Bakalářská práce

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Využívání kombinací kontextových informací o uživateli pro tvorbu adaptivních mobilních aplikací se zaměřením na adaptaci textu.

Tomáš Hodek

Vedoucí práce: Ing. Jiří Šebek
Obor: Softwarové inženýrství a technologie
Zaměření: Softwarové inženýrství a technologie
Květen 2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hodek** Jméno: **Tomáš** Osobní číslo: **456948**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Využívání kombinací kontextových informací o uživateli pro tvorbu adaptivních mobilních aplikací se zaměřením na adaptaci textu

Název bakalářské práce anglicky:

Using combination of various context informations about user for making adaptive mobile applications, mostly adaptation of text.

Pokyny pro vypracování:

Při návrh adaptivní struktury aplikace je třeba brát ohled na více faktorů a uživatelských vstupů. [3] Odvětví text-miningu v kombinaci s adaptivní strukturou aplikace má velký potenciál pro lepší komunikaci softwaru s uživatelem. Také uživateli zpříjemní i obsahovou stránku aplikace.
Cílem této práce je:
1. Seznámit se s:
a. mobilními android aplikacemi [2]
b. principy text miningu [3]
c. adaptivním designem aplikace [1]
d. již vyvynutými adaptivními frameworky, jako například:
i. emotional based framework [4]
ii. senzor based [5]
iii. behavioral based [6]
2. Navrhnout framework, který umožní využít kontextové informace o uživateli (náladu, okolní světlo, polohu, ...) a následně adaptovat UI aplikace. Adaptace bude ve formě sémantické změny textu.
3. Naimplementovat framework a jednoduchou ukázku jeho využití
4. Otestovat framework a analyzovat výhody a nevýhody jeho použití při implementaci adaptivních aplikací s ním a bez něj

Seznam doporučené literatury:

1. Šebek, J. - Richta, K.: Usage of Aspect-Oriented programming in Adaptive Application Structure. New Trends in Databases and Information Systems: ADBIS 2016 Short Papers and Workshops, BigDap, DCSA, DC, Prague, Czech Republic, August 28-31, 2016, Proceedings
2. Oficiální dokumentace [online]: https://developer.android.com
3. COHEN, K. Bretonnel; HUNTER, Lawrence. Getting started in text mining. PLoS computational biology, 2008, 4.1: e20.
4. LUNOVA, Anastasiia. Adaptace UI založena na emocích uživatele. Praha, 2017. Bakalářská práce. ČVUT v Praze, Fakulta elektrotechnická.
5. TITOVA, Tamara. Využiti aspektově orientovaného přístupu a lokálních dat ve vývoji adaptivního uživatelského rozhraní pro Android. Praha, 2016. Bakalářská práce. ČVUT v Praze, Fakulta elektrotechnická.
6. MISHCHENKO, Nikita. Aspektově orientovaný vývoj adaptivní struktury aplikace pro Java EE aplikace. Praha, 2016. Bakalářská práce. ČVUT v Praze, Fakulta elektrotechnická

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek, laboratoř inteligentního testování softwaru FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **09.02.2018** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce: **30.09.2019**

Ing. Jiří Šebek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Poděkování bych chtěl věnovat především mému vedoucímu práce panu Ing. Jiřímu Šebkovi, ale také spolužákům, za občasné konzultace a cenné rady.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 23. May 2018

Abstrakt

Tato práce se zabývá návrhem frameworku pro adaptivní mobilní aplikace, využívající text-mining a kontextové informace uživatele. V práci je kladen důraz na kombinaci text-miningu a kontextových informací a na jejich vhodné použití ve frameworku. Ten je navržen tak, aby byl pohodlně použitelný pro vývojáře Android aplikací.

Klíčová slova: text-mining, adaptivní uživatelské rozhraní, uživatelský kontext, android, mobilní aplikace

Vedoucí práce: Ing. Jiří Šebek

Abstract

This thesis contains design of a framework for adaptive mobile aplications, using text-mining and context informations. There is more focus in combination of text-mining and context information, and their proper usage in framework. The framework is designed to be comfortable for developers of Android apps to use.

Keywords: text-mining, adaptive user interface,user context, android, mobile application

Title translation: Using combination of various context informations about user for making adaptive mobile applications, mostly adaptation of text.

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Tato bakalářská práce práce obsahuje řešení pro problematiku adaptivních mobilních aplikací. Tím je framework využívající kontextové informace uživatele a text-mining pro adaptaci mobilní aplikace. Text-mining je využit pro získání kontextových informací textů v aplikaci. Oba tyto kontexty, uživatelský i textový jsou následně ve frameworku využity k adaptaci textu a uživatelského rozhraní.

Celý framework se dělí na dvě částí. Jednou z nich je část s názvem definovaná adaptace a druhou částí je automatická adaptace. Definovaná adaptace využívá kontextové informace uživatele pro adaptaci textových prvků v aplikaci s použitím programátorem předdefinovaných textů. Automatická adaptace využívá kontextové informace získané z textu pro adaptaci uživatelského rozhraní a také k adaptaci textů.

Práce obsahuje rešerši dané problematiky, ze které se vychází v ostatních částech práce. Rešerše obsahuje především jednotlivé podproblémy celkového řešeného problému práce, ze kterých se celková práce skládá.

Obě výše zmíněné části podlehly důkladné analýze i návrhu. Jak analýza, tak návrh nejprve obsahují celkový pohled na problém a jeho rozbor. Následně se obě dvě části dělí na analýzu definované adaptace, analýzu automatické adaptace a obdobné dělení je rovněž u návrhu.

Implementace je rovněž částečně rozdělena a popisuje implementační metodiku obou částí zvlášť. Naproti tomu testování proběhlo jako jeden celek. Testování proběhlo nad celou touhle prací, tedy otestován byl hlavně celkový návrh frameworku pomocí testů s uživateli.

Práce je svým návrhem a implementací zaměřena na mobilní telefony s operačním systémem Android, to především kvůli faktu, že je Android OS nejpoužívanějším operačním systémem[2] a rovněž jsem majitelem zařízení s Android OS a tento systém je mi blízký, tak jako programovací jazyk Java, ve kterém je framework napsán.

1.1 Motivace

V posledních letech se s vývojem mobilních technologií klade stále větší důraz na tvorbu uživatelského rozhraní a na použitelnost mobilního software. Kvůli stále se rozšiřující konkurenci a s tím i zvyšujícím se nárokům uživatelů

jsou vývojáři software nuceni více myslet na pohodlí a spokojenost právě cílových uživatelů. Mnohdy v těchto případech už není dostačující responzivní a kvalitně zpracované uživatelské rozhraní, ale je potřeba vytvořit rozhraní, které se dokáže přizpůsobit přímo cílovému uživateli tak, aby bylo jako na míru ušité právě pro něj. To je právě účelem adaptivního uživatelského rozhraní, adaptovat rozhraní tak, aby bylo šité na míru každému uživateli a bylo jim pohodlné ho používat a hlavně, aby ho chtěli používat. Komplikací při návrhu adaptivních uživatelských rozhraní je zdroj informací pro adaptace. Nejlepším zdrojem je samotný uživatel, neboli kontextové informace uživatele[1]. Ty nám díky dnešním mobilním technologiím a senzorům dokáží poskytnout poměrně obsáhlé informace k tomu, aby jsme byli schopni adaptovat rozhraní k potřebám uživatele.

V dnešní době se již můžeme setkávat se spoustou adaptivních rozhraní, samotné mobilní operační systémy jsou adaptivním rozhraním už jen tím, že dokáží adaptivně upravovat jas obrazovky na základě okolního osvětlení. Dalším případem může být například chování navigace, když uživatel vjede to tunelu (navigace většinou ztmavne a zobrazí upozornění, že ztrácí GPS signál), což je rovněž reakce na uživatelův kontext.

Další oblastí, která se stále v poslední době vyvíjí kupředu je oblast text-miningu. Díky text-miningu jsme schopni analyzovat nestrukturovaný text a získat z něj informace. V mobilních aplikacích nám text-mining může poskytnout kontextové informace o aplikaci samotné, nebo spíše o jejím obsahu. To mohou být v některých aplikacích velmi důležité a užiteční informace, který by mohli poskytnout nové možnosti právě pro již zmíněné adaptivní uživatelské rozhraní. Velký potenciál může mít i spojení kontextových informací o aplikaci a kontextových informacích uživatele, díky čemuž by jsme byli schopni analyzovat kompletní kontext celé situace a adaptovat rozhraní přesně podle něj, díky čemuž by mohlo být uživateli poskytnuto unikátní rozhraní, které by mu nejvíce vyhovovalo.

Adaptivní uživatelské rozhraní, sběr uživatelských kontextových informací a text-mining jsou poměrně komplexní oblasti, které pro zlepšení mohou využívat další oblasti, jakými může být například machine-learning, uměla inteligence a tak dále. Jsou to ale velmi silné nástroje, kterými lze při dobré analýze a odhodlání dosáhnout ideálních výsledků v adaptivních mobilních aplikacích.

1.2 Cíle

Cílem této bakalářské práce je navržení frameworku pro mobilní adaptivní aplikace. Framework by měl využívat kontextové informace, o uživateli i o aplikaci, a také oblast text miningu pro adaptovaní uživatelského rozhraní mobilní aplikace. Práce bude obsahovat analýzu problému a také rešerši jednotlivých oblastí a technologií. Z důvodu malého výskytu již existujících řešení, bude potřeba při návrhu a analýze improvizovat a vycházet spíše z jednotlivých oddělených řešení, jako je adaptace uživatelského rozhraní, kontextové informace a nebo text-mining sám jako takový.

Při návrhu frameworku bude kladen důraz na použitelnost a na jednoduchost použití pro programátora. Framework bude mít za cíl pracovat co nejvíce samostatně, aby ušetřil programátorovi složité nastavování s inicializací frameworku. Zároveň však musí poskytnout programátorovi kontrolu nad fungováním frameworku, aby byl schopen například nastavit míru adaptace a jednotlivé případy použití adaptace.

Framework bude disponovat možnostmi adaptovat definovaný text, získat a zpracovat uživatelův kontext, rovněž kontext aplikace nebo aktivity za použití oblasti text-miningu. Tuto funkcionalitu by měl framework využít nejlépe kombinovaně, adaptovat text na základě uživatelova kontextu, nebo adaptovat uživatelské rozhraní vzhledem ke kontextu aplikace, případně aktivity. Tyto funkčnosti budou podrobně popsány v návrhu frameowrku.

Práce bude rovněž obsahovat implementační část, která bude spíše prototypem k návrhu frameworku, a která bude demonstrovat styl chování a struktury frameworku. Celý návrh a především implementace budou určeny pro platformy android a implementace bude programována v Javě. Využity budou také jiné frameowrky pro Javu nebo Android.

Kapitola 2

Rešerše

2.1 Uživatelské rozhraní a adaptivní uživatelské rozhraní

Uživatelské rozhraní, nebo pouze ve zkratce UI, představuje způsob komunikace člověka s daným systémem. Na UI se klade čím dál větší váha a v posledních letech je vývoj dobrého UI jedním ze stěžejních součástí vývoje všech systému se kterými lidé pracují[4, 3].

Hlavním cílem dobrého UI je umožnit uživateli vykonávat potřebné akce rychle a efektivně. Zároveň se klade důraz na to, aby bylo UI co nejvíce intuitivní a uživatelé se museli co nejméně učit používat systém. Dalšími důležitými prvky při vývoji UI jsou hedonické, emoční, prožitkové a estetické aspekty, které jsou pro dnešního uživatele rozhodující při výběru veškerých systémů[5].

V případě této práce se budeme zaměřovat pouze na jeden typ UI a tím je GUI, neboli graphic user interface.

UI se ještě může dělit na:

- Statické UI.

  UI se nemění, zůstává vždy stejné.

- Dynamické UI (adaptivní).

  UI se mění v průběhu používání nebo chodu aplikace. Může reagovat na změny kontextu nebo nastavení uživatele.

2.2 Adaptivní UI

Adaptivní user interface nebo také AUI je dynamické UI, které dokáže měnit svoji podobu, strukturu nebo obsah. Tyto změny mohou probíhat na základně několika faktorů. Jedním z nich může být kontext uživatele. Například změna jasu při velkém osvětlení okolí uživatelova prostředí už je dynamické UI. Dalším faktorem může být obsah aplikace, který se mění za běhu. Toto chování si lze představit například na aplikaci s předpovědí počasí, který mění své UI na základě předpovědi, kterou dostala.

Velkou výhodou adaptivního UI je možnost přizpůsobení jedné aplikace hned několika uživatelům. UI se v tom případě dokáže chovat podle předem definovaných skupin uživatelů a každé skupině se zobrazit jinak.

Nevýhodou adaptivního UI je časová náročnost a pracnost jeho implementace. Jelikož je potřeba definovat veškeré případy a změny v UI, které chceme pokrýt, tak je náročnost až několikanásobně vyšší.

2.3 Kontextové informace uživatele

Uživatelskou kontextovou informací se rozumí každá relevantní informace, která se týká uživatelova prostředí a situace. Existuje několik důležitých aspektů ze kterých se uživatelův kontext skládá a některými z nich jsou například: poloha uživatele, kdo se nachází s uživatelem, zdroje informací poblíž uživatele, prostředí v okolí, a tak dále. Už mezi konkrétní informace se řadí více elementární prvky, jako například hluk, vlhkost, osvětlení, konkrétní poloha, nálada uživatele, teplota a další[7, 8].

V případě mobilních aplikací nebo spíše aplikací jakýchkoli se do kontextu uživatele přidávají další informace. Tyto informace bývají většinou uživatelem neovlivnitelné a také málo variabilní. Mezi ně patří například hardwarové informace o zařízení používané uživatelem, rychlost a stabilita připojení k internetu nebo třeba verze nainstalovaného software.

Největší komplikací při zjišťování a používání kontextových informací je určení formátu, který použijeme pro jejich reprezentaci. Z důvodu vysoké komplexity a složitosti případných kontextových informací je potřeba mít robustní formát, kterému porozumí jak lidé, tak i stroje. Kvůli tomu se často používají následující požadavky pro reprezentaci kontextu. Formát by tedy měl být[7]:

- Strukturovaný

  Kontextové informace můžou obsahovat velké množství dat, které je potřeba efektivně třídit a prohledávat. Správné strukturování toto zjednodušuje a také zjednodušuje i určení pojmenování jednotlivých atributů.

- Zaměnitelný

  Kontext musí být zaměnitelný mezi jednotlivými komponenty systému (mobilní zařízení, portál,...). To ale vyžaduje serializovatelnou reprezentaci dat.

- Composable/Decomposable

  Je třeba mít možnost distribuované správy na kontextem.

- Jednotný

  Pro všechny typy kontextových dat.

- Rozšiřitelný

  S rostoucími možnostmi je nutnost rozšiřovat sbírané informace.

- Standardizovaný

  Kontextové informace je často potřeba posílat mezi různými systémy a zařízeními, které nemusí být spravovány stejnou doménou. Proto je třeba posílat data ve standardizované podobě.

Mezi existující řešení pro formáty kontextových informací nebo kontextových profilů, jak uvádí autoři těchto formátů, patří například, Composite Capability/Preference Profiles language (CC/PP), IETF Media Feature Set nebo novější Comprehensive Structured Context Profiles (CSCP). Kromě IETF Media Feature Set jsou veškeré formáty založené na Resource Description Framewrok (RDF), standartizovaný framework založený na XML meta datech.

2.3.1 RDF

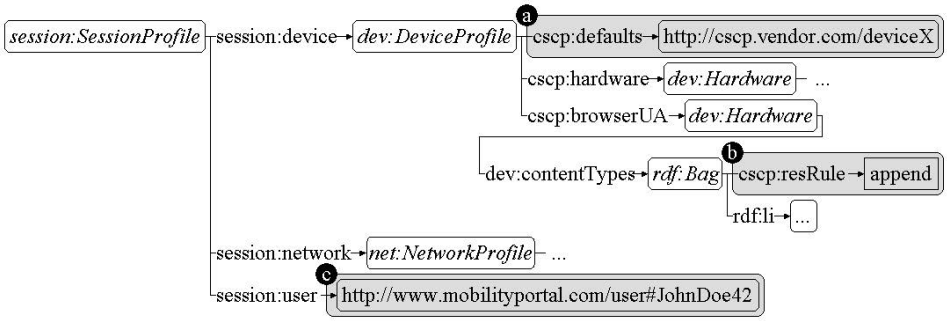
RDF je považován za standardní model pro předávání informací na webu. Je designován tak, aby byl čitelný počítači a také aby usnadnil slučování dat i když mají rozdílné zásadní schéma. RDF používá XML a je součástí webové sémantické aktivity společnosti W3C[9].

2.3.2 CC/PP

CC/PP definuje základní strukturu pro kontextové profily. Kontextový profil je v tomto případě popsán striktní dvou úrovňovou hierarchií. Tak, že každá úroveň obsahuje komponenty a ty poté obsahují jednotlivé atributy. Slovníky pro tuto strukturu by měli být definovány pomocí RDF, tím se zabezpečí případná rozšiřitelnost. Nevýhodou CC/PP je fakt, že většina zařízení nemá pouze dvou úrovňovou hierarchii, což může být trochu komplikace při vytváření kontextu. Dále CC/PP požaduje jednoznačné pojmenování pro každý atribut i když se nachází v jiné komponentě. Tedy pokud 2 komponenty mají stejný atribut, nemůže se v obou jmenovat stejně. CC/PP ale i přes to splňuje většinu požadavků zmíněných výše. Jedinou výjimkou může být struktura, která je zde pevně daná[7].

