Mendelova univerzita v Brně
Provozně ekonomická fakulta

Komponenta pro analýzu telefonních hovorů

Diplomová práce

Vedoucí práce:
Ing. Jan Přichystal, Ph.D.

Bc. Jan Matoušek

Brno 2016

Na tomto místě bych chtěl poděkovat panu Ing. Janu Přichystalovi, Ph.D. a kolegům z firmy Phonexia s.r.o. za odborné vedení, čas strávený konzultacemi, cenné připomínky a rady při tvorbě diplomové práce. Poděkovat bych chtěl také své rodině a přátelům za jejich neustálou podporu.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Komponenta pro analýzu telefonních hovorů** vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*
Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.
Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 19. května 2016 .....

Abstract

Matoušek, J. Software component for analysis of speech recordings. Diploma thesis. Brno: Mendel University in Brno, 2016.

This diploma thesis deals with a design and the implementation of web application module for visualization of data mined from speech recordings. The theoretical section of this thesis is focused on data mining concepts which are followed by specifics of multimedia mining and speech processing. A web application module for visualization of data extracted from speech recordings is designed in next sections. This tool also allow a custom data grouping. Designed module is afterwards implemented and few examples of typical use cases are demonstrated and described.

Keywords

Speech technologies, web application, data visualisation, visual analytics

Abstrakt

Matoušek, J. Komponenta pro analýzu telefonních hovorů. Diplomová práce. Brno: Mendelova univerzita v Brně, 2016.

Diplomová práce se zabývá návrhem a implementací nástroje na vizualizaci výstupů technologií pro analýzu nahrávek mluvené řeči. V úvodní části je popsáno dolování dat a jeho specifika pro audio záznamy mluvené řeči. Následuje návrh nástroje, který umožňuje nejen grafickou prezentaci získaných dat, ale i jejich volitelné seskupení a kombinování za účelem podpory vizuální analýzy dat. Na konec je navržený nástroj implementován a prezentován na několika ukázkových případech užití.

Klíčová slova

Řečové technologie, webová aplikace, vizualizace dat, vizuální analýza

Obsah

Seznam obrázků

1 Úvod a cíl práce

1.1 Úvod do problematiky

Běžný člověk v dnešní době stráví přibližně třetinu dne ve svém zaměstnání. Další třetinu dne zabere spánek. Na ostatní aktivity má zbylou třetinu dne. Odečtemeli však ještě dobu strávenou každodenní rutinou jako doprava, nakupování, jídlo, či hygiena, zjistíme, že volného času opravdu moc nezbývá, a proto se ho snažíme různými způsoby šetřit.

Jedním ze způsobů šetření našeho času je nakupování zboží a služeb prostředky elektronické komunikace, často za využití telefonu. Kontaktní centra takto denně odbaví ohromné množství hovorů. Nicméně mnoho z nich je pro ně neúspěšných. Klienti si produkt, či službu nezakoupí, a tak jsou tyto hovory pro kontaktní centrum proděleč né. K dosažení větší úspěšnosti, potažmo efektivity, je nutná kontrola práce operátorů. Běžným způsobem takové kontroly je znovu poslechnutí si nahrávek hovorů supervizorem a upozornění na případné chyby ze strany operátora. Tento způsob je ovšem značně neefektivní. Představuje ohromné úsilí, ale ve výsledku je vzhledem k velkému objemu hovorů stejně možné zkontrolovat průměrně pouze 1 až 3 % všech hovorů.

Na tomto místě do hry vstupují technologie pro analýzu řeči. Ty mají poměrně širokou škálu aplikací – od identifikace mluvčího, odhad věku, pohlaví, až po hledání určených slov v nahrávce. Jejich vhodnou automatizací je pak kontaktní centrum schopné kontrolovat 100 % hovorů. Už jen díky jejich využití mohou kontaktní centra dosáhnout zajímavé úspory nákladů v podobě redukce počtu supervizorů. Další neméně atraktivní úspory mohou přijít správnou analýzou dat dostupných z řečových technologií.

Zde ovšem často nastává problém. Dat z různých technologií pro zpracování řeči je veliké množství, které je ještě umocněno objemem zpracovávaných nahrávek. Člověk se navíc přirozeně jen velmi obtížně orientuje v rozsáhlých číselných souborech, kterými výstupy řečových technologií jsou. Případná analýza v podobě hledání podobností či vzorů je tak téměř nemožná.

