. *Towards a Better Understanding of Context and Context-Awareness*. 1999. Springer-Verlag London, UK: HUC '99 Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, 1999. ISBN 3-540-66550-1. ISSN 3-540-66550-1.

[7] *Developers* [online]. Content is licensed under Creative Commons Attribution 2.5: df, df [cit. 2016-04-24]. Dostupné z: http://developer.android.com

[8] What is Accelerometer and how does it work on smartphones: [image]. *Techulator* [online], 2013 [cit. 2016-05-09]. Dostupné z: http://www.techulator.com/resources/8930-How-does-smart-phone-accelerometer-work.aspx

[9] VOROPAJ, Evgenij. The accelerometer in the phone: the principle of functioning. In: *GeekNose* [online], 2015 [cit. 2016-05-18]. Dostupné z: http://geek-nose.com/

[10] *Google Play* [online]. [cit. 2016-05-18]. Dostupné z: https://play.google.com/

[11] What is gyroscope? In: *MedicalPlanet* [online]. MedicalPlanet [cit. 2016-05-18]. Dostupné z: http://medicalplanet.su/telemedicina/geroscop.html

[12] Basic Visual: What is Light? *DNP* [online]. Karlslunde: Dai Nippon Printing Co, 2016 [cit. 2016-04-17]. Dostupné z: http://www.dnp-screens.com/DNP08/Technology/Basic-Visual/What-is-light.aspx

[13] Illuminance - Recommended Light Levels. *The Engineering ToolBox*, [online], [cit. 2016-04-17]. Dostupné z: http://www.engineeringtoolbox.com/light-level-rooms-d_708.html

[14] Global Positioning System: [image]. *Nevim* [online]. nevim: nevim, nevim [cit. 2016-05-09]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/2/27/GPS24goldenSML.gif

[15] ŠKOPEK, Pavel. *Techbox: váš telefon je prošpikovaný senzory* [online]. 2013 [cit. 2016-04-17]. Dostupné z: http://mobilenet.cz/clanky/techbox-vas-telefon-je-prospikovany-senzory-12496

[16] PAVLOV, Valentin. *Analysis of usage of aspect- oriented approach in the development of software systems*. Saint Petersburg, 2003. Dostupné také z: http://www.javable.com/columns/aop/workshop/01/. Master thesis. Saint Petersburg Electrotechnical University (ETU). Vedoucí práce Zhuravlev E.A., Kir'janchikov V.A.

[17] TUREK, Tomáš. *Využití aspektově orientovaného přístupu pro tvorbu adaptivních uživatelských rozhraní*. Praha, 2014. Diplomová práce. České vysoké učení technické v Praze. Vedoucí práce Ing. Tomáš Černý.

[18] LILOLE—A Framework for Lifelong Learning from Sensor Data Streams for Predictive User Modelling. FETTER, Mirko a Tom GROSS. *Human-Centered Software Engineering*. 5th IFIP WG 13.2 International Conference, HCSE 2014. Heidelberg: Springer, 2014, 126 - 143. ISBN 978-3-662-44810-6.

[19] ČERNÝ, Tomáš, Karel ČEMUS, Michael J. DONAHOO et al. *AspectFaces* [online]. In: . [cit. 2016-05-19]. Dostupné z: www.aspectfaces.com/

[20] Šebek, J. - Richta, K.: Aspect-oriented User Interface Design for Android Applications. In DATESO 2015. Prague: MATFYSPRESS, 2015, p. 121-130. ISSN 1613-0073.

[21] KERSTEN, Mik. AOP@Work: AOP tools comparison. In: *IBM developerWorks* [online]. British Columbia: University of British Columbia, 2005 [cit. 2016-04-17]. Dostupné z: http://www.ibm.com/developerworks/java/library/j-aopwork1/

[22] *La ricetta delle app baby: [image]* [online]. [cit. 2016-05-09]. Dostupné z: http://www.apogeonline.com/webzine/2014/07/07/la-ricetta-delle-app-baby

Příloha A
Příklady kódu

- **Příklad kódu 1:** Inicializace objektu LocationManager
- **Příklad kódu 2:** Povolení v AndroidManifest.xml
- **Příklad kódu 3:** Metoda určení změny umístění v LocationListner
- **Příklad kódu 4:** Příklad použití anotace @Country
- **Příklad kódu 5:** Příklad použití anotace @AutoColor
- **Příklad kódu 6:** Příklad metody nahrání zvuku
- **Příklad kódu 7:** Příklad použití anotace @Brightness
- **Příklad kódu 8:** Konvenční přístup k magnetometru
- **Příklad kódu 9:** AUIF framework přístup k magnetometru

Příloha B
Obsah přiloženého CD

/
- `AUIFramework_tex.zip` ..... Zdrojové soubory k textu bakalářské práce
- `AUIFramework_code.zip` ..... AUIF framework
- `AUIFramework_testovani.zip` ..... Obrázky testování
- `AUIFramework_ea.zip` ..... Diagramy v Enterprase Architect
- `maximtam_obrazky2016bach.pdf` ..... Obrázky nakreslené ve Photoshopu
- `maximtam_2016bach.pdf` ..... Text bakalářské práce
- `maximtam_zadani2016bach.pdf` ..... Zadaní bakalářské práce

Příloha C
Použité zkratky

AF	Aspect Faces
AOP	Aspected-Oriented Programing (Aspektově orientované programování)
AUI	Adaptive User Interface (Adaptivní uživatelské rozhraní)
AUIF	Adaptive User Interface Framework (Framework pro adaptivní uživatelské rozhraní)
CAC	Context-Aware Computing
EV	Exposure Value (Hodnota expozice)
GPS	Global Positioning System (Globální polohovací systém)
HCI	Human Computer Interaction (Interakce člověk - počítač)
id	Identification number (Identifikační číslo)
ISO	International Standards Organization (Mezinárodní organizace pro normalizaci)
ms	milliseconds (milisekundy)
OS	Operating system (Operační systém)
PC	Personal Computer (Osobní počítač)
READ	Rich-Entity Aspect/Audit Design
Ubicomp	Ubiquitous computing (Všudypřítomná výpočetní technika)
UI	User Interface (Uživatelské rozhraní)