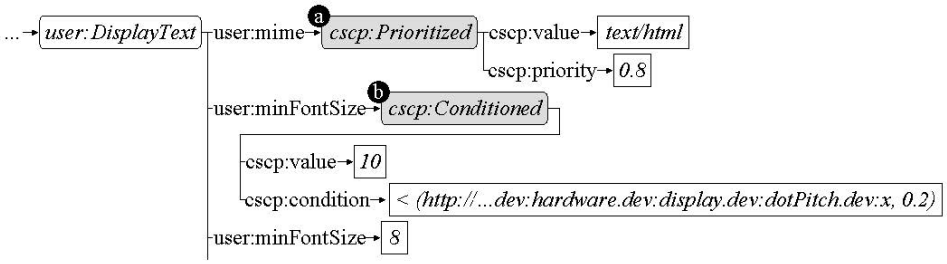
2.3.3 CSCP

CSCP vyjadřuje kontextové informace pomocí relačních profilů. Jednotlivé profily popisují veškeré informace relevantní k uživatelské mobilní relaci. Stejně jako CC/PP je CSCP založeno na RDF, ale navíc poskytuje možnost strukturalizace, která zde není pevně daná. Podporuje tedy plnou flexibilitu RDF. CSCP také umožňuje slučování jednotlivých profilových podstromů s jejich odpovídajícími defaultními podstromy. Chování podstromů a jednotlivých kontejnerů je určeno meta atributem cscp:resRule, který může mít hodnoty merge a override u podstromů, a u kontejnerů hodnoty override, append, prepend, intersection a difference[10].



Obrázek 2.1: Příklad CSCP v RDF notaci (převzato z [10])

Další výhodou CSCP je možnost přidání priorit a podmínek k jednotlivým atributům. Ty se přidávají především z důvodů řešení případných konfliktů. Priorita může nabývat hodnot 0 - 1 a defaultně je nastavena na 0,5. Pro podmínky jsou porovnávána numerická, slovní a množinová porovnání s boelovským výsledkem. Za pomocí vícero podmínek lze na atributu dosáhnout například if-then-elsif-else výrazu[10].



Obrázek 2.2: Příklad CSCP podmínek a vlastností v RDF notaci (převzato z [10])

Jednotlivé profily CSCP jsou dotazovány pomocí atributů jednotlivých entit v profilech. Ty jsou referencovány jako CSCP path expressions.

2.4 Text-mining

Text mining, nebo také text data mining, převážně znamená automatickou extrakci zajímavých, relevantních a netriviálních částí informací z nestrukturovaného textu. Na text-mining může být nahlíženo jako na rozšíření data miningu s tím rozdílem, že data mining získává informace ze strukturovaných dat, zatímco text-mining z nestrukturovaných. Text-mining zahrnuje hned několik dalších disciplín s ním spojených, jako jsou získávání informací, textová analýza, clustering, categorizaci, vizualizaci, machine learning nebo i samotný data mining[11].

Díky výše zmíněné absenci strukturalizace dat se jedná o mnohem více komplexnější úkol při získávání informací. To především z toho důvodu, že zatím neexistuje způsob, jak bychom mohli reprezentovat nestrukturovaný text počítačům. Text je totiž napsán tak, aby šel číst lidmi, tím pádem bychom

museli naučit počítače "číst" a chápat souvislý text, aby byl text-mining stejně dobrý jako lidské čtení. Spousta vědců tvrdí, že je zapotřebí úplné simulace toho, jak naše mysl funguje, abychom byli schopni napsat programy, které budou "číst" jako my. Doposud se nám daří získávat pouze malé fráze z textů, které ve výsledku shrnou obsah celého textu[12].

Důležitým aspektem text-miningu je získání doposud neznámé informace, čímž se výrazně liší například od vyhledávacích služeb.

Nejvíce se text-mining používá v oborech biologických věd, kde je zapotřebí zpracovávat obrovské množství publikací a knih k získání užitečných informací.

Je několik základních postupů při používání text-miningu. V biologických oborech se mluví o třech základních[13].

Prvním je hledání společného výskytu. Pokud se nějaká dvě klíčová slova vyskytují v jedné části nebo větě, vytvoří se mezi nimi vazba. Tento přístup však není příliš úspěšný a je spojován s velkou nepřesností, protože takto může spojit i nerelevantní data. Dokonce některými odborníky není ani považován za obor text-miningu.

Další přístup je založený na pravidlech nebo vědění o předem nadefinovaných vazbách. Tyto vazby jsou dopředu známy vědci a nadefinovány tak, aby omezily nerelevantní výsledky analýzy. Jako příklad lze uvést následující definice: <gen> je vázán s <chorobou>. Další částí tohoto přístupu může být definice lingvistických a sémantických analýz, které pak pomohou vyhledat lepší vazby[13].

V kontrastu s druhým přístupem se používá třetí, který je založen na statistice nebo machine-learningu. Tento přístup staví na klasifikátorech, které pracují na jakékoliv úrovni dokumentu od labelování částí textu až po používání syntaktických stromů pro analyzování celých vět nebo dokumentů.

Problémem ve všech těchto přístupech k text-miningu a vlastně v celém text-miningu je homonymie slov[13]. Tento problém se vyskytuje ve všech jazycích. Například v anglickém jazyce se může stát, že jedno slovo může stát za sloveso i za podstatné jméno. Zároveň to v odborných článcích může být používaný výraz pro více věcí, zatímco v používané řeči to znamená něco jiného. Řešením by mohla být specifikace takovýchto slov, avšak ta by byla pravděpodobně moc časově náročná.

2.5 Data-sety

Data sety jsou obecně kolekce dat. Přesněji se jedná o strukturované kolekce dat, získatelné pomocí jednoduchého dotazu nebo linku jako jednotlivá data, jejich kombinace nebo jako celé entity[14]. Data set může reprezentovat například tabulku v databázi. Termín data set lze také použít pro kolekce úzce vázaných tabulek nebo dat, korespondující s určitým tématem[15]. Data setům, které jsou příliš velké pro práci s běžnými analytickými nástroji se jinak také říká „big data".

2.6 Analýza sentimentu (Sentiment analysis)

Analýza a opinion mining jsou oblasti zaměřující se na získávání lidských názorů, sentimentu, postojů a nálad ze psaných textů. Jedná se o velmi probírané oblasti text miningu, data miningu a také web miningu. S rozrůstáním sociálních sítí je analýza sentimentu více žádaným tématem i v oblastech managementu, marketingu a sociálních studií, kde nejvíce vypomáhá právě firmám, například při zjišťování jejich oblíbenosti na sociálních sítích. Díky tomuto nárůstů dnes existuje velké množství dat a data setů právě se zaměřením na analýzu sentimentu z různých sociálních sítí, jako je Twitter, Facebook nebo jiné[16].

Analýza sentimentu je v dnešní době velmi důležitá pro moderní business. Lidské názory jsou totiž velmi důležité pro ostatní a dokážou velmi ovlivnit chování lidí i firem. Na základě těchto analýz jsou lidé a firmy schopni upravovat reklamy, příspěvky nebo i svoje business modely na základě názorů ostatních lidí na sociálních sítích[16].

Nejrozšířenější metodou pro analýzu sentimentu je použití ručně anotovaných data setů určených pro analýzu sentimentu. Tyto data sety mohou být přímo určeny i pro konkrétní platformy, jako například pro analýzu sentimentu na Twitteru, nebo Facebooku. Nejčastěji se jedná o data sety anotované labely sentimentu, například jako „positive“, „neutral“, „negative“. Některé data sety mají i labely „mixed“, „other“ nebo „irrelevant“, například v případě anotovaných delších frází. Vyskytují se i takové, které mají numerické hodnoty, od -5 do 5, od negativního po pozitivní[17]. Problémem těchto metod je fakt, že jsou anotované pouze dané fráze, ale už ne jejich spojení nebo jen jejich části. Pouze s těmito datasety není možnost jak určit sentiment například celé věty. Je možnost ji rozdělit a zjisti sentiment slov a frází zvlášť a poté udělat průměr, ale tato metoda bude velmi nepřesná.

Příklady existujících data setů používaných pro analýzu sentimentu na Twitteru:

- Stanford Twitter Sentiment Test Set

  Obsahuje dva data sety. Jeden automaticky vytvořený, pomocí sledování emotikonů, tedy tweet[1] s „:)“ je automaticky označený jako pozitivní. Tato metoda není vždy přesná, jelikož například nemá šanci rozpoznat sarkasmus. Druhý set je anotovaný ručně z tweetů vybíraných podle obsahu jmen, produktů a firem.

- Health Care Reform (HCR)

  Manuálně vytvořený data set z tweetů obsahujících „#hcr“. Anotovaný ručně pomocí pěti labelů a rozdělený do tréninkového, developerského a testovacího setu.

- Obama-McCain Debate (OMD)

[1] Tweet je výraz pro příspěvek na Twiteru.

Data set zkonstruovaný z prezidentských debat. Jednotlivé sentimenty byly získány pomocí Amazon Mechanical Turk[2], kdy každý tweet byl označen aspoň třemi anotacemi. Tento data set je často používaný pro zjišťování polarity tweetů.

- Sentiment Strength Twitter Dataset (SS-Tweet)

  Manuálně anotovaný data set pomocí positivních a negativních anotací. U každého je ještě síla, u negativního se jedná o hodnoty -1 (není negativní) až -5 (extrémně negativní). S opačnými znaménky poté i pro pozitivní anotace.

- Sanders Twitter Dataset

  Data set nasbíraný z tweetů týkajících se 4 rozdílných témat (Apple, Goodle, Microsoft, Twitter). Anotovaný jedním člověkem pomocí anotací „positive“, „negative“, „neutral“ a „irrelevant“.

- SemEval-2013 Dataset (SemEval)

  Tento dataset byl vybudován v rámci Semantic Evaluation of Systems challenge. Data set byl manuálně anotovaný s použitím Amazon Mechanical Turk pracovníků za pomocí pozitivních, negativních a neutrálních labelů. Navíc byly přidány anotace pro subjektivitu a objektivitu tweetů.

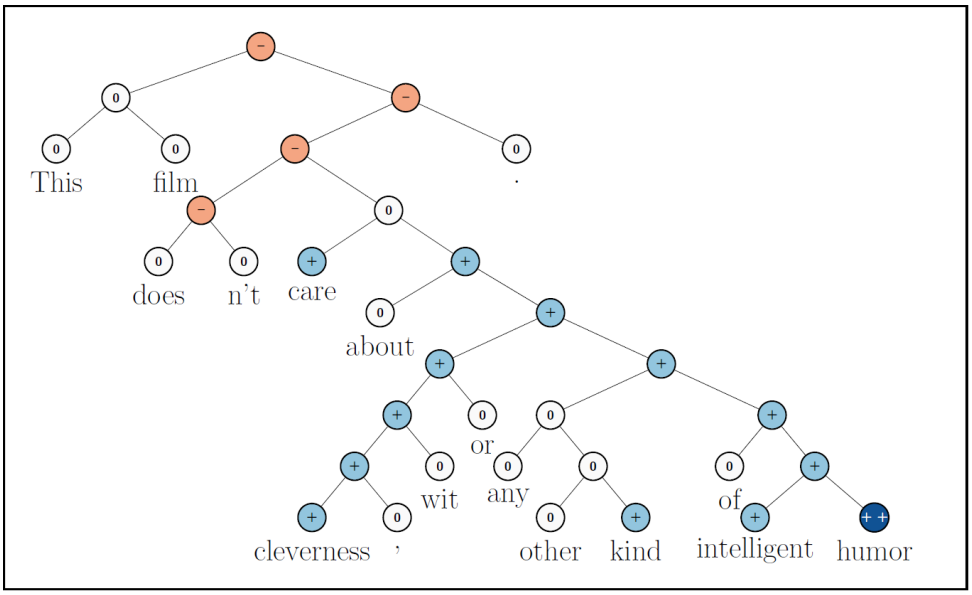
Pro použití těchto data setů je potřeba vytvořit algoritmy, které s nimi budou pracovat a správně posuzovat kombinace jednotlivých frází a slov. Velkým problémem těchto algoritmů je množství pravidel a přesných instrukcí, které musí programátoři psát. Tento problém řeší program NaSent - Neutral Analysis of Sentiment. Cílem vývojářů programu bylo právě odstranit potřebu psát pravidla a instrukce pro práci s data sety. Pro tento program existují data sety vytvořeny z 12 000 filmových hodnocení, rozdělených do 214 000 frází a vět ohodnocených intenzitou „líbí se“ a „nelíbí se“. S pomocí deep learningu[3] vytvořili algoritmus, kterého účelem je nastudovat tyto data sety a vazby v jednotlivých frázích. Přesněji vytvořit pro tyto výrazy matematické atributy, se kterými bude následně pracovat. Celý program tedy funguje pouze se dvěma vstupy, a to data setem a algoritmem, který se „naučí“ pracovat s daným data setem. Výhodou tohoto programu je, že se zaměřuje i na celé věty a dokáže i určit možnosti změn jednotlivých frází[18].

NaSent pro správné zpracování delších frází využívá Stanford Sentiment Treebank a model jemuž se říká Recursive Neutral Tensor Network (RNTN), který dokáže přesně predikovat kompoziční sémantické efekty. Stanford Sentinent Treebank obsahuje plně označené stromy, které pomohou ke kompletní analýze kompozičních efektů sentimentu. Pro získání větší přesnosti NaSent využívá model RNTN, který reprezentuje fráze pomocí vektorů slov složených do stromu, kde počítá vektor nejvyššího node ve stromu dokud nepoužívají stejnou kompoziční funkci[19].

[2]Služba umožňující najímat lidi na práci, kterou zatím nezvládnou počítače.

[3]Oblast zaměřující se na možnost počítačů porozumět věcem více z lidského hlediska.

Program NaSent je zahrnut ve frameworku Stanford CoreNLP, který se zabývá rozborem nestrukturovaného textu jako celku.

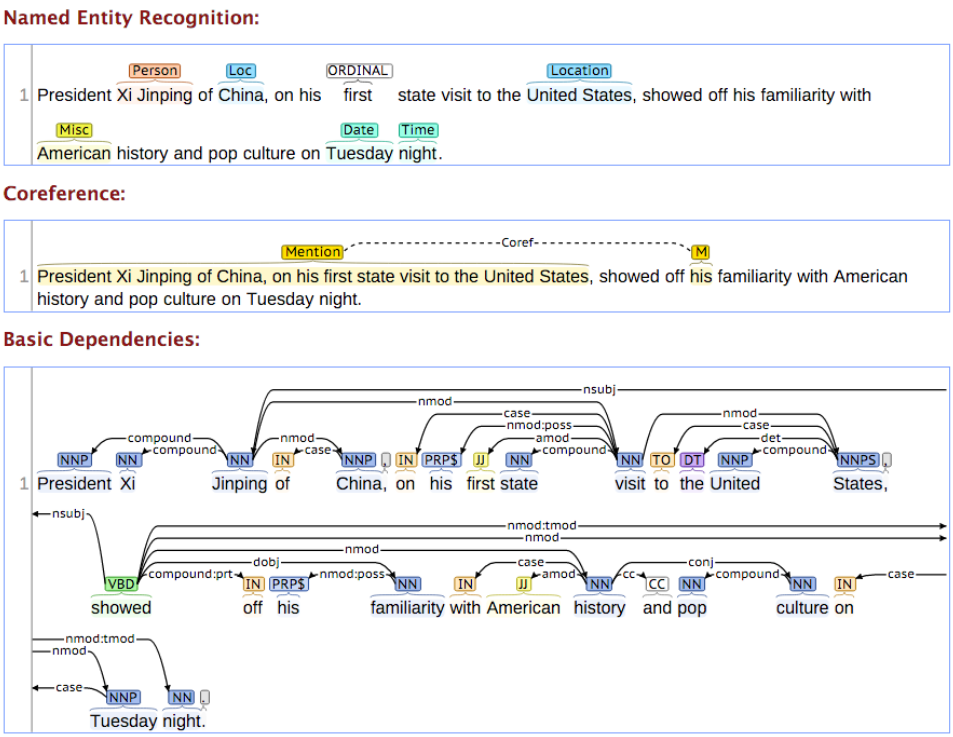


Obrázek 2.3: Příklad modelu RNTN, předovídající 5 tříd sentimentu, velmi negativní až velmi pozitivní (–, -, 0, +, ++), v každém node a zaznamenání negativity věty jako celku (převzato z [19].)

2.6.1 Stanford CoreNLP - Natural language software

Stanford CoreNLP poskytuje sadu nástrojů pro rozbor lidského jazyka. Jedná se o komplexní framework, který umí zjistit kompletní strukturu textů, formy slov a jejich význam v textu, názvy firem, jména lidí, data. Dokáže také rozpoznat strukturu vět a syntaktickou stavbu věty, podmět s přísudkem, sentiment věty (NaSent) a další[20].

Tento framework je kompatibilní s většinou dnešních programovacích jazyků, přednostně určen pro Javu 8. Lze jej použít i pouze přes příkazový řádek a nebo jako vzdálený API server. Velkou výhodou frameworku je práce s textem a možnost zjištění sentimentu z jednotlivých vět.



Obrázek 2.4: Příklad rozkladu věty pomocí Stanford CoreNLP. (převzato z [20].)

2.7 Hledání synonym

Synonyma jsou slova se stejným významem. Jsou důležitá pro náš jazyk a mohou být nápomocná pro zvýšení výkonosti a přesnosti nebo i použitelnosti mnoha aplikací. Například mohou pomoci pro zlepšení vyhledávání ve vyhledávačích, aby byly výsledky více generalizované. Dalším použitím mohou být překlady, generování přirozených textů, v komunikačních aplikacích a v neposlední řadě v text minigu a adaptaci textu. Pro potřeby hledání synonym existují různé obsáhlé data sety, nebo jinak nazývané statické listy synonym. Tyto data sety ale nestačí díky rapidně rozvíjejícím se jazykům, novým produktům a názvům. Proto je snaha přijít s co nejlepšími automatickými algoritmy pro vyhledávání a generování synonym automaticky, například z webu. Takovéto algoritmy a programy by měli splňovat určité podmínky pro generovaná synonyma. Ta by neměla měnit kontext, neměla by změnit názvy (CTU - Czech Technical University nebo Counter Terrorist Unit) a také by neměla být zastaralá (dříve byl např. jen jeden PlayStation, dnes je třeba je rozlišovat)[22].