A právě s pomocnou rukou k řešení tohoto problému přichází tato práce. Jak bylo řečeno, člověk se velmi obtížně orientuje v sadě číselných hodnot. Je ale prokázané, že pokud mu předložíme obrázek s odpovídající vizualizací stejných dat, je více než pravděpodobné, že na něm okamžitě identifikuje případné podobnosti či vzory. Lidský zrak s mozkem tak funguje jako velmi mocný a výkonný analytický nástroj.

V této práci tedy bude navržen a vytvořen nástroj na vizualizaci výstupů vybraných řečových technologií, který bude sloužit jako podpora pro rozhodování supervizora.

1.2 Cíl práce

Tato práce si klade za cíl návrh a vývoj vizualizačního modulu, jež svým začleněním rozšíří platformu Speech Analytics o grafickou prezentaci výstupů technologií pro zpracování mluvené řeči. Umožní tak formou grafů prezentovat zvolená data, což povede k výraznému zvýšení uživatelské přívětivosti a urychlení práce uživatele při analýze výsledků. Grafická prezentace také umožní uživateli provádět základní vizuální analýzu nad daty. Ta vhodnou aplikací umožní identifikovat případné trendy, podobnosti či vzory v datech bez nutnosti dodatečných statistických analýz. Implementovaný modul se tak stane dalším důležitým zdrojem dat pro podporu rozhodování supervizorů v kontaktních centrech.

Níže jsou vyjmenovány kroky vedoucí k dosažení cílů práce:

1. Seznámení se se strukturou a možnostmi platformy Speech Analytics
2. Nastudování současných trendů vývoje webových aplikací
3. Analýza stavu a stanovení požadavků
4. Návrh a implementace řešení
5. Zhodnocení provedené implementace

2 Metodika

Tato práce má za úkol vytvořit nástroj, který umožní supervizorům kontaktních center analyzovat dostupná data. Účelem analýzy má být hledání možných vzorů v datech, zkoumání klientské základny a identifikace nastalých nestandardních událostí. To vše za účelem rozšíření znalostní báze k optimalizaci práce operátorů. Současně tento nástroj bude nutné vhodně zasadit do již existující platformy Speech Analytics.

Aby bylo možné takový nástroj úspěšně navrhnout, je nutné stanovit si logickou posloupnost kroků, jež povede k naplnění stanoveného cíle.

Úvodním krokem, než proběhne jakákoli analýza, bude postupné seznámení se s problematikou spojenou s doménou kontaktních center. Porozumění doméně kontaktních center je klíčovým předpokladem k dalšímu postupu v této práci. Bez pochopení jejích základních problémů by mohlo dojít při následné komunikaci k nechtěným nedorozuměním vedoucím k nevhodnému návrhu nástroje. V druhém kroku se zaměřím na seznámení se se strukturou a možnostmi platformy pro řečovou analytiku společnosti Phonexia s.r.o. – Speech Analytics. Jelikož výsledek této práce má být začleněn do platformy Speech Analytics, bude nutné detailní pochopení softwarové architektury platformy i jejího celkového konceptu. V souvislosti s platformou bude dále nutné nastudování nabízených řečových technologií, zejména z nutnosti pochopení jejich výstupů a způsobu použití. S ohledem na doménu a na základě zjištěných informací o analytické platformě budu zkoumat možnosti vizualizace dat řečových technologií. Zmíněný proces bude iterativního charakteru a bude zahrnovat diskuze s některými z uživatelů platformy Speech Analytics za účelem nalézt vhodné požadavky na vizualizační nástroj. Klíčovým krokem, jenž vychází ze všech předcházejících, bude vlastní návrh vizualizačního nástroje (modulu). Následuje samotná implementace, testování a zhodnocení práce.