Na vyhledávání synonym lze použít několik metod. Jednou je vyhledávání založené na slovnících. To má však velké omezení především díky obsahu slovníků. Další je použití metod založených na vyhledávání ve webových prohlížečích. Jedním z nich je například PMI-IR, Pointwise mutual information - information retrieval, který používá vylepšené vyhledávání přes fulltextový vyhledávač, nebo algoritmus Synonyms discovery from full-text search log[21].

Další metodou jsou také algoritmy založeny na korpusech (corpora based algorithms). Ty pracují na základě velkého množství textů, ze kterého jsou schopny získat synonyma. Myšlenkou je fakt, že synonyma jsou používána nejčastěji na stejných místech a podobným způsobem[22]. V neposlední řadě lze použít jíž existující vyhledávače synonym jako je například Thesaurus[24]. Na těchto vyhledávačích bývá většinou dostupné WEB API pro přístup z aplikací. Avšak jejich nevýhodou je, že hledají synonyma pouze slov nebo krátkých frází a tak neexistuje záruka, že by se nezměnil význam věty.

Kapitola 3

Analýza

Cílem této práce je navrhnout framework pro adaptivní mobilní aplikace. Fremwork by měl využívat oblasti text-miningu a také kontextoxtové informace pro vytvoření adaptace. Nejvhodnějším řešením by bylo využití obou oblastí zároveň, tak aby navzájem poskytovali více rozšířené možnosti, než každá zvlášť. Při návrhu frameworku se pokusím zaměřit na použití obou oblastí zároveň.

Ze strany kontextových informací je velké množství možností, jak přistupovat k adaptivnímu uživatelskému rozhraní. Většina již existujících řešení adaptuje především menu a jeho rozložení, nebo odstíny barev v aplikaci, případně dialogové okna nebo upozornění. Nadruhou stranu text-mining je využíván v mobilních aplikacích velmi málo a nebylo ani nalezeno rozumné podobné nebo použitelné řešení, kde by se vyskytoval text-mining v mobilních aplikacích. Díky tomu ani řešeních, která by používala obě tyto oblasti je velmi málo a rovněž nebylo nalezeno žádné vhodné pro porovnání s touto prací. Proto budu při návrhu tohoto frameworku vyházet především z oddělených řešení a z problematiky popsané v rešerši. Výsledné řešení proto nebude téměř ničím inspirované a bude se tak jednat o vlastní návrh.

Framework by měl být lehce použitelný pro jakéhokoliv programátora a neměl by mu přidělávat žádné psaní kódu navíc. Zároveň by však měl mít programátor určitou kontrolu nad tím, jak bude použita adaptace aplikace. Bude tedy potřeba najít nějaký kompromis v tom, co bude nutné pro programátora a jaké bude mít možnosti měnit chování frameworku. Zároveň framework musí splňovat některé požadavky a vlastnosti. Měl by být jednoduše rozšiřitelný, měl by mít možnosti přidání a úprav kontextových informací a také úpravy možnosti adaptace, které by měl mít možnost částečně přizpůsobovat i programátor sám. Co se kontextových informací týče, framework bude potřebovat určitě knihovny třetích stran, proto tedy moduly pro získání kontextových informací musí být dobře škálovatelné.

Na základě debat z vedoucím práce a vlastního průzkumu jsem přišel se dvěma částmi, na které se bude framework dělit. Jednou částí umožní framework zcela plnou kontrolu nad adaptací. Ten si zvolí co, jak a kdy se bude adaptovat. Druhá část bude více automatická a programátor nastaví pouze míru, druh a scope adaptace. Tyto dvě řešení pokryjí jak sběr kontextových informací, tak částečně oblast text-miningu. První řešení se rovněž zaměří

pouze na adaptaci textu, zatímco druhé bude více globální a bude mít více možností adaptace jak textu, tak uživatelského rozhraní. V této práci budou tyto dvě části frameworku nazývány jako Definovaná adaptace a Automatická adaptace.

3.1 Část definované adaptace

V části definované adaptace bude mít programátor téměř úplnou kontrolu nad chováním frameworku. Nejlépe tím způsobem, že si programátor nakonfiguruje jeden konfigurační soubor a o zbytek se framework postará. Konfigurační soubor ale zároveň definuje celé chování frameworku.

Velký výskyt textů v mobilních aplikacích je dnes zcela běžný. V jistých případech mají zájem autoři aplikací ale zobrazovat a měnit některé texty. To tak, aby zrovna co nejlépe vyhovovali uživateli. Může se tak jednat o například o zkrácení textů, které nemusí být zcela relevantní nebo pouze zestručnění nějakého popisku, návodu nebo poznámky v aplikaci. Uživateli může takováto změna zjednodušit práci, nebo naopak, pokud by se text zobrazil podrobnější, uživatel si může být jistější v používání aplikace a vyvarovat se tak chybám. Rozumným řešením pro tuto problematiku je využití kontextových informací uživatele. Na základě těchto informací poté adaptovat text přesně podle jeho potřeb, tak aby mu naprosto vyhovoval.

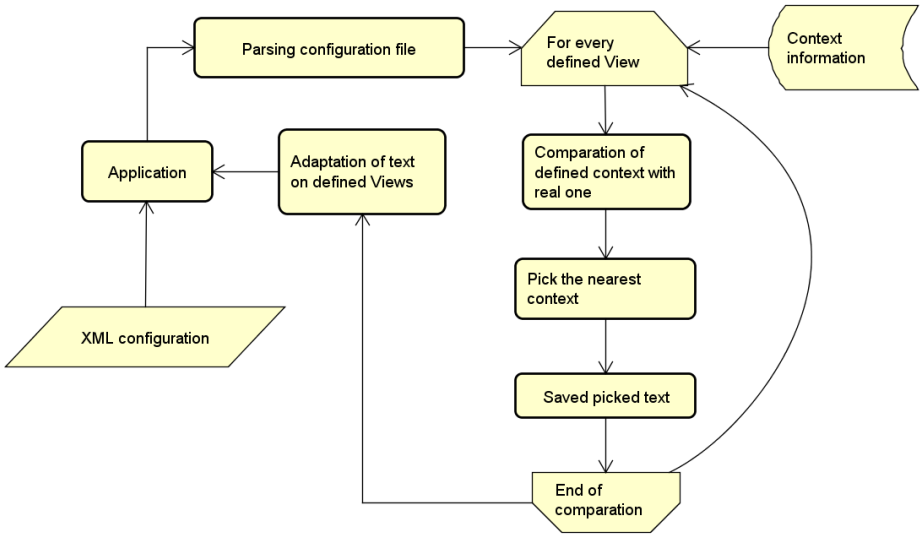
K tomuto účelu bude sloužit definovaná adaptace. Programátor nadefinuje bude mít možnost nadefinovat přesné texty, které se budou moci pomocí adaptace zobrazit správným uživatelům. Tyto texty bude programátor definovat právě ve zmiňovaném konfiguračním souboru, který by měl řídit celou adaptaci v aplikaci.

Konfigurační soubor bude muset obsahovat 3 základní prvky. Prvním z nich je specifikace adaptovaného View[1]. Ten bude obsahovat text, který má být určený k adaptaci. Dalším prvkem je definice kontextových informací, při kterých se má provádět adaptace. Těchto definic může být více a každá reprezentuje jeden případ, v jakém může být kontext uživatele. Ke každé definici kontextových informací poté bude přiřazen poslední prvek a tím je text určený k adaptaci. Pro každou definici View bude tedy existovat 0 až n možností páru, kontext - text, které budou určovat případné možnosti adaptace v daném View. Výjimkou v těchto párech bude defaultní text, který nebude obsahovat případ kontextových informací a bude použít v případě, že nebude získán ani jeden z definovaných případů kontextových informací.

Fungování definované adaptace bude tedy vypadat tak, že framework nejdříve zjistí kontext uživatele. Potom prohlédne konfigurační soubor a jednotlivé definice. Pro každý definovaný View porovná případy kontextů se zjištěným kontextem uživatele. Vybere takový kontext, který bude nejblíže zjištěnému, pokud nebude ani jeden dost blízko, vybere defaultní text. Vybraný kontext k sobě bude mít text, který se následně použije k adaptaci právě porovnávaného View. Tento proces se provede pro každé definované

[1] Základní stavební blok pro komponenty v UI

View v konfiguračním souboru.



Obrázek 3.1: Chování definované adaptace

Příkladem může být potvrzování a vyplňování formulářů. Zjišťujeme to, jak moc je uživatel ve spěchu a jestli je nervózní a projíždí formuláře s jistotou. Takovému uživateli je u formulářů zbytečné zobrazovat detailní popisky jednotlivých vstupů, ale spíše by potřeboval krátké a stručné vysvětlení. Na druhou stranu, pokud by byl uživatel nervózní a u každého formuláře by váhal, bylo by lepší mu pro jistotu zobrazit detailnější popisky ke vstupům, aby jsme mu dodali pocit, že ví co vyplňuje a nebyl tolik nervózní a nejistý.

Hlavním úkolem v této části je vhodně navrhnout, jak by měl vypadat konfigurační soubor a jak bude reprezentován kontext. Pro reprezentaci kontextu je zde možnost zvolit například CSCP (Comprehensive Structured Context Profiles) nebo jakýkoliv jiný formát založený na RDF (Resource Description Framewrok). Při použití těchto formátů by bylo velice snadné rozpoznání kontextů a práce s nimi. Ale programátorovi by to přidalo spoustu práce, když by musel psát pro každý definovaný text kontext v takto komplexním formátu. Pravděpodobně by musel mít nejprve někde předem už nadefinované kontexty a poté je jen používat. I přes to, by musel každý spát pravděpodobně zvlášť. Proto je třeba definici kontextů zjednodušit a nejlépe zakomponovat přímo do konfiguračního souboru s texty. Jelikož jsou veškeré definice, konfigurace a zdroje v android aplikacích psány v XML formátu a také RDF je psáno v XML formátu, bude i konfigurační soubor pro tento framework psán v XML formátu, vytvořeným přímo pro tento framework.

Další důležitou částí je samotný kontext uživatele. Je potřeba navrhnout, jaké kontextové informace sbírat a jak je ukládat. V této práci bude využito především uživatelských vstupů jako zdroje kontextových informací, tedy vstupů jako jsou hluk, okolní světlo, kamera uživatele, díky které se dá získat například nálada, okolí nebo třeba věk. Další vstupy pro kontextové informace nejsou vyloučeny, ale pro účely této práce je třeba začít s omezeným počtem

kontextových informací. Jelikož je kontextových informací velké množství, bude framework navržen tak, aby bylo jednoduché vždy přidat nový zdroj informací nebo přímo informace. Ve finální verzi může být ve frameworku obsaženo velké množství kontextových informací a programátor si poté pouze vybere, které chce použít a které ne. Kvůli konfiguračnímu souboru byl pro uložení kontextových informací vyloučen formát RDF i na něm postavený CSCP, tudíž bude potřeba vymyslet vlastní, který bude lehce porovnatelný s formátem použitým v konfiguračním souboru.

V neposlední řadě je důležité navrhnout vnitřní fungování frameworku a to hlavně způsob porovnání kontextů a poté způsob následné adaptace. U porovnání kontextů je důležité najít algoritmus, který vrátí vždy nejbližší kontext tomu reálnému a přitom bude dostatečně rychlý a nebude mít příliš velkou složitost, jelikož bude volán pro každý definovaný případ kontextu. Také je nutné aby byl co nejméně chybový, což by mohlo vyústit například v adaptaci přesně opačnou než je požadovaná. Jako nevhodnějším kandidátem se jeví počítat nejpodobnější kontext pomocí Euklidovi vzdálenosti. Adaptace jednotlivých textů musí být také rychlá a neměla by zpomalit chod aplikace. Jelikož se ale jedná pouze o změny textu, předpokládá se, že by ovlivnění chodu aplikace nemělo být znatelné a to i z důvodu, že textů na jedné obrazovce není většinou až tak velké množství.

Z analýzy vyplývají pro definovanou adaptaci následující požadavky:

Funkční

- Umožní programátorovi definovat view pro adaptaci.
- Umožní definovat případy kontextu pro každé view a k nim text určený k adaptaci.
- Umožní programátorovi nastavit defaultní text pro view.

Kvalitativní

- Zpracuje konfigurační xml soubor.
- Zpracuje výstupy senzorových a jiných knihoven pro získání kontextových informací.
- Vypočítá co nejpřesněji nejbližší případ kontextu z konfigurace vůči kontextu uživatele.
- Změní texty v definovaných View.

3.2 Část automatické adaptace

Automatická adaptace se bude více soustřeďovat na text-mining a získání dat z textů v aplikaci. Text-mining se v mobilních aplikacích používá zřídka, natož k adaptaci rozhraní, proto je část tohoto frameworku spíše experimentální. Nicméně má toto spojení velký potenciál s postupným vývojem lepšího hardware pro mobilní telefony a také lepší internetové dostupnosti.

Programátorovi umožní pouze kontrolu nad mírou adaptace, nad tím, co se bude adaptovat, jakým způsobem a také z jakých prvků aplikace se budou data pro adaptaci získávat. Rovněž by mělo být programátorovi umožněno upravovat nebo přidávat styly adaptace, pokud by si chtěl adaptaci více optimalizovat pro své řešení. Samotnou adaptaci a sběr dat však bude mít na starosti pouze framework, tudíž programátor nebude muset provádět žádné rozsáhlejší konfigurace nebo definice pro adaptaci.

Pro větší efektivitu a relevantnost získaných dat z textů je potřeba se zaměřit na větší bloky textu nebo na více prvků dohromady, klidně i na celou aktivitu[2]. Pokud by existovalo málo zdrojů informací nebo by se braly zdroje pouze z tlačítek nebo krátkých frázi, výsledná data by nemusela být vůbec relevantní. Díky těmto získaným datům můžeme dostat kontext aplikace a ten poté využívat k adaptaci. Kontext aplikace může obsahovat informace o sentimentu, formalitě, relevantnosti zobrazené aktivity nebo třeba použitém jazyku a stylu textu. Tyto bloky z jakých se budou získávat informace potřebné pro vytvoření kontextu bude definovat programátor pro každou aktivitu. Může vybrat celou aktivitu, nebo jen nějaké tipy bloků. Rovněž bude moci definovat, které prvky například bude framework ignorovat při získávání informací. Tomuto výběru budu říkat scope, který reprezentuje skupinu prvků pro získaní informací ke kontextu.

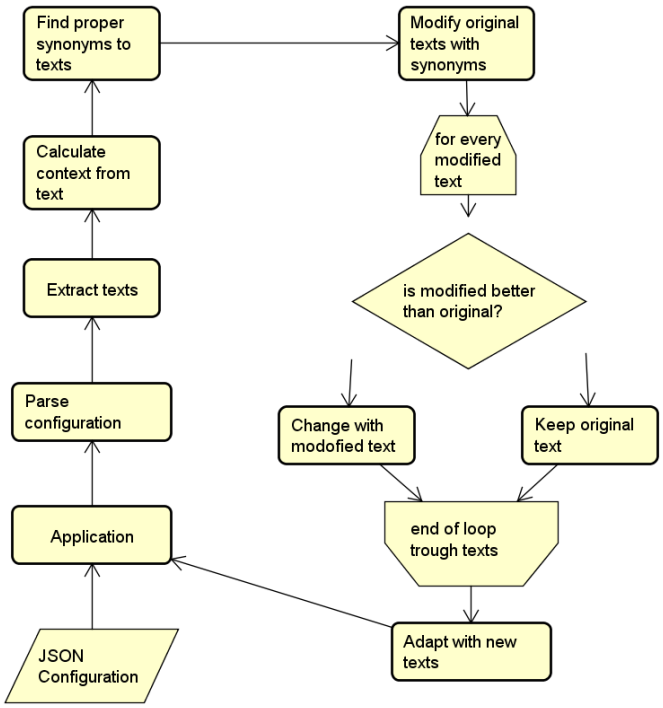
Získání kontextu lze provést několika způsoby. Tím nejjednodušším je použití datasetů a jednoduchý algoritmů na porovnávání a vyhledávání v daných datasetech. Text by se poté jen porovnal s daným datasetem, například dataset pro sentiment jednotlivých slov a frází, získaly by se data o jednotlivých blocích textu a následně by se sloučili dohromady a měli bychom potřebné informace ke kontextu. Takovýto postup sice není úplně součástí text-miningu, ale je jednoduchý na použití a při dobrém datasetu a algoritmu na slučování výsledků, lze získat poměrně dobrá data. Problémem je však právě slučování jednotlivých výsledků z hledání v datasetech, při větších a složitějších textech nebude výsledek nikdy zcela přesný a je i vysoká pravděpodobnost pro zcela nepřesné výsledky. Dalším a asi nejlepším způsobem je získání dat s použitím text-miningu jako takového, využití složitějších algoritmů a programů, které mohou využívat machine learning nebo i umělou inteligenci. Nevýhodou takového postupu je velká náročnost a složitost takových programů a také těch spolehlivých stále ještě není mnoho. Jako nejvhodnějším kandidátem se nabízí použití Stanford CoreNLP, který lze použít přímo v aplikaci nebo vytvořit vlastní server a k němu přístup například přes REST API[20].

Pro adaptaci aplikace se zde dají využít dvě možnosti, jak by šel využít získaný kontext aplikace z text-miningu. Jednou možností je využití kontextu a text-miningu na adaptaci textu. Druhou možností je využití kontextu aplikace k adaptaci uživatelského rozhraní.

Adpatace textu by se prováděla s použitím nástrojů text-miningu, kterými jsme také získali kontext aplikace. Asi nejslibnější možností adaptace by bylo vyhledání synonym k frázím, či slovům, které nejsou žádoucí a jejich následná změna za právě jejich synonyma. Změna by probíhala takovým

[2]Aktivita v Androidu, představuje jednu obrazovku s kterou uživatel pracuje

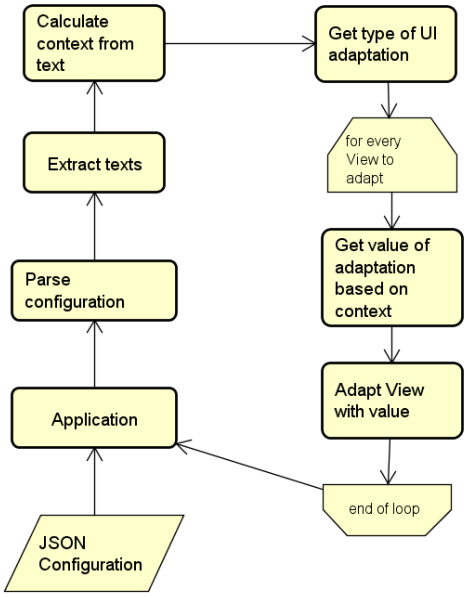
způsobem, že by se nejdříve v textech nahradily synonyma a následně by se zjistilo, jestli kontext zjištěný z nových adaptovaných textů je lepší než ten starý. Příkladem může být kontext obsahující sentiment. Zjišťovalo by se tak, jestli má adaptovaný text lepší hodnotu sentimentu (je pozitivnější) než text původní, v takovém případě by se adaptovala aplikace s použitím nového textu. Důležitým aspektem, na který bude potřeba brát ohled je význam textu. Vyhledání synonym nesmí v žádném případě změnit význam textu. Další otázkou je, která slova nebo fráze vybírat pro hledání synonym. Pro výběr by měl sloužit ten aspekt, podle kterého se poté bude porovnávat výsledný text. Tedy pokud budeme chtít adaptovat text, aby měl lepší sentiment, framework vybere pro hledání synonym taková slova, která sentiment textu dělají nejvíce negativní, ve zkratce vybere negativní slova a fráze, a k nim se následně pokusí najít synonyma, která nebudou tak negativního sentimentu. Pro adaptaci textu bude programátor muset uvést scope adaptace, kde chce aby se nahrazovali texty. Může uvést buď pouze konkretní prvky nebo klidně celou aktivitu.



Obrázek 3.2: Flowchart adaptace textu automatické adaptace

Využití kontextu k adaptaci rozhraní by vycházelo ze stejných dat, jako adaptace textu. Avšak adaptace bude mít po získání kontextu jiný směr a to místo textu, tak směrem k uživatelskému rozhraní. Pro takovouto adaptaci je potřeba určit typ adaptace, neboli jak se má adaptace projevit a také kde se má adaptace projevit, na jakých View nebo na jaké scope, ale tentokrát scope pro adaptaci. Tyto dvě věci si nakonfiguruje programátor. Pro typ adaptace bude mít frameowrk pár základní možností, kterými může být změna barvy

textu, změna barvy pozadí, změna velikosti textu, změna velikosti celého prvku a například změna stylu písma. View které programátor bude chtít adaptovat mohou být například pouze určitého typu nebo jen s nějakým atributem nebo třeba i celá aktivita. Rovněž bude mít programátor možnost nastavit, které prvky bude adaptace ignorovat, i přesto že budou zapadat do výběru adaptovaných prvků. Adaptace jako taková bude probíhat na základě nějakých předem definovaných a defaultních hodnot, které budou pro každý typ kontextu jiné a budou rozlišené podle případů, jakých můžou kontextové informace nabývat. Tyto defaultní hodnoty pro adaptaci bude moci programátor rovněž měnit, pokud bude chtít. Pouze určí hodnotu a případ kontextu, při jakém chce hodnotu použít. Také bude programátorovi umožněno definovat zcela nový typ adaptace a také jeho podoby pro jednotlivé případy kontextu.



Obrázek 3.3: Flowchart adaptace rozhraní automatické adaptace

Příkladem pro adaptaci uživatelského rozhraní může posloužit například internetová služba YouTube a komentáře u videí. Programátor by určil scope pro získání informací právě nad každým komentářem. Následně by mohl nastavit scope pro adaptaci na každou hlavičku komentáře a typ změny například na změnu barvy pozadí. Pro získání kontextu by se použila analýza sentimentu, tím bychom získali sentiment pro každý komentář. Následně by nad každým komentářem proběhla adaptace, kdy negativním komentářům by bylo pozadí hlavičky změněno na odstín červené, zatímco pozitivním by byly přiřazeny odstíny zelené, neutrální by zůstali podbarveny bíle. Tímto způsobem je možné informovat uživatele o sentimentu komentářů a zpříjemnit mu tak jejich čtení, jelikož například nemusí chtít negativní komentáře vůbec číst. Nebo může naopak autora zajímat proč a jaké negativní komentáře mu lidé píšou a tímto způsobem se mu budou lépe vyhledávat a zefektivní to jeho

práci.

Obrázek 3.4: Příklad adaptace YouTube komentářů

Příklad zmíněný výše by se dal použít i pro adaptaci textu, například pokud by se jednalo o video určené pro děti, mohl by framework zabránit negativním nebo nevhodným komentářům tím, že by pouze nahradil nevhodná slova tak, aby komentář nebyl nadále tolik nevhodný. U komentáře by poté mohlo být například zobrazeno tlačítko pro zobrazení originálu.

Dalším příkladem může být například aplikace se zprávami. V takovém případě by kontext mohl obsahovat informaci například o důležitosti nebo relevantnosti zpráv. Na základě toho by se adaptovala velikost textu a případně velikost bloků s titulky zpráv a díky tomu by měl uživatel ty důležitější zprávy více na očích, než ty méně důležité. Případně by se mohla použít adaptace s barvami a sentimentem, jako v předchozím případě a uživatel by se tak mohl vyvarovat negativních zpráv, pokud by o ně při raní kávě zrovna neměl zájem a nemusel si tak kazit hezké ráno.

Přidanou hodnotou pro sběr kontextu by mohla být jeho kombinace například s kontextem uživatele nebo jakýmkoliv jiným. Kontext aplikace by se tím mohl rozšířit například o uživatelovo prostředí nebo náladu. Tyto informace by se mohli následně využít při adaptacích jako přidaná hodnota pro rozhodování, jak adaptovat. Jednou možností by bylo kombinovat tyto

informace před adaptací a adaptovat až na základě jejich spojení. Druhou možností by bylo kontrolování každé adaptace například s kontextem uživatele, jestli pro něj zrovna není takováto adaptace nevhodná, i přesto, že je relevantní vůči kontextu aplikace. Třeba i přes zobrazení negativní zprávy aplikace nemusí zdůraznit negativitu (například červeně podbarvit titulek), pokud má uživatel už tak špatnou náladu. Nebo může v takovém případě dokonce zjemnit negativní zprávu nějakým stylem adaptace, aby uživatel neměl takový pocit negativity ze zprávy, pokud se už tak necítí dobře. Díky tomuto spojení bychom mohli mít unikátní adaptaci založenou na kontextu aplikace s ohledem na kontext uživatele.

Pro automatickou adaptaci vyplývají z analýzy následující požadavky:

Funkční

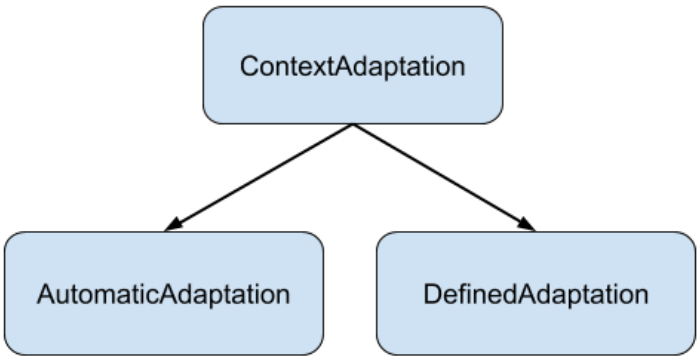
- Umožní programátorovi definovat objekt pro zjištění textového kontextu a definovat ignorované objekty.
- Umožní nastavit možnost adaptace - textová, adaptace UI, nebo obě.
- Umožní nastavit objekt, který se bude adaptovat.
- Umožní nastavit možnosti adaptace UI z výchozích možností pro jednotlivé objekty určené k adaptaci.
- Umožní nastavit ruční spuštění celého cyklu adaptace.
- Umožní přidání stylu adaptace UI.
- Umožní zvolit scope adaptace nezávisle nad více stejnými prvky.
- Umožní přidat nové pravidlo pro adaptaci.

Kvalitativní

- Z definovaného objektu načte veškeré textové prvky a uloží je.
- Z načtených textových prvků zjistí kontext. (Sentiment)
- Uloží původní kontext všech definovaných objektů.
- Najde synonyma k textům v objektech tak, aby byl výsledný textový kontext lepší (pozitivnější).
- K původním textům a kontextu uloží nové texty a jejich kontexty.
- Adaptuje textové prvky ze zjištěných nových textových prvků.
- Bude mít definovaná pravidla pro jednotlivé typy adaptace vůči jednotlivým typům kontextu.
  - Změna barvy textu
  - Změna barvy pozadí
  - Změna velikosti/stylu textu
  - Zmenšení, zvětšení prvku
- Podle definované UI adaptace adaptuje daný objekt podle pravidel.

Kapitola 4
Návrh

Návrh frameworku se bude skládat z návrhů obou částí frameworku zvlášť. Instance obou částí bude nakonec udržovat jedna třída, kterou bude používat programátor využívající tento framework. Tato třída bude fungovat jako singleton[1], tedy tak bude fungovat i celý framework.



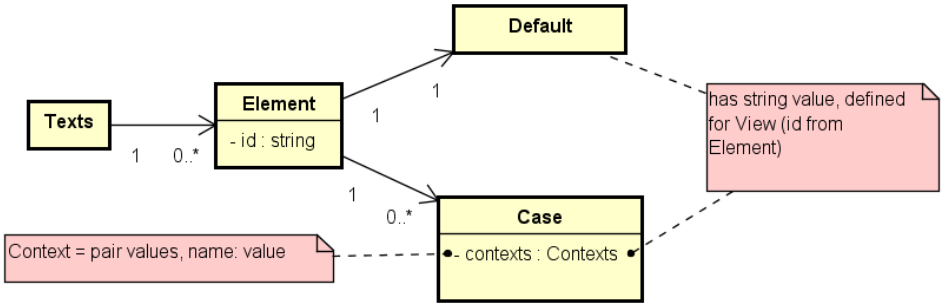
Obrázek 4.1: Základní struktura frameworku.

4.1 Část definované adaptace

Jak bylo zmíněno v analýze, jednou z důležitých částí bude určení formátu XML.

Pro udržení jednoduchosti pro programátora bude i zde formát v XML velmi jednoduchý. Jako kořenový element bude element <Texts>, který bude obsahovat veškeré definice. Elementem pro jednotlivá view bude element <element>, který musí mít atribut "id", který musí obsahovat hodnotu id hledaného View. Tento element pak bude obsahovat možnosti textů. Prvním bude element <default>, který bude bez atributů a bude obsahovat defaultní text. Za ním následuje libovolné množství elementů <case>, jehož atributy budou proměnné kontextu s hodnotami a uvnitř elementu text, který se použije pro dané View.

[1] Návrhový vzor, ve kterém je povolená pouze jedna instance dané třídy.



Obrázek 4.2: Metamodel XML formátu

Příklady 4.1: Ukázka XML souboru

```
<?xml version="1.0" encoding="utf-8"?>
<Texts>
    <element id="textView">
        <default>this is default case</default>
        <case emotion="2,2">This is first case</case>
        <case emotion="9,9">This is second case</case>
    </element>
    <element id="textView2">
        <default>this is default case</default>
        <case emotion="2,2">This is first case</case>
        <case emotion="9,9">This is second case</case>
    </element>
</Texts>
```

Na základě analýzy taktéž vyšlo najevo, že bude třeba vymyslet, jakým způsobem sbírat a udržovat kontextové informace. Pro udržení kontextu bude vytvořena třída ContextManager, která se o něj bude starat. Třída bude mít na starosti také formát kontextu a také také bude mít kontrolu nad uživatelskými vstupy pro kontext.

Kontextové informace budou drženy v základních meta datech. To znamená, že budou reprezentovány pouze číselnými hodnotami od 0 do 10. Na autorovi jednotlivých vstupů potom bude rozhodování, co bude reprezentovat 0 a co 10.

Jako příklad můžou být emoce, zmíněné v ukázce u XML formátu Př.4.1. Jejich zápis vypadal takto `emotion="2,2"`. Každá hodnota v řetězci definuje jinou emoci. První hodnota může například definovat radost, druhá například znechucení. Čím větší hodnota dané emoce je, tím víc se tato emoce u uživatele nachází. Tedy 0 pro radost by znamenala, že u uživatele nebyla zjištěna žádná radost, zatímco 10 by znamenala, že je uživatel plný radosti.

Kontext jako celek bude reprezentovat jeden objekt, jehož atributy budou kontextové informace.

Uživatelské vstupy musí být jednoduše rozšiřitelné, tím se myslí, že by měla

být možnost přidat jakýkoliv vstup a pouze nadefinovat jeho podobu. Proto bude každý uživatelský vstup muset implementovat interface, díky kterému se zajistí stejná podoba všech vstupů. Tento interface bude mít metody, pomocí kterých se získají kontextové informace v potřebném formátu. Pokud by byla potřeba přidat nový formát pro kontextovou informaci, stačí tento formát přidat do objektu reprezentujícího kontext a poté ho správně používat v konfiguračním souboru. To proto, že formát kontextové informace je důležitý pouze při porovnávání jednotlivých kontextových hodnot. Interface by měl v nejlepším případě mít jen jednu metodu na získání kontextu.

Další částí frameworku bude XML parser, který zpracuje konfigurační XML soubor. Ten jej převede do hashmapy, ve které klíčem bude id daného view a hodnotou pak další hashmapa. Tato hashmapa bude mít jako klíč hodnoty kontextových informací a hodnotou budou k nim přiřazené texty. Díky této struktuře bude vždy jasné, pro které View, který text použít.

Při porovnávání jednotlivých hodnot je důležité, aby byl algoritmus rychlý a málo náročný. Zároveň je potřeba eliminovat chybné výsledky, které by se na první pohled mohly zdát správné. Například bude-li kontext uživatele mít hodnoty 4, 5, 3. Pro zjednodušení zde bude použit pouze výčet čísel, u kterých je důležité pořadí, v implementaci potom každé číslo bude stát za nějakou proměnnou v kontextu. Tento kontext potom budeme chtít porovnat s následujícími hodnotami: 4, 1, 3 a 6, 3, 4 a 5, 4, 4. Je potřeba vybrat takové hodnoty, které jsou nejblíže k uživatelově kontextu, zároveň však takové, které mají nejmenší výkyvy. Je chybné vybírat takové hodnoty, které se liší v nejmenším počtu proměnných, protože by mohla, jako v prvních uvedených hodnotách (4,1,3), být jedna hodnota zcela vychýlená, a požadovaná adaptace by tak nemusela být správná. Proto nejjednodušším řešením tohoto problému je výpočet vzdálenosti v n-rozměrných prostorech, Euklidova vzdálenost. Jako n-rozměrný prostor se v tomto případě dá představit počet proměnných, které vytvoří prostor. Poté je v tomto prostoru potřeba najít takovou sadu hodnot, které jsou nejblíže k těm referenčním.

Vzorec pro výpočet Euklidovy vzdálenosti je následující:

$$d(k,l) = \sqrt{\sum_{i=0}^{n}(k_i - l_i)^2}$$

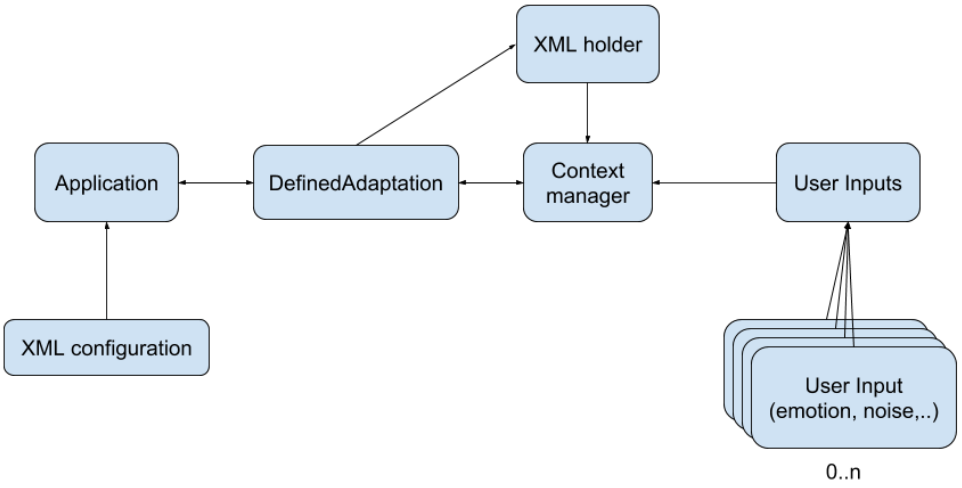
Po umocnění získáme:

$$d(k,l)^2 = \sum_{i=0}^{n}(k_i - l_i)^2$$

Jelikož v tomto případě vzdálenost jako taková není relevantní, bude použita umocněná formulace vzorce a následně budou porovnávány pouze tyto umocněné hodnoty. Takové hodnoty, které budou mít nejmenší vzdálenost, budou vybrány a uloženy do aktuálních relevantních použitelných spolu s textem. Ten poté bude použit k adaptaci textů v aplikaci.