3 Získávání znalostí z dat

Problém získávání znalostí z dostupných dat se řadí mezi nejstarší problémy, kterým lidstvo čelí téměř po celou dobu své existence. Tento problém, na rozdíl od jiných, je možné sledovat ve všech jednotlivých etapách vývoje lidstva. Například již první lovci pozorovali jednoduché vzory v chování různých zvířecích druhů, jako třeba sezónní migrace, a na základě takto získaných znalostí se dokázali přizpůsobit tak aby přežili. Jiným, mnohem pozdějším, příkladem mohou být farmáři. Ti rozeznávali rozdíly v úrodě určitých plodin na různých místech. Jednoduchým příkladem z dnešní doby může být pekařství. To se na základě předchozích zkušeností rozhoduje, který druh pečiva, kdy a v jakém objemu péct. (Witten et al., 2011)

Zde je důležité poznamenat, že problém získávání znalostí z dat a jejich využití je úzce spjat se zkoumanou doménou i cílem, jehož chce pozorovatel zkoumáním dostupných dat dosáhnout. Toto je patrné i z dříve uvedených příkladů – viz lovci či pekařství.

Ačkoli některá období vývoje lidstva vedla k útlumu či rozvoji oblastí aktivity lidstva, problém pozorování, shromažďování a analýzy dostupných dat přetrval až do dnešní doby. Ba co víc, v posledních desetiletích se z něj stává všeobecně velmi aktuální problém pro téměř jakoukoli oblast lidské působnosti.

Postupně totiž přecházíme do informační doby, respektive doby dat (Han et al., 2012). V této době jsme na každém kroku neustále zaplavováni ohromným množstvím nových dat. Příval dat zajišťuje dnešní elektronika. Přístroje, ať už osobní – mobilní telefony, počítače, nositelná elektronika, nebo externí – různé senzory, kamery, mikrofony neustále generují další a další data. Například údaje o naší poloze, běžných činnostech, komunikaci a aktivitách – to vše obalené časovými záznamy. Takto každodenně generujeme i vstřebáváme ohromné množství dat, která se zpravidla někde uchovávají a čekají na své zpracování.

Tento stav nás dostává do přesně opačné situace, než nastávala v minulosti. Dříve totiž pokud člověk chtěl cíleně hledat informace v datech, často narážel na problém nedostatku požadovaných dat, ze kterých by mohl vyvozovat závěry. Dnes zmíněné neplatí. Díky elektronice existuje nepřeberné množství dat, navíc i historické záznamy na datových nosičích. Objevuje se tedy spíše nový další problém – výběr vhodných záznamů, které použijeme pro zkoumání dat. Výběr a příprava vhodných dat předchází získání znalostí požadovaného charakteru ze zkoumaných dat dané oblasti.

Dalším přímo souvisejícím, ale neméně palčivým problémem je množství dat. Pro zpracování dnes dostupných objemů dat často nestačí běžně dostupné počítače, proto je potřeba hledat dostatečně výkonné výpočetní jednotky.

Za účelem analýzy dat a překonání výše popsaných problémů vznikla oblast „data mining“ – někdy uváděná jako KDD (Knowledge discovery from data) (Han et al., 2012), tedy jako proces získávání znalostí z dostupných dat.

3.1 Dolování dat (data mining)

Pojem dolování znalostí z dat (nebo pouze dolování dat) se objevuje v osmdesátých letech minulého století. Od té doby prošla tato oblast velmi prudkým vývojem, který ani v posledních letech neutichá. Pro dolování dat existuje nepřeberné množství vzájemně velmi podobných definic. Většina z nich se však shoduje na následujícím: dolování znalostí z dat je proces hledání vzorů v datech (Witten et al., 2011). Důležitá je podmínka říkající, že nalezené vzory musí být srozumitelné a přínosné, bez toho by výsledky procesu dolování dat ztrácely smysl. Přínosnost znamená, že nalezená pravidla by měla přinést informace, které pro nás znamenají nějakou výhodu. Zpravidla se výhodou rozumí ekonomický prospěch. Ruku v ruce s definicí procesu dolování znalostí jdou i další omezující podmínky: data musí být uložena v elektronické podobě, zpracování masivních objemů dat, zpracování je automatické či poloautomatické.

Účelem dolování dat je tedy zpracovávat masivní objemy dat dostupné v různých druzích úložišť, hledat v nich zajímavé vzory, potažmo nové informace, a tím přispět k transformaci datových úložišť na znalostní bázi.

Proces dolování znalostí z dat

V některých zdrojích je data mining uváděn jako samostatný krok procesu hledání vzorů v datech (získávání znalostí). Nejinak tomu je i v následujícím popisu procesu dolování dat. Ovšem zpravidla je pojem dolování dat chápán jako celý proces hledání vzorů.