O adaptaci textu se bude starat část frameworku s názvem DefinedAdaptation. Každá aktivita bude předávat tomuto objektu svůj Context[2] a ViewGroup[3] objekt, který drží veškerá view v aktivitě. DefinedtAdaptation při pokusu o změnu zjistí aktuální kontext uživatele a poté požádá kontext manažera o dodání relevantních textů pro jednotlivá View, a následně pomocí objektů z aktivity změní texty v jednotlivých View. Pro správné pracování celého frameworku je potřeba aby DefinedAdaptation měla pouze jednu instanci. Toho je docíleno díky tomu, že takto poběží celý framework a jediné co bude potřeba pohlídat bude předání nových objektů Contextu a ViewGroup. Tyto objekty se předají pokaždé, proběhne-li požadavek na instanci, který musí provést každá právě zobrazená aktivita v aplikaci. Tím se zajistí, že DefinedAdaptation bude mít vždy informace o aktuální zobrazené aktivitě.

Struktura definované adaptace bude vypadat takto:



Obrázek 4.3: Modelový návrh struktury definované adaptace

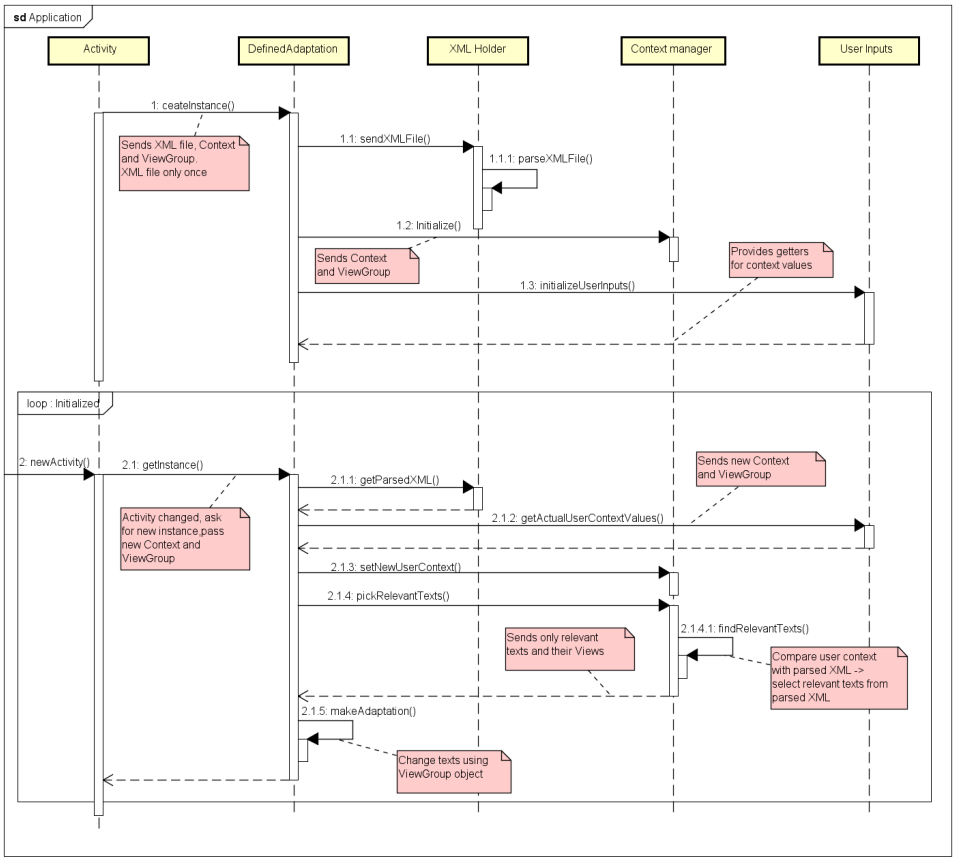
Fungování aplikace bude mít tento postup. Aplikace vytvoří první instanci DefinedAdaptation třídy a předá jí XML konfiguraci, Context a ViewGroup. Ta pošle XML konfiguraci do XML holderu, který převede XML soubor na požadovaný formát a předá Context managerovi. Ten bude mezitím nainicializován a s ním i UserInputs modul, který spravuje uživatelské vstupy. Při požadavku na adaptaci dodá UserInputs aktuální kontextové informace. ContextManager provede výpočet nejrelevantnějších textů na základě porovnání kontextových informací s daty z XML konfiguračního souboru. Context manager tyto vybrané relevantní texty uloží a následně je předá do zpět do DefinedAdaptation, který provede změnu na aktuální aktivitě. Adaptace proběhne vždy při změně aktivity, tedy vždy, když se aktualizuje Context a ViewGroup.

Sekvenční diagram Obr.4.4 ukazuje, jak bude probíhat volání jednotlivých

[2]Objekt v Androidu, obsahující informace a aktuálním prostředí aplikace

[3]Objekt v Androidu. Jedná se o speciální View, obsahující veškeré View pod ním (v aktivitě)

částí frameworku.



Obrázek 4.4: Sekvenční diagram části definované adaptace

4.2 Část automatické adaptace

Pro návrh této části je podobně jako u definované adaptace nutné navrhnout formu konfigurace frameworku pro programátora. Vzhledem ke stylu adaptace a datům ke konfiguraci, bude konfigurace v soubor typu JSON. Každá aktivita, kde se bude mít provádět adaptace bude mít vlastní JSON konfigurační soubor. Hodnotu ID tohoto souboru se vždy nastaví při dotazu na instanci frameworku. Pokud se nenastaví konfigurační soubor, není možné provést adaptaci na dané aktivitě. Příklad formátu konfiguračního souboru lze vidět na 4.2.

Příklady 4.2: JSON formát konfiguračního souboru

```
{
  "scope": "tag",
  "scopeName": "comment",
  "contextType": ["sentiment"],
  "ignorePrefix": "i_",
  "adaptScope": "tag",
  "adaptScopeName": "comments_header",
```

```
  "adaptPrefix": "comments_header",
  "adaptIgnorePrefix": "i_",
  "adaptation": "ui",
  "uiAdaptation": ["backgroundColor"],
  "multiple": "false",
  "customUIAdaptation":[]
}
```

Formát JSON se skládá z těchto atributů:

- scope

  Typ scope ze kterého framework bude získávat kontextové informace.
  Může nabývat hodnot: „tag“, „id“, „activity“

- scopeName

  String hodnota v atributu určeném ve scope. Tedy id nebo tag. Pro scope „activity“ není třeba určovat.

- contextType

  Typy kontextových informací, které se mají získávat, zapsány v poli. Prozatím bude k dispozici jen „sentiment“.

- ignorePrefix

  Prefix pro ignorování prvků i když jsou zahrnuty ve scope.

- adaptScope

  Scope pro adaptování.
  Může nabývat hodnot: „tag“, „id“, „activity“

- adaptScopeName

  Hodnota atributu definovaném v adaptScope. Nenastavuje se v případě celé aktivity.

- adaptPrefix

  Prefix pro prvky k adaptaci v případě nastavení adaptScope na „actitivity“.

- adaptIgnorePrefix

  Prefix pro ignorování prvků i přes jejich výskyt v adaptScope.

- adaptation

  Typ adaptace, který se má provádět na daném scope.
  Může nabývat hodnoty: „ui“, „text“, „all“

- uiAdaptation

  Styly adaptace, které se mají provádět. Nenastavuje se při adaptation „text“.
  Defaultně hodnoty: „textColor“, „backgroundColor“, „fontSize“, „fontStyle“, „viewSize“

- multiple

  Použitelný jen pro scope „tag“. Pokud je nastaven na „true“, bude framework počítat kontext pro každý prvek zvlášť a rovněž i adaptaci. AdaptScope musí být stejný jako scope.

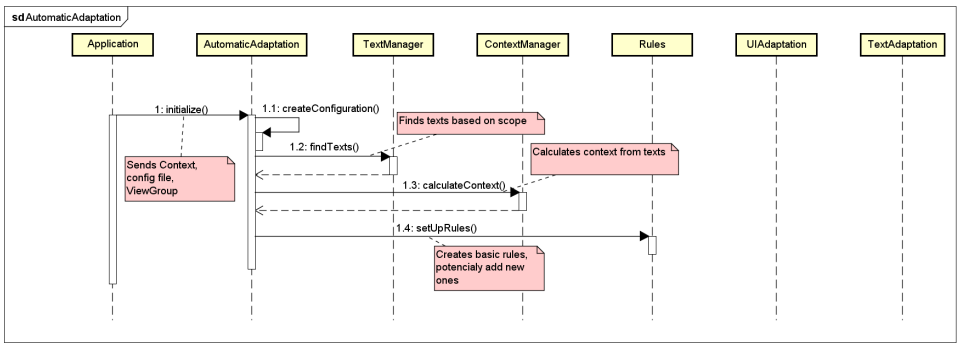
- customUIAdaptation

  Pole názvů typů adaptací uživatelských rozhraní, které si programátor naprogramuje sám. Není povinné, funguje jen na implementaci interface pro adaptaci dodanou do frameworku.

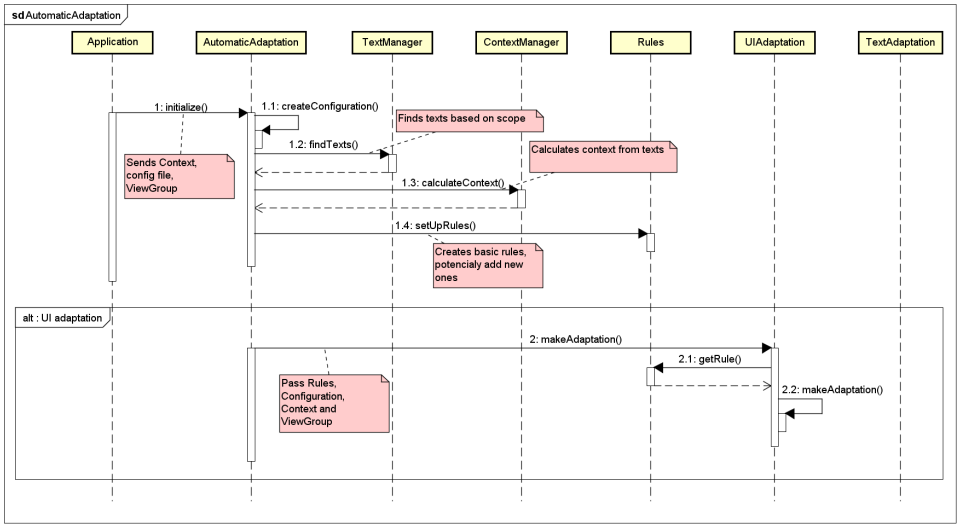
Konfigurační soubor bude reprezentován objektem, do kterého bude pro každou aktivitu převeden nový konfigurační soubor. Další práce s konfiguračním souborem bude už pouze přes jeho objekt pomocí metod get. O tomto objektu bude dále v práci mluveno jako o konfiguraci.

Na základě konfigurace bude nutné získat patřičné texty podle scope a scopeName. O získání textu se bude starat část pojmenovaná TextManager. Ta bude mít na starosti dolování textů z ViewGroup a jejich uchování. Rozšířením funkcionality by zde byla možnost text ukládat například do databáze. Díky tomu by při znovuotevření aktivity, kdy se načte nová konfigurace, by se pouze pouze porovnali uložené texty. Pokud by zůstali stejné, provedla by se ta stejná adaptace jako předtím. Tím pádem by se kromě textů ukládal i kontext, případně i přímo adaptace, pokud by se například jednalo o adaptaci textu, která je náročnější. Tímto způsobem by bylo možné zrychlit adaptaci.

Dalším důležitým prvkem je získání a uchování kontextu. K jeho získání bude sloužit knihovna Stanford CoreNLP, která může fungovat buď někde na serveru nebo přímo vložená do aplikace. O získání kontextu a jeho správu se bude podobně jako v definované adaptaci starat ContextManager. Bude navržen tak, aby byl jednoduše rozšířitelný o jakoukoliv kontextovou informaci, ale prozatím se v této práci bude pracovat pouze s informací o sentimentu. Ten může nabývat celkem 5 hodnot, jimiž jsou: „VERY_NEGATIVE“, „NEGATIVE“, „NEUTRAL“, „POSITIVE“, „VERY_POSITIVE“. Jak bylo zmíněno výše, rozšířením by mohli být i ukládání kontextu pro znovupoužití, čímž by se ušetřil čas a výkon potřebný na výpočet kontextu. Výše zmíněné bude pro oba typy adaptace stejné a bude se provádět při inicializaci části automatické adaptace.



Obrázek 4.5: Sekvenční diagram inicializace automatické adaptace



Obrázek 4.6: Sekvenční diagram adaptace uživatelského rozhraní

K těmto hodnotám kontextu budou existovat pravidla pro adaptace uživatelského rozhraní. Starat se o ně bude samostatná část Rules, kterou bude inicializovat hlavní modul, viz 4.5. Ta bude mít uložené základní nadefinované hodnoty pro každý druh adaptace a to ke každému kontextu. Vhodným prostředkem pro držení takovýchto pravidel by mohla být HashMapa, která by měla jako klíč typ adaptace a jako hodnotu další HashMapu, která by obsahovala samotné páry, případ kontextu a k němu hodnotu.

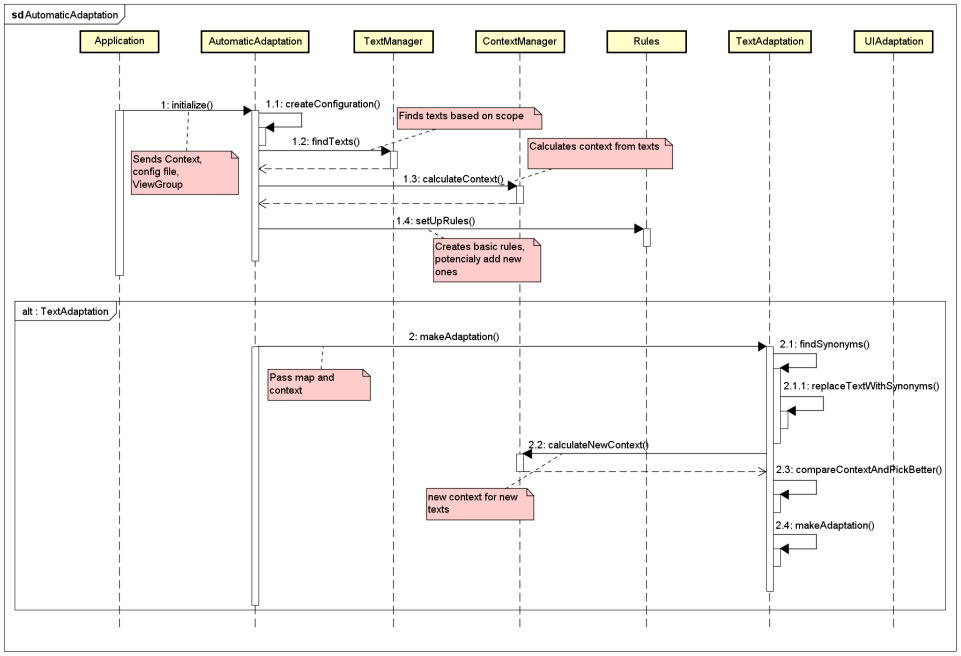
Příklady 4.3: Struktura pro uchování pravidel

```
HashMap<UIAdaptation, HashMap<Context, Vaule>> Rules;
```

Tyto pravidla budou mít uložené pouze nějaké základní defaultní hodnoty a základní typy adaptace. Základními typy bude změna barva písma, změna pozadí, změna velikosti textu, jeho stylu a také změna velikosti celého View. Pravidla bude moci programátor měnit pomocí funkce, do které zadá typ adaptace, případ kontextu a hodnotu pro danou změnu.

O samotnou adaptaci uživatelského rozhraní se bude starat samostatná část UIAdaptation. Pro adaptaci jí bude poskytnut objekt ViewGroup a konfigurace, na základě kterého bude vybírat View k adaptaci. Rovněž z konfigurace zjistí typ adaptace a na základě toho následně zjistí z Rules správnou hodnotu pro adaptaci a provede ji. Jak bylo zmíněno, bude podporovat i vlastní adaptace, které si bude moci naprogramovat programátor. Adaptaci poté jen přidá do frameworku jako instanci a do konfigurace její název. Toto chování popisuje sekvenční diagram 4.6.

Adaptace textů bude prováděna skrze modul TextAdaptation. Práce s ním bude podobná jako s UIAdaptation, ale vnitřní chování bude značně složitější. Jako parametr dostane mapu, kde klíčem bude TextView a hodnotou jeho text, a kontext. Prvním krokem bude zjištění nových textů. K tomu bude sloužit vyhledání synonym pro některé slova nebo fráze, které budou mít,



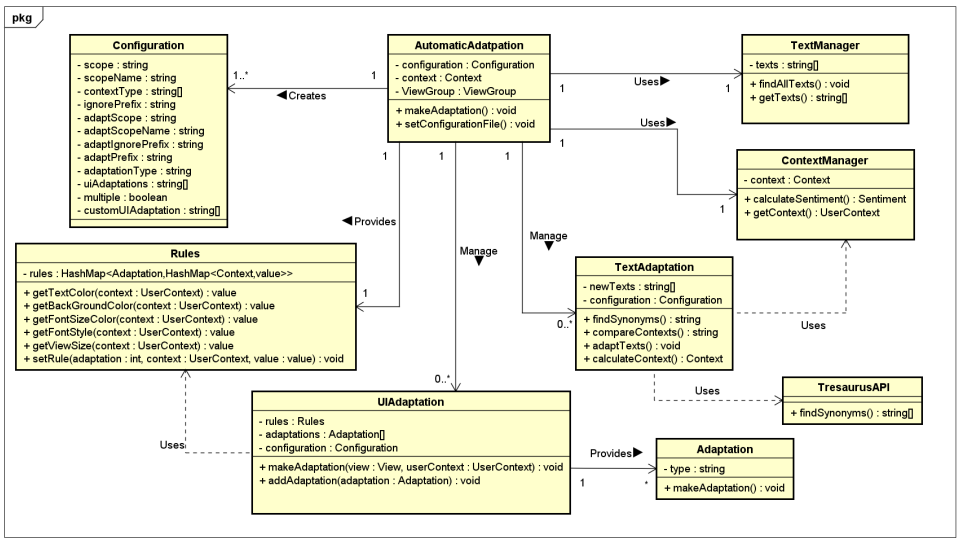
Obrázek 4.7: Sekvenční diagram adaptace textu

v případě práce se sentimentem, negativní význam.

Vyhledání synonym může proběhnout například pomocí Thesaurus WEB API pro jednodušší implementaci. Nejlepší by bylo najít vyhledávání, které zároveň kontroluje, aby nebyl změněn význam věty. O tuto kontrolu se částečně může postarat Stanford CoreNLP knihovna.

Jakmile budou nalezena synonyma, bude třeba spočítat nový kontext pro nové texty upravené pomocí synonym. Nový kontext se porovná se starým, pokud bude vhodnější, pozitivnější, bude starý text nahrazen novým. Pro zlepšení budoucí adaptace a její efektivity se nový text může uložit a poté jen použít, pokud by se původní text nezměnil. Adaptace textu je zachycena na diagramu 4.7.

Strukturu celé části automatické adaptace zachycuje class diagram 4.8.



Obrázek 4.8: Class diagram automatické adaptace

Kapitola 5
Implementace

Framework je pro operační systém Android a je psaný v programovacím jazyku Java. Framework používá některé knihovny třetí stran. Definovaná adaptace využívá jiné knihovny pro vstupy kontextových informací. Podobně jako automatická adaptace, ta však využívá knihovny i k fungování vnitřní logiky.

V implementaci se nepodařilo zprovoznit Stanford CoreNLP jako importovanou knihovnu. Možností by bylo ještě spustit server na kterém Stanford CoreNLP poběží, ale to nebylo původním záměrem. Proto je tedy v implementaci tato knihovna vynechána a s ní i funkcionalita adaptace textu v automatické adaptaci. Avšak v implementace je připravená pouze na implementování metod používajících CoreNLP, který se v budoucnu může podařit zprovoznit.

5.1 Požadavky

Framework je určený pro operační systémy Android minimální verze 21. Pokud by se povedlo zprovoznit Stanford CoreNLP, minimální verze by byla posunuta na verzi 25 a aplikace by musela podporovat Javu 8. V případě serveru by tahle změna nebyla nutná. Framework je nepřenositelný na jiné platformy.

5.2 Závislosti

Framework je závislý na následujících knihovnách.

- Affectiva Emotion SDK[25]

  Knihovna pro získání informací z kamery zařízení. Získává emoční hodnoty. Použita v definované adaptaci jako zdroj kontextových informací. Pro použití Affectiva Emotion SDK je potřeba přidat závislost na repositář affectiva.

Příklady 5.1: Přidání závislosti na affectiva repositáři

```
    allprojects {
```

```
    repositories {
        maven {
            url "http://maven.affectiva.com"
        }
    }
}
```

- GSON[26]

  Nástroj pro parsování JSON souborů na objekty. Použit pro zpracování konfiguračního souboru v automatické adaptaci.

5.3 Struktura

Struktura projektu je implementována podle návrhu. Hlavní třídou je třída ContextAdaptation. S touto třídou a její jedinou instancí bude pracovat programátor při používání frameworku. Třída je navržena podle návrhového vzoru singleton a při každém dotazu na instanci se stará o to, aby obě části frameworku dostali aktuální Context a ViewGroup, případě id pro kameru v případě Definované adaptace.

Příklady 5.2: Struktura hlavní třídy ContextAdaptation

```
/**
 * Main singleton class for ContextAdaptationFramework
 */
public class ContextAdaptation {
    private static ContextAdaptation instance = null;

    private ContextAdaptation(Context context,
            ViewGroup viewGroup, int cameraSurfaceView){}

    /**
     * Creates and return the only
     *    instance of ContextAdaptation
     * @param context actual activity Context
     * @param viewGroup actual activity ViewGroup
     * @param cameraSurfaceView camera SurfaceView
     *  id for defined adaptation
     * @return ContextAdaptation instance
     */
    public static ContextAdaptation getInstance(Context context,
            ViewGroup viewGroup, int cameraSurfaceView){}

    /**
     * Sets id for defined adaptation configuration file
```

```
     * @param id ID of configuration file (R.xml.texts)
     */
    public void setDefinedAdaptationConfigurationFile(int id){}

    public void startDefinedAdaptation(){    }

    public void startAutomaticAdaptation(){}

    /**
     * Sets configuration json for every activity
     * @param id id of configuration file (R.raw.file)
     */
    public void setConfigurationFile(int id){}

    /**
     * Add new rule to automaticAdaptation of UI
     * @param ui Type of adaptation
     * @param sentiment Case for sentiment context
     * @param value Value of adaptation (color, size, ...)
     */
    public void setRule(UIAdaptations ui,
            Sentiment sentiment, int value){}

    /**
     * Called whenever is asked for instance.
     * Set new Context and ViewGroup.
     */
    private void reinitialize(int cameraSurfaceView){}

    private void setContext(Context context){}
    private void setViewGroup(ViewGroup viewGroup) {}
}
```

Tato hlavní třída se tedy stará o udržování a chod instancí obou částí frameworku. Ty pak pracují zcela samostatně podle návrhu.

5.4 Použití

Před použitím frameworku je nutné aby programátor nastavil povolení pro kameru, internet a stav sítě. Tyto povolení je potřeba přidat do manifestu aplikace následovně.

Příklady 5.3: Přidání povolení do manifestu

```
<?xml version="1.0" encoding="utf-8"?>
<manifest >
```

```
  <uses-permission
    android:name="android.permission.INTERNET"/>
  <uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE"/>
  <uses-permission
    android:name="android.permission.CAMERA"/>
    ...
  <application>...</application>
</manifest>
```

Dále by měl před spuštěním frameworku zkontrolovat, jestli jsou povolení udělena. Pokud by nebyla, vyžádá se povolení ještě explicitně v kódu viz 5.4. Následně je možné použít například posluchač na výsledek udělení povolení a dle něj se zachovat dále viz 5.5.

Příklady 5.4: Příklad kontroly a vyžádání povolení.

```
private void setPermissions() {
    if (ContextCompat
            .checkSelfPermission(
                this, Manifest.permission.CAMERA)
            != PackageManager.PERMISSION_GRANTED)
    {
        ActivityCompat
        .requestPermissions(this,
            new String[]{Manifest.permission.CAMERA},
                MY_PERMISSIONS_REQUEST_CAMERA);
    } else {
        runContextAdaptation();
    }
}
```

Příklady 5.5: Příklad posluchače na výsledek žádosti o povolení.

```
@Override
public void onRequestPermissionsResult(int requestCode,
    @NonNull String[] permissions,
    @NonNull int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_CAMERA: {
            // If request is cancelled,
            // the result arrays are empty.
            if (grantResults.length > 0
                && grantResults[0]
                == PackageManager.PERMISSION_GRANTED) {
                runContextAdaptation();
            }
        }
    }
```

```
}
```

Po získání veškerých povolení je možné začít využívat framework. Pokud by byl do frameworku přidán nový zdroj informací, bylo by potřeba přidat další povolení a vyžádat je u uživatele, obdobně jako s povolením pro kameru.

Framework může být inicializován získáním instance ContextAdaptation třídy. Tuto instanci je potřeba získat v každé aktivitě, aby každá aktivita předala aktuální Context, ViewGroup a kameru. Pro získání instance je třeba zavolat statickou metodu, která vrátí instanci ContextAdaptation.

Příklady 5.6: Získání instance ContextAdaptation

```
contextAdaptation = ContextAdaptation
    .getInstance(this, viewGroup, R.id.cameraPreview);
```

Při zavolání instance se inicializují obě části frameworku, rovněž tak některé stěžejní moduly jednotlivých částí. Při prvním volání instance se vytvoří instance obou částí. Definovaná část rovněž vytvoří instance XMLHolderu a také instance vstupů kontextových informací, v tomto případě pouze instanci AffectaInput třídy, která se stará od informace o emocích. Třída AffectaInput při inicializaci pouze vytvoří detektor a nastaví emoce, které bude analyzovat. Automatická adaptace při inicializaci vytvoří pouze instanci Rules a uloží aktuální Context a ViewGroup.

5.4.1 Spuštění definované adaptace

Pro spuštění definované adaptace je potřeba nejdříve nastavit id pro konfigurační soubor. Ten zpracuje již inicializovaný XMLHolder, který xml soubor přečte a uloží do struktury 5.7.

Příklady 5.7: Struktura pro uchování konfiguračního XML

```
HashMap<String, HashMap<ContextManager, String>> xmlDefinition
```

V této struktuře je první klíč reprezentuje TextView a jeho hodnoty se poté skládají z dvojic kontextů a textů.

Příklady 5.8: Nastavení id konfiguračního souboru

```
contextAdaptation
    .setDefinedAdaptationConfigurationFile(R.xml.texts);
```

Po nastavení konfiguračního souboru je možné spustit adaptaci. Spuštěním adaptace se spustí i knihovny pro získání kontextových informací.

Příklady 5.9: Spuštění definované adaptace

```
contextAdaptation.startDefinedAdaptation();
```

Voláním této funkce se spustí knihovny na získání kontextových informací. Následně se spustí výpočet relevantních textů, o který se postará ContextManager třída. Po výpočtu relevantních textů se provede adaptace, kterou provádí přímo třída DefinedAdaptation.

5.4.2 Spuštění automatické adaptace

Podobně jako u definované adaptace, tak i u automatické je nejdříve potřeba nastavit konfigurační soubor pro danou aktivitu. Ten se pomocí GSON knihovny převede na objekt Configruation, se kterým následně framework pracuje.

Příklady 5.10: Nastavení id pro JSON konfiguraci

```
contextAdaptation.setConfigurationFile(R.raw.activity_second);
```

Nastavením konfiguračního souboru se rovněž vytvoří instance TextManageru a ContextManageru, které pro své vytvoření potřebují právě konfiguraci. TextManager při své inicializaci rovnou podle konfiguraci extrahuje veškeré potřebné texty k zjištění kontextu. A ContextManager na základě získaných textů provede analýzu sentimentu a vytvoří kontext.

Dalším krokem je spuštění automatické adaptace.

Příklady 5.11: Spuštění automatické adaptace

```
contextAdaptation.startAutomaticAdaptation();
```

Spuštěním adaptace se v hlavní třídě AutomaticAdaptation rozhodně podle konfigurace, který typ adaptace se provede. Typy jsou adaptace text, adaptace uživatelského rozhraní a nebo oba zároveň. Po rozhodnutí, který typ se spustí je odpovědnost přesunuta k relevantnímu typu. Tedy pro adaptaci rozhraní je vytvořena instance UIAdaptation, které se předají pravidla, kontext a konfigurace a následně se zavolá funkce pro spuštění adaptace, která má jako parametry ViewGroup a kontext, v tomto případě sentiment.

Příklady 5.12: Spuštění adaptace uživatelského rozhraní

```
uiAdaptation
    = new UIAdaptation(rules, context, configuration);
uiAdaptation
    .makeAdaptation(viewGroup, contextManager.getSentiment());
```

Spuštění adaptace textu probíhá obdobně, pouze s tím rozdílem, že se nepředává objekt Rules a místo ViewGroup se předává HashMapa s jednotlivými texty pro adaptaci, kterou vytvoří TextManager.

Příklady 5.13: Spuštění adaptace textu

```
textAdaptation
  = new TextAdaptation(context, configuration);
textAdaptation
  .makeAdaptation(textManager.getTextsMap(),
    contextManager.getSentiment());
```

Při nastavení multiple konfigurace na hodnotu „true“ se místo jednoho sentimentu posílá vícero sentimentů pro každý scope zvlášť. Další chod adaptace si už řídí oba typy adaptace nezávisle na sobě, tak jak je popsáno v návrhu a analýze.

5.5 Struktura definované adpaptace

V této čísti popíšu strukturu kódu definované adaptace. Hlavní třídou tohoto modulu je DefinedAdaptation, která řídí průběh adaptace. Třída DefinedAdaptation implementuje DefinedAdaptationInterface, který je používán v ContextAdaptation třídě.

Příklady 5.14: DefaultAdaptationInterface

```
public interface DefinedAdaptationInterface {

    void reinitialize(Context context,
        ViewGroup viewGroup, int cameraSurfaceView);

    void startAdaptation();

    void setDefinedAdaptationConfigurationFile(int id);
}
```

Třída se stará hlavně o získání a předání dat mezi ostatními třídami v modulu. Kromě toho ale implementuje logiku provádění adaptace textu, tou je metoda `makeAdaptation()`. Tato funkce získá z ContextManager třídy již vypočítané relevantní texty k adaptaci. Následně iteruje přes veškeré TextView určené k adaptaci a pro každý definovaný TextView nalezený na aktuální aktivitě, neboli v aktuálním Contextu, provede adaptaci textu pomocí funkce `setText()`. Tato funkce funguje pro TextView a prvky, které z něj dědí.

5.5.1 XMLHolder

Tato třída je odpovědná za udržování XML konfiguračního souboru. Rovněž XML soubor převádí pomocí funkce `readXML()`.

5.5.2 ContextInputInterface

Rozhraní pro implementaci vstupů kontextových informací. V aktuální implementaci je zcela jednoduché a obsahuje pouze metodu pro získání pole číselných hodnot, které v tomto případě znamenají hodnoty jednotlivých emocí.

5.5.3 ContextManager

Třída ContextManager má na starosti uchování kontextu a také obsahuje stěžejní metodu pro zjištění relevantních textů k aktuálnímu uživatelskému kontextu. Tato metoda používá podle návrhu Euklidovu vzdálenost pro výpočet vzdáleností jednotlivých kontextů mezi sebou. Nejdříve vezme defaultní text a nastaví mu počáteční vzdálenost. Tato vzdálenost je aktuálním minimem a defaultní text je pro začátek i textem relevantním pro dané TextView. Následně se pro dané TextView vypočítají vzdálenosti všech kontextů a ten

který má nejmenší vzdálenost je dosazen jako relevantní. Funkce nakonec vrátí Mapu všech procházených TextView a k nim relevantních textů.

5.6 Struktura automatické adaptace

V této části objasním strukturu kódu automatické adaptace o kterou se stará třída AutomaticAdaptation implementující AutomaticAdaptationInterface, se kterým pak pracuje ContextAdaptation.

Implementace toho modulu není zcela kompletní a to právě z důvodu nefunkčnosti frameworku Stanford CoreNLP, který je obtížné zprovoznit na platformě Android. Chybějící implementace je buďto naznačena a patřičně okomentována nebo nasimulována náhodnými hodnotami.

Příklady 5.15: AutomaticAdaptationInterface

```
public interface AutomaticAdaptationInterface {

    /**
     * Sets new Context and ViewGroup
     * @param context Context
     * @param viewGroup ViewGroup
     */
    void reinitialize(Context context, ViewGroup viewGroup);

    void startAdaptation();

    void setConfigurationFile(int id);

    void setRule(UIAdaptations ui,
        Sentiment sentiment, int value);

    void addAdaptation(Adaptation adaptation);
}
```

Podobně jako u definované adaptace je třída odpovědná za chod celého modulu a fungování jednotlivých tříd. Rovněž řídí tok dat mezi třídami. Modul automatické adaptace obsahuje některé enumy, s kterými se následně lépe pracuje při rozhodování. Tyto enumy reprezentují možnosti hodnot některých atributů konfigurace.

5.6.1 Enumy

Enumy jsou v modulu celkem 4.

Prvním je enumu Scope, který reprezentuje možnosti scope atributu. Jeho hodnotami jsou `id, tag, activity`.

Dalším enumem je AdaptationType, který reprezentuje typ adaptace. Hodnoty jsou `text, ui, all`.

Třetím enumem je reprezentace typů adaptace rozhraní, UIAdaptations. Hodnoty jsou následující `textColor`, `backgroundColor`, `fontSize`, `fontStyle`, `viewSize`.

Čtvrtý enum neslouží k lepší práci s konfigurací, ale pro lepší práci s kontextem. Reprezentuje sentiment, neboli kontextovou informaci, nese název Sentiment. Obsahuje kromě String hodnot, také int hodnoty, jelikož se sentiment počítá jako číslo od 1 do 5.

Příklady 5.16: Sentiment enum

```
public enum Sentiment {
    VERY_NEGATIVE(1), NEGATIVE(2),
        NEUTRAL(3), POSITIVE(4), VERY_POSITIVE(5);

    public final int Value;

    private Sentiment(int value){
        Value = value;
    }

    public static Sentiment forInt(int id) {

        for (Sentiment sentiment: values()) {
            if(sentiment.Value == id) {
                return sentiment;
            }
        }
        return null;
    }
}
```

5.6.2 Configuration

Třída Configuration zastupuje objekt, který má stejné atributy jako konfigurační JSON soubor. Ten je do této třídy překládán, hned při načtení. Třída obsahuje pouze metody get na veškeré atributy.

5.6.3 Rules

Třída Rules obsahuje soubor pravidel, tak jak bylo specifikováno v návrhu. Navíc obsahuje atribut `customRules`, do kterého je možné přidávat vlastní pravidla, které lze následně využít při implementaci rozhraní Adaptation. Třída rovněž poskytuje metody na změny hodnot základních pravidel.

5.6.4 Rozhraní Adaptation

Rozhraní Adaptation poskytuje rozhraní pro implementaci vlastních typů adaptací. Rovněž základní adaptace rozhraní implementují právě tento in-

terface. Rozhraní poskytuje metodu k adaptaci, která má parametry Rules, View pro adaptaci a Sentiment.

Příklady 5.17: Rozhraní Adaptation

```
public interface Adaptation {

    String getAdaptationType();

    void makeAdaptation(Rules rules,
        View view, Sentiment sentiment);
}
```

5.6.5 TextManager

Třída TextManager má za úkol uchování textů. Obsahuje metody na extrakci TextView a textů z celé aktivity, které poté poskytuje zbytku modulu.

5.6.6 ContextManager

Třída ContextManager je odpovědná za zjištění kontextových informací a sestavení kontextu z textů poskytnutých TextManager třídou. V této implementaci se pracuje pouze s informací o sentimentu textu. Výpočet sentimentu probíhá buď nad celými poskytnutými texty nebo nad každým zvlášť, kdy poté uchová Mapu jednotlivých TextView, ke kterým přiřadí vypočítané sentimentu.

Kvůli problému s implementaci Stanford CoreNLP do android projektu, je tato třída značně zjednodušena. Místo reálného výpočtu sentimentu, kdy by se text poslal do knihovny Stanford CoreNLP a ta by následně vrátila jeho sentiment, se sentiment vypočítává náhodně, pomocí funkce Random, jako simulace právě reálného výpočtu.

5.6.7 TextAdaptation

Tato třída je spíše prototypem reálné třídy. Měla by mít na starosti právě adaptaci textu, ale právě díky chybějící knihovně Stanford CoreNLP zde není způsob jak kontrolovat adaptaci. Třída je schopná zatím využívat pouze Thesaurus API, díky kterému je schopna získávat synonyma pro jednoduché fráze nebo slova, ale samotnou adaptaci není reálné použít, protože neexistuje žádná kontrola nad nalezenými synonymy a jejich vlivem na výslednou změnu sentimentu nebo celého kontextu.

Nicméně třída TextAdaptation má poskytnuté veškeré potřebné metody pro adaptaci, chybí pouze jejich implementace.

5.6.8 UIAdaptation

UIAdaptation je třída odpovědná za adaptaci uživatelského rozhraní. Má přístup ke všem základním adaptacím a také drží vlastní adaptace, které

je vždy při reinicializaci frameworku, tedy dotazu na instanci, nutné přidat pomocí metody `addAdaptation(Adaptation adaptation)`.

Dále obsahuje metody pro provedení adaptace. Jsou to dvě přetížené metody, jedna je pro adaptaci celého bloku jako celku a druhá je pro adaptaci několika scope s vlastní hodnotou sentimentu. Obě metody pracují stejně, pouze jinak přistupují k získání hodnoty pro adaptaci. Mají za úkol podle typu adaptace rozhraní provést adaptaci na určených View, které zároveň musí kontrolovat, jestli neobsahují ignorovaný prefix.

5.7 Instalace

Demo aplikace obsahující modul s frameworkem je v .zip na přiloženém CD. Aplikace by měla být spustitelná na každém mobilním zařízení bežícím na Android 7.0 a vyšší. Zařízení rovněž musí mít přední kameru a povolit její použití při spuštění.

Aplikace je přiložena jako Android Project, pro její spuštění je tedy třeba rozbalit .zip soubor a importovat projekt do Android Studio. Aplikace byla testována v Android Studio 3.0 na Windows 10 s telefonem Xiaomi Mi5 s Android 7.0.

Kapitola 6
Testování

Testování je nedílnou součástí vývoje a návrhu jakéhokoli software a proběhne i v této práci. Kvůli povaze navrhovaného frameworku je v tomto stádiu téměř nemožné a hlavně nelogické provádět testování bez uživatelů. Proto bude tedy tato práce otestována s uživateli. Testovat se bude spíše návrh a také jednotlivé typy adaptace a jejich potenciál. Uživatelům bude představen prototyp zobrazen v implementaci.

6.1 Metodiky

Pro testování byly vybrány celkem tři metodiky pro testování s uživateli. Tyto metodiky umožní říci, jestli je práce směřována správným směrem, jestli se uživatelům zamlouvají jednotlivé prvky frameworku, jako jsou druhy adaptace nebo druhy sběru kontextových informací a jejich použití. Rovněž je očekávaným výsledkem představa o uživatelské oblíbenosti aplikací využívajících tento framework.

První metodikou je obyčejné interview s uživateli, ve kterém jim představím ve zkratce návrh a fungování aplikace využívající tento framework a pobavím se s ním o potenciálu a jeho názorech na framework jako celek, ale i na jednotlivé části, jako je adaptace textu, UI a například sběr kontextových informací[28].

Druhou je využití on-line dotazníku, ve kterém se budu ptát na hodnocení jednotlivých adaptací a jejich využití[28].

Třetí metodikou je metodika nazývající se SUS neboli System Usability Scale, která bude pojata jako test možné funkční aplikace a dokáže nám říct, jak moc by byla použitelná. Tato metodika bude dávána účastníkům první nebo druhé metodiky, aby již měli představu o fungování, zároveň se před testováním ukáže a vyzkouší prototyp aplikace[28].

6.2 Interview

Cílem interview je představit svůj nápad a fungování frameworku a v návaznosti na představení získat odezvu uživatelů. S uživateli bude probíráno každé fungování frameworku zvlášť a diskutováno. Každé fungování vysvětlím

a ukážu na příkladu, řeknu známé negativa a základní pozitiva. V závislosti na vysvětlení je cílem získat odpovědi na otázku přínosu frameworku a toho, jestli by s tím byl uživatel spokojen, v čem konkrétně vidí přínos při používání a také jestli by uvítal tuto funkcionalitu ve více aplikacích. Případně i na to, co by mu na něm vadilo, například snímání emocí, a také v jakých situacích by fungování frameworku rád viděl.

Obecné otázky dotazované v interview:

- Dokážete si představit takovéto chování v běžných aplikacích? V jaký případně?
- Vadilo by vám, že by aplikace sledovala tolik informací o vás? Které ano a které ne?
- Jaký potenciál vidíte v takovýchto aplikacích?
- Jaký přínos si myslíte, že budou aplikace používající podobné funkce a metody mít pro uživatele při používání aplikací? (rychlost, efektivity, pohodlnost)

Otázek bude obecně více, ale výše zmíněné otázky budou stěžejní pro analýzu výsledků z interview, které bude vedeno více jako debata o tématu.

6.2.1 Výsledky interview

Interview proběhlo dohromady s šesti osobami. Se třemi programátory, z toho 2 se zaměřením na front-end a jeden spíše na back-end. A zbylí tři byli laici v oboru, jeden účastník studentem VŠ, který nemá zkušenosti s vývojem software, ale mobilní aplikace používá hojně denně a druhý byl již pracující člověk mezi 40 a 50 lety, který používá pouze nejnutnější aplikace a třetí pracující člověk kolem 20 let v počítačové grafice, ale bez zkušeností s programováním nebo vývojem aplikací.

Interview s programátory bylo jednodušší, jelikož více znají toto téma. Ale bylo důležité získat aspoň nějaké informace i od lidí, kteří neznají práci v pozadí software.

1. Front-end programátor

- Nadšený z celého návrhu, velmi se mu líbí myšlenka aplikace na adaptující se přímo na míru použití.
- Dokáže si představit tyto funkce ve většině aplikací v budoucnu.
- Nemá žádný problém s tím, že by aplikace zjišťovala informace pro adaptaci. Tyto věci tolik neřeší, jde mu o použitelnost software.
- V aplikacích vidí velký potenciál, dokonce by podobně funkce rád používal pro vývoj v budoucnu.
- Přínosů může být spousty, rozhodně větší pohodlí a efektivita při používání aplikace, s tím i rychlost a bezchybnost.

2. Front-end programátor

- Automatická adaptace mu přijde jako cenzura a může být v rozporu se svobodou slova při nějakých použitích. Dobrý by bylo ale použití u psaní příspěvků nebo zprávy jako návrh na změnu.
- Dokázal by si to představit například v single-player hrách.
- Sledování by mu nevadilo, pokud by aplikace nesledovala nějaké detailnější a zásadnější informace.
- Potenciál tolik nevidí, jelikož si myslí, že je to příliš časově náročné na vývoj a zatím ne tolik chtěné.
- Adaptace UI na základě textu mu přijde zajímavá. Dokázala by zrychlit a zefektivnit práci, jelikož by uživatel viděl informace dříve. Což pro něj může být i příjemnější.

3. Back-end programátor

- Nápad se mu líbí, ale záleží hodně na správném použití.
- Použití si dokáže představit např. u zmíněných komentářů nebo adaptaci textu ve zprávách (návrhy).
- Trochu by měl problém se sledováním uživatelských informací. Hlavně sledováním emocí.
- Potenciál v budoucnosti ano, ale vidí velký prostor pro zneužití informací.
- Přínosem by mohla být například edukativního charakteru při automatické adaptaci textu ve zprávách. Pak i zjednodušení používání.

4. Student

- Nápad byl zajímavý, ale nemyslí si, že by využil. Nechtěl by aby se aplikace adaptovala podle chování uživatele.
- Nelíbilo by se sledování uživatele kvůli informacím.
- Potenciál by mohl být například ve skrývání negativních věcí.
- Přínosů moc nevidí, ale myslí, že určitě nějaké budou a v budoucnu by se mohly najít dobrá použití, na které by měl jiný názor.

5. Pracující mladý

- Zajímavé téma do budoucna. Má strach ze zneužití a uchování citlivých dat o uživatelích.
- Dokáže si to představit použití za podmínek, že by aplikace neposílali žádná data. To samé platí se sledováním informací, trochu nepříjemné je ale sledování emocí, ale je zároveň zajímavé.

- Potenciál vidí hlavně v budoucnu, až bude lepší HW a bude se moci většina údajů analyzovat přímo na zařízení.
- Přínosem by mohla být větší efektivita a příjemnost při používání.

6. Pracující 40+

- Pro mladší generaci to je velká budoucnost. Pro její generaci už tolik ne. Mohla by být přínosem adaptace textů pro lepší pochopení aplikace a jejího ovládání s čím mají lidé ve vyšším věku problémy. Lepší popisky ovládání a formulářů například.
- Použití by si dokázal představit v hudební aplikaci.
- Sledování není problémem, pokud bude mít záruku, že se data nikam neposílají.
- Při používání by to mohlo pomoct k větší efektivitě a příjemnějšímu použití.

Na základě všech interview jsem zjistil zajímavé poznatky. Největším problémem nebo spíše komplikací při používání uživatelských informací je jejich zpracování. V dnešní době mají lidé strach poskytovat jakékoliv informace. Celkový nápad přišel všem zajímavý, ale zatím poněkud hůře představitelný, spíše ho vidí jako vizi do budoucna co se normálního používání týče. Potenciál v těchto aplikacích viděli všichni, jen za určitých podmínek. Největším přínosem vidí zpříjemnění používání a větší efektivitu uživatelů. Většina dotazovaných by rovněž viděla použití informací k adaptaci logiky nebo chování aplikace a ne jen UI a textů, což je zajímavý poznatek.

6.3 Dotazník

Cílem dotazníku je zjistit od případných uživatelů názor a jejich zalíbení v adaptivních aplikacích. Tím je myšleno získání jejich hodnocení od 1, nejhorší, do 10, nejlepší, na jednotlivé funkce a chování frameworku, jako je například adaptace textu v závislosti na emocích nebo jiné věci. Díky tomu bych měl získat obecný názor nejen na framework jako celek, ale rovněž na jeho jednotlivé části a fungování. Dotazník může pomoci i v budoucí práci, úpravách a rozšiřování frameworku, protože budu vědět, jaké metody jsou u uživatelů přijatelnější a které se jim líbí a díky tomu se na ně budu moci zaměřit. Zároveň je cílem zjistit, jestli uživatelé souhlasí, že by poskytovaná funkčnost byla pro ně přínosem a také jakým.

Pro vyhodnocení dotazníku byla stanovena hypotéza, kterou je následující hodnotící stupnice pro výsledky bodovaných otázek. Hodnoty v intervalu od 1 do 4 vyjímaje budou chápány jako negativní výsledek. Hodnoty od 4 včetně do 7 vyjímaje budou chápány jako neutrální. Zbylé hodnoty od 7 včetně do 10 včetně budou chápány jako pozitivní. Cílem je dosáhnout co nejpozitivnějších hodnot.

6.3.1 Výsledky dotazníku

Ke každé otázce v dotazníku jsou zprůměrovány hodnoty nebo popsané jejich rozdělení v seznamu níže. Odpovídalo celkem 14 respondentů ve věku kolem 20 let, pouze 2 respondenti ze všech měli více než 40 let. Respondenti byli nakombinování i co se zkušeností s programováním týče, jelikož bylo cílem získat co nejširší a nejobecnější informace, protože by se mohlo stát, že programátoři budou více naklonění podobným novým věcem než například lidi, kteří nemají s programováním žádné zkušenosti.

- Máte již zkušenosti s adaptivními aplikacemi?

  Ano - 6 (42,9 %), Ne - 5 (35,7 %), Nevím - 3 (21,4 %)

- Používáte nějaké adaptivní aplikace?

  Ano - 4 (28,6 %), Ne - 6 (22,2 %), Nevím - 10 (55,6 %)

- Máte zkušenosti s programováním?

  Ano, programuji v zaměstnání / škole. - 6
  Mám o tom povědomí, ale nikdy jsem neprogramoval/a. - 3
  Mám základy ze školy / práce. - 4
  Nemám žádné zkušenosti. - 1

- Aplikace, která je adaptována (změněna) na míru uživateli, je příjemná na používání.

  Hodnocení od 1 do 10. Průměr: 7,93. Odchylka: 0,95.

- Aplikace mění některé informativní nedůležité texty (například uvítací obrazovka) na základě vašich emocí. (smutek, radost, smích, znechucení, ...)

  Hodnocení od 1 do 10. Průměr: 7,79. Odchylka: 1,28.

- Aplikace mění některé texty na základě jiných informací spojených s uživatelem než jsou emoce. (počasí u uživatele, okolní ruch, umístění uživatele, okolní světlo,...).

  Hodnocení od 1 do 10. Průměr: 8,29. Odchylka: 1,43.

- Aplikace mění některé texty na základě informací získaných z aplikace, nezávisle na uživateli (zprávy, počasí, stav baterie,...).

  Hodnocení od 1 do 10. Průměr: 7,71. Odchylka: 1,51.

- Aplikace snímá moje emoce přes kameru pouze pro lokální použití k adaptaci.

  Hodnocení od 1 do 10. Průměr: 6,21. Odchylka: 1,76.

- Aplikace snímá moje okolí a informace o něm pro získání dat k adaptaci.

  Hodnocení od 1 do 10. Průměr: 7,9. Odchylka: 1,56.

- Změny na základě informací z textu.

  - Změna barvy textů.
    Hodnocení od 1 do 10. Průměr: 7,57. Odchylka: 1,57.
  - Změna barvy pozadí některých prvků.
    Hodnocení od 1 do 10. Průměr: 6,79. Odchylka: 1,31.
  - Změna velikosti textu.
    Hodnocení od 1 do 10. Průměr: 5,86. Odchylka: 1,71.
  - Změna velikosti celého bloku.
    Hodnocení od 1 do 10. Průměr: 6,29. Odchylka: 1,86.

- Některé adaptace z předešlé části (barva textu, velikost bloku) by se prováděly i na základě jiných informací získaných z textu (formálnost, důležitost, spisovnost).

  Hodnocení od 1 do 10. Průměr: 6,64. Odchylka: 1,27.

- Aplikace mění svůj vzhled na základě negativity či pozitivity textu v ní.

  Hodnocení od 1 do 10. Průměr: 6,93. Odchylka: 1,24.

- Aplikace mění svůj vzhled (uživatelské rozhraní) na základě celkového významu textů v ní (pozitivita, negativita, formálnost, ...).

  Hodnocení od 1 do 10. Průměr: 6,86. Odchylka: 1,02.

- Aplikace mění svůj vzhled na základě celkového významu textu a i na základě informací o uživateli.

  Hodnocení od 1 do 10. Průměr: 7. Odchylka: 1,14.

- Aplikace mění texty tak, aby se nezměnil jejich význam, ale byly například pozitivnější.

  Hodnocení od 1 do 10. Průměr: 6,36. Odchylka: 1,88.

- Aplikace mění texty tak, aby se nezměnil význam, ale byly například pozitivnější a to i s ohledem na data od uživatele, například jeho emocí.

  Hodnocení od 1 do 10. Průměr: 5,93. Odchylka: 2,38.

- Aplikace mění texty tak, aby se nezměnil význam, podle jiných hodnot než pozitivita a negativita, například spisovnost.

  Hodnocení od 1 do 10. Průměr: 7,29. Odchylka: 2,1.

- Dokážete si představit aplikaci s výše zmíněnými prvky?

  Rozhodně ano - 4, Spíše ano - 8, Nevím - 2, Spíše ne - 0, Rozhodně ne - 0

- Myslíte si, že je výše zmíněná funkcionalita aplikace dobrá a má budoucnost?

  Rozhodně ano - 4, Spíše ano - 7, Nevím - 3, Spíše ne - 0, Rozhodně ne - 0

- Napadá vás lepší využití nebo kombinace jednotlivých adaptací a použití informací než byly zmíněny výše?

  Ano - 3 (21,4 %), Ne - 7 (50 %), Nevím - 4 (28,6 %)

- Pokud jste v předchozí odpovědi dali "Ano", můžete prosím napsat váš nápad? (Volná otázka)

  - Změna celkové struktury aplikace, podle prováděné aktivity (Jízda autem, sport...)
  - Co kontrolovat pravdivost zpráv rozšiřovaných na sociálních sítích například pomocí umělé inteligence. Následně pak můžou být příspěvky označovány v závislosti na objektivnosti.
  - Například složité systémy jako je excel používají uživatelé na různé úrovni. Takové aplikace by se těmto uživatelům mohli snadno přizpůsobit.

- Mají adaptivní aplikace budoucnost?

  Rozhodně ano - 4, Spíše ano - 9, Nevím - 1, Spíše ne - 0, Rozhodně ne - 0

- Chtěl/a byste používat adaptivní aplikaci, která by se chovala přesně podle vás?

  Rozhodně ano - 4, Spíše ano - 5, Nevím - 4, Spíše ne - 1, Rozhodně ne - 0

- Vyberte možnosti ve kterých si myslíte, že může být aplikace přínosná při používání.

  Vyberte několik odpovědí.

  - Zrychlení práce s aplikací - 6
  - Větší efektivita používání - 7
  - Zpříjemnění používání - 11
  - Zjednodušení používání - 8
  - Zpřehlednění aplikace - 7

- Adaptivní aplikace a jejich využití. (Volná otázka)

  - Osobně mám rád například funkci AIMPu, který pokud je na příém světle zesílí kontrast a podsvětlení, aby byly texty v aplikaci čitelné. Obecně
  - Adaptivní aplikace využívající kontext uživatele mohou společně se světem IoT pomoci zlepšit nakupování výrobků, sbírat metadata o používání aplikací a dále je vylepšovat nebo zlepšit služby pro uživatele s indispozicí fyzickou či mentální.
  - Adaptivní aplikace mohou být zpětnou vazbou pro uživatele.

- Využívání informací o uživateli a jeho prostředí. (Volná otázka)

- Obecně se mi nelíbí umělé pokřivování reality uživateli, ale souhlasím, že pro určitou skupinu je to dobrý nápad, viz jak výše uvedeno pro děti. Tudíž jsem hodnotil kategorii trochu rozpolceně.
- Je to základním prvkem principu context-awareness a je jenom na vývojářích, aby data z kontextu zlepšily používaní aplikace natolik, že uživatelům nebude vadit určitá ztráta soukromí.
- Tady vidím problém, který se dá nazvat "Oni nás ledují .... "

- Využívání informací z textu v aplikacích. (Volná otázka)
  - Již nyní se sbírají metadata o textech v aplikací například pro marketingové kampaně. Uděláme jenom dobře, pokud tyto informace využijeme ke zlepšení interakce člověka s počítačem z důvodů uvedených v odpovědích výše.
  - Musí fungovat stylem "Pomoz a zapomeň"
  - Tady vidím problém, který se dá nazvat "Oni nás ledují .... "

6.3.2 Vyhodnocení dotazníku

Následující tabulka ukazuje průměry všech odpovědí a jejich střední hodnoty. Očíslované podle pořadí v jakém se vyskytují výše.

Otázka	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
Průměr	7,93	7,79	8,29	7,71	6,21	7,79	7,57	6,79	5,86
Medián	8	8	8,5	8	7	8	7,5	7	5,5

Otázka	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17
Průměr	6,29	6,64	6,93	6,86	7	6,36	5,93	7,29
Medián	6,5	7	7	7	7,5	7	6,5	8

Tabulka 6.1: Průměry a střední hodnoty jednotlivých odpovědí

Z tabulky 6.1 byl zjištěn celkový průměr **7,013**, celková střední hodnota středních hodnot **7** a průměrná odchylka (středních hodnot) **0,61**. Na základě čísel a předem stanovené hypotézy můžeme říci, že uživatelé hodnotili průměrně dobře, tedy framework se jim líbil. Avšak číslo je na hraně s neutrálním intervalem, čemu přispívá i průměrná odchylka. Výsledkem tedy je spíš hodnocení na hraně neutrální a pozitivního s větším přiblížením k pozitivnímu hodnocení.

Důležitým faktorem, který je potřeba si všimnout, že převážně neutrální hodnocení, v jednotlivých případech výjimečně i negativní, dostávali především otázky ohledně adaptace textu. Za to adaptace rozhraní dostávali celkové spíše pozitivní hodnocení. Podobně odměřený přístup k adaptaci textu byl zaznamenán i v interview, proto je k této adaptaci třeba přistupovat opatrně, jelikož hodně lidí dnes odmítá cenzuru, pod kterou si můžou takovouto adaptaci představit. Na druhou stranu nejnižší hodnocení 5,86 dostala změna velikosti textu, což by se dalo vysvětlit tím, že se uživatelé bojí, že by se aplikace mohla stát příliš nekonzistentní a rozhozenou.

6.4 System Usability Scale - SUS

SUS[27] je rychlá a jednoduchá metodika jak otestovat použitelnost aplikací. Obsahuje deset hodnotících otázek od „Rozhodně nesouhlasím“ po „Rozhodně souhlasím“. Každá otázka je následně ohodnocena od 0 do 4. Pro vyhodnocení se hodnoty odpovědí sečtou a vynásobí hodnotou 2.5, tím je získán výsledek v rozmezí od 0 do 100. Tento výsledek ale neznamená procentuální hodnotu, ale spíše něco přirovnatelné k percentil. Na základě průzkumů, hodnoty nad 68 jsou brány jako pozitivní pro použitelnost[27]. Čím vyšší je číslo, tím lepší je výsledek. Ze všel posbíraných výsledků se následně provede střední hodnota a tu budu porovnávat. Výpočet sumy jednotlivých testů vychází z ohodnocení otázek od 0 po 4. Po nasbírání dat se každá sudá otázka, která je negativní, invertuje, to znamená její hodnota se odečte od 4. Až po této operaci se provede součet a je získána výsledná suma. Sumy se nakonec zprůměrují a získá se střední hodnota.

Otázky:

1. Myslím, že bych takovou aplikaci chtěl používat pravidelně.
2. Myslím že taková aplikace bude zbytečně složitá.
3. Myslím, že by byla aplikace jednoduchá na používání.
4. Myslím, že bych potřeboval pomoci nebo poradit, abych mohl používat takovou aplikaci.
5. Myslím si, že různé funkce můžou být v aplikaci dobře zakomponované.
6. Myslím, že by taková aplikace byla moc nekonzistentní.
7. Myslím, že by se většina lidí naučila používat takovou aplikaci velmi rychle.
8. Myslím, že by používání takové aplikace bylo velmi těžkopádné.
9. Myslím, že bych si byl velmi jistý při používání aplikace.
10. Myslím, že bych se musel naučit hodně věcí než bych mohl aplikaci používat.

6.4.1 Vyhodnocení

Pro vyhodnocení se každá otázka převede na hodnotu 0 až 4 (rozhodně nesouhlasím až rozhodně souhlasím) a následně se u každé sudé (negativní) otázky provede inverze hodnoty. Následně se sečtou hodnocení všech otázek a vynásobí se hodnotou 2.5. Výsledky tohoto testování lze vidět v tabulce 6.2.

Cílem bylo získat hodnotu aspoň 68, která se udává jako průměr použitelnosti, tedy cokoliv nad tuto hodnotu je lepší. V tabulce lze vidět poměrně široké spektrum hodnot od nejnižší 47,5 po nejvyšší 90. Relevantním výsledkem v tomto testu je průměr nebo spíše střední hodnota, jelikož průměr může

být ovlivněn právě jednou nebo dvěma odchylkami v testování. Průměrná hodnota tedy je **67,92** a střední hodnota je **71,25**. Jak lze vidět průměr je díky dvou odchylkám nižší než střední hodnota, ale jen i přesto je jen těsně pod hranicí 68. Střední hodnota je nad touto hranicí, ale ne o moc.

Díky tomuto testu mohu říci, že by aplikace používající funkcionalitu z této bakalářky byly pro uživatele použitelné, v pozitivním slova smyslu, neboli by byli pravděpodobně spokojeni s používáním podobných aplikací.

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	SUMA
3	0	2	3	3	1	1	2	1	3	47,5
3	1	2	3	3	2	1	3	2	3	57,5
2	1	1	1	2	2	3	3	2	1	45
3	2	3	2	4	2	1	2	3	3	62,5
3	3	3	3	4	3	3	4	2	3	77,5
4	2	3	4	3	3	3	3	2	4	77,5
3	2	2	3	3	3	4	4	3	3	75
4	4	3	2	4	3	3	3	3	4	82,5
4	3	3	3	4	3	3	3	3	3	80
3	2	3	1	2	2	3	2	2	1	52,5
2	3	3	3	3	2	3	3	2	3	67,5
4	3	3	4	4	4	4	4	3	3	90

Tabulka 6.2: Převedené hodnoty - SUS

Kapitola 7

Závěr

Cílem práce bylo navrhnout framework, který bude využívat kontextové informace o uživateli a text-mining k adaptaci aplikace. Obsahem také měla být analýza problému, implementace a testování. Bylo potřeba provést analýzu jednotlivých podproblémů a na jejich základě provést návrh.

Z analýzy vyplynulo rozdělení frameworku na 2 části. V jedné, definované adaptaci, se využívá kontext uživatele pro adaptaci textu, zatímco v druhé, automatické adaptaci, se využívá především kontextu získaného z text-miningu textů aplikace pro adaptaci uživatelského rozhraní a rovněž textů.

Toto rozdělení nakonec bylo velmi přínosné, jelikož poskytlo dvě možnosti pro programátora. Jednu, kdy má programátor úplnou kontrolu nad adaptací a druhou, kdy se zase nemusí programátor o adaptace tolik starat a ušetří mu čas. Rovněž poskytli poměrně rozšířené možnosti pro provedení adaptací na základě kontextových informací.

Implementace frameworku vychází z návrhu, ale je spíše prototypem, který má ukázat možnou strukturu implementace tohoto návrhu a také základní funkcionalitu. Jelikož se během implementace vyskytly problémy s importem Stanford CoreNLP, implementace automatické adaptace není zcela úplná a používá pouze náhodné data pro napodobení kontextu, aby ukázala styl fungování adaptace.

Na základě celkového návrhu a analýzy bylo provedeno testování s uživateli, jehož cílem bylo odhalit spokojenost uživatelů s tématem a jednotlivými funkcionalitami frameworku, tedy styly adaptací a získávání dat. Z jeho převážně pozitivních výsledků bylo zjištěno, že se práce ubrala správným směrem a rozhodně má potenciál pro budoucí rozšíření a kompletní implementaci. Největším problémem s testování je snímání uživatelského kontextu, které budí nedůvěru v aplikace, a také adaptace textu, které se pravděpodobně zdá uživatelům nekonzistentní.

Výsledkem této práce je tedy kompletní analýza problému. Následně podrobný návrh frameworku a jeho dvou částí, ke kterému je implementován prototyp frameworku a ukázková testovací aplikace. Celkový návrh frameworku je otestován uživateli s převážně pozitivním výsledkem a důležitou zpětnou vazbou od respondentů.

7.1 Budoucí práce

Budoucí práce na tomto frameworku nabízí široké možnosti rozšíření. Nejdůležitější bude zprovoznit analýzu sentimentu z textů pomocí Stanford CoreNLP. Jak bylo zmíněno, mohla by se analýza přesunout na server nebo do cloudu. Díky tomu by bylo možné dokončit automatickou adaptaci textu, tedy správně vybírat synonyma, tak aby nebyl změněn význam věty a rovněž aby výsledná adaptace byla opravdu lepší než původní text.

Dalším potenciálem by mohlo být rozšíření kontextu v automatické adaptaci. Jednak by šel rozšířit větší analýzou textu a získat tak jeho formálnost nebo spisovnost. Zároveň by však šel obohatit o uživatelův kontext. Tím by se dala získat zcela optimální adaptace jak uživatelského rozhraní, tak adaptace textu. Rovněž by s touto změnou bylo třeba patřičně rozšířit pravidla pro adaptace, které by tak byly mnohem složitější.

V definované adaptaci by šlo framework rozšířit o další informace z uživatelova kontextu. Také je zde potenciál rozšířit framework tak, aby neměnil pouze texty, ale rovněž i celá View. Tento návrh byl několikrát navrhnut během interview a objevil se i v dotaznících.

Nejdůležitější části po zprovoznění získávání textového kontextu je ale upravit adaptace tak, aby byli robustnější a kvalitnější. Také by bylo lepší optimalizovat nastavení adaptací a přidat možnost adaptovat více prvků různými adaptacemi zároveň. Rovněž vidím prostor pro optimalizaci frameworku co se výkonu týče, aby byl rychlejší a svižnější.

Co se samotného návrhu, tak na něm se může pouze rozšiřovat, to ostatně potvrdilo i testování, díky kterému jsem byl schopen potvrdit, že je práce směřována dobrým směrem.

Příloha A

Literatura

[1] MOYERS, STEPHEN. Understanding the Potential of Adaptive User Interfaces [online]. 2018 [cit. 2018-05-09]. Dostupné z: https://speckyboy.com/adaptive-user-interfaces/

[2] Gartner Says Worldwide Sales of Smartphones Grew 9 Percent in First Quarter of 2017. In: Gartner [online]. Egham, U.K., 2017, May 23, 2017 [cit. 2018-01-13]. Dostupné z: https://www.gartner.com/newsroom/id/3725117

[3] RAMACHANDRAN, Krish. Adaptive user interfaces for health care applications. From the developerWorks archives [online]. 2009, 2009, 1-3 [cit. 2018-01-12]. Dostupné z: https://www.ibm.com/developerworks/web/library/wa-uihealth/

[4] HASSENZAHL, Marc a Noam TRACTINSKY. User experience: a research agenda. Behaviour and Information Technology [online]. 2006, 25 (2): 9197 [cit. 2015-12-06]. DOI: 10.1080/01449290500330331. ISSN 0144929x. Dostupné z: http://www.tandfonline.com/doi/abs/10.1080/01449290500330331

[5] FERENC, Jakub. Kontextualizace a definice User Experience Designu. Academia [online]. , 2-5 [cit. 2018-01-12]. Dostupné z: https://www.academia.edu/19563748

[6] 2013 Annual IEEE India Conference (INDICON) [online]. IEEE, 2013 [cit. 2018-01-13]. ISBN 978-1-4799-2275-8. Dostupné z: http://ieeexplore.ieee.org/document/6726014/

[7] HELD, Albert; BUCHHOLZ, Sven; SCHILL, Alexander. Modeling of context information for pervasive computing applications. Proceedings of SCI, 2002, 167-180.

[8] KORPIPAA, P., J. MANTYJARVI, J. KELA, H. KERANEN a E.-J. MALM. Managing context information in mobile devices. IEEE Pervasive Computing [online]. 2003, 2(3), 42-51 [cit. 2018-01-13]. DOI: 10.1109/MPRV.2003.1228526. ISSN 1536-1268. Dostupné z: http://ieeexplore.ieee.org/document/1228526/

[9] TAUBERER, Joshua. What Is RDF. In: XML.com [online]. July 26, 2006 [cit. 2018-01-13]. Dostupné z: https://www.xml.com/pub/a/2001/01/24/rdf.html

[10] IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second [online]. IEEE, 2004 [cit. 2018-01-13]. ISBN 0-7695-2106-1. Dostupné z: http://ieeexplore.ieee.org/document/1276903/

[11] TAN, Ah-Hwee, et al. Text mining: The state of the art and the challenges. In: Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases. sn, 1999. p. 65-70.

[12] HEARST, Marti. What is text mining. SIMS, UC Berkeley, 2003.

[13] COHEN, K. Bretonnel a Lawrence HUNTER. Getting Started in Text Mining. PLoS Computational Biology [online]. 2008, 4(1), e20- [cit. 2018-01-13]. DOI: 10.1371/journal.pcbi.0040020. ISSN 1553-734x. Dostupné z: http://dx.plos.org/10.1371/journal.pcbi.0040020

[14] Dataset definition. Meloda [online]. 2018 [cit. 2018-05-03]. Dostupné z: http://www.meloda.org/dataset-definition/

[15] ROUSE, Margaret. Data set. What is? [online]. 2016, 2016 [cit. 2018-05-03]. Dostupné z: https://whatis.techtarget.com/definition/data-set

[16] LIU, Bing. Sentiment analysis and opinion mining. Synthesis lectures on human language technologies, 2012, 5.1: 1-167.

[17] SAIF, Hassan, et al. Evaluation datasets for Twitter sentiment analysis: a survey and a new dataset, the STS-Gold. 2013.

[18] ABATE, Tom. Stanford algorithm analyzes sentence sentiment, advances machine learning. Stanford Engineering [online]. 2013 [cit. 2018-05-03]. Dostupné z: https://engineering.stanford.edu/magazine/article/stanford-algorithm-analyzes-sentence-sentiment-advances-machine-learning

[19] SOCHER, Richard, Alex PERELYGIN, Jean Y. WU, Jason CHUANG, Christopher D. MANNING, Andrew Y. NG a Christopher POTTS. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank [online]. [cit. 2018-05-03]. Dostupné z: https://nlp.stanford.edu/sentiment/

[20] Stanford CoreNLP [online]. [cit. 2018-05-03]. Dostupné z: https://stanfordnlp.github.io/CoreNLP/

[21] TURNEY, Peter D. Mining the web for synonyms: PMI-IR versus LSA on TOEFL. In: European Conference on Machine Learning. Springer, Berlin, Heidelberg, 2001. p. 491-502.

[22] VESELÝ, Tomáš. Automatic Synonyms Creation [online]. Praha, 2014 [cit. 2018-05-04]. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/24398/F3-BP-2014-Vesely-Tomas-prace.pdf. Bakalářská práce. Czech Technical University in Prague. Vedoucí práce Ing. Jan Šedivý, CSc.

[23] HAYES, Jane Huffman; DEKHTYAR, Alex; SUNDARAM, Senthil. Text mining for software engineering: how analyst feedback impacts final results. In: ACM SIGSOFT Software Engineering Notes. ACM, 2005. p. 1-5.

[24] Thesaurus [online]. [cit. 2018-05-13]. Dostupné z: http://www.thesaurus.com/

[25] Affectiva Documentation [online]. [cit. 2018-05-13]. Dostupné z: https://knowledge.affectiva.com/docs/using-the-emotion-sdk

[26] GSON - github [online]. [cit. 2018-05-13]. Dostupné z: https://github.com/google/gson

[27] System Usability Scale (SUS). Usability [online]. [cit. 2017-12-29]. Dostupné z: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

[28] Usability Evaluation Methods. Usability.gov [online]. [cit. 2018-05-19]. Dostupné z: https://www.usability.gov/how-to-and-tools/methods/usability-evaluation/index.html

Příloha B

Příklady