Pokud se na proces hledání zajímavých vzorů v datech podíváme zeširoka a zobecníme jej, zjistíme, že jej můžeme rozdělit do sekvence několika kroků (Han et al., 2012):

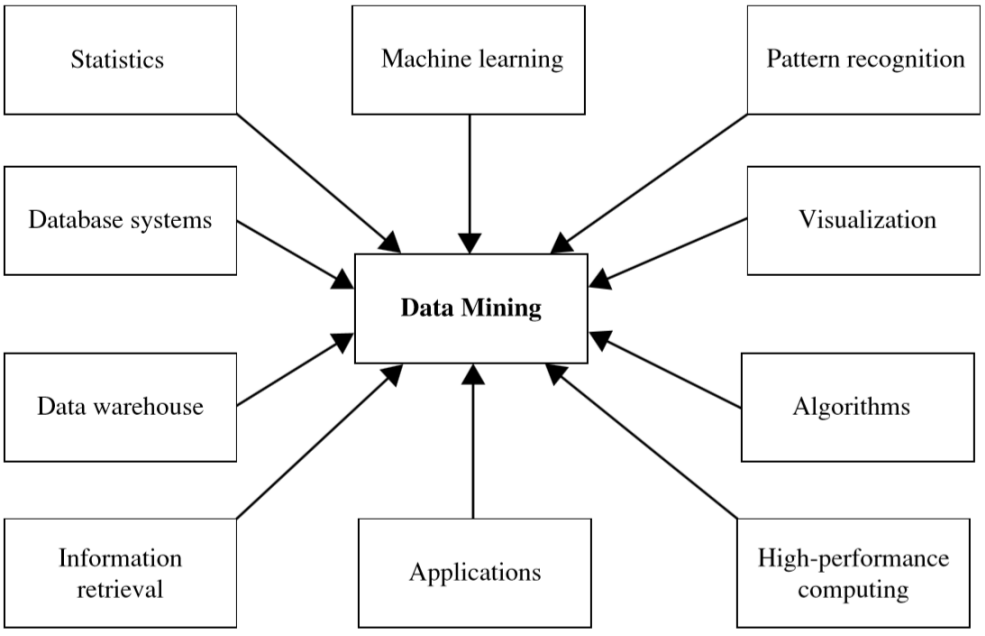
1. Čištění dat – odstranění šumu a prázdných hodnot
2. Integrace dat – sloučení různých zdrojů dat
3. Selekce dat – výběr dat pro danou analýzu
4. Transformace dat – nutná příprava pro vybranou analýzu
5. Data mining – hledání vzorů v datech
6. Vyhodnocení nalezených vzorů
7. Prezentace výsledků[1]

V dnešní době je možnost aplikace procesů dolování znalostí z dat téměř neomezená. Ve skutečnosti se s ní můžeme setkat téměř kdekoli, kde je využita výpočetní technika, což v podstatě znamená, že tyto procesy nějakým způsobem zasahují do všech oblastí našeho života. Pro lepší představu uvedu pár příkladů.

[1] první čtyři body jsou často chápány jako fáze předzpracování

Představme si nakupování v obchodě. Drtivá většina obchodů má zavedené různé věrnostní karty, které používáme při našem nákupu. Tím ale umožňujeme obchodníkovi sledovat naše nákupní zvyklosti. Výhodou pro nás pak je například personalizovaná nabídka vybraného zboží. Zamysleme se ale více. Pokud si obchodník u každého ze zákazníků spojí informace o tom, co, kdy a kde nakupuje i s našimi demografickými údaji, získá tím obrovskou datovou bázi, na níž pak aplikuje algoritmy dolování znalostí z dat. Z nich pak může získat informace o tom, kdy a jaké zboží se nakupuje, jaké druhy zboží se kupují společně a jaký druh lidí jej nakoupí, aj. Jiným, ne tak zřejmým příkladem jsou internetové vyhledávače. Už dávno neplatí, že by to byly pouze indexovací nástroje, které vyhledají jeden určitý výsledek podle předloženého dotazu. Naopak, na základě dotazu nabídnou možná řešení na základě složitého výpočtu relevance k danému dotazu. Zahrnují například shodnost hledaného výrazu, hledanou tématiku, historii procházení webu aj.

Volnost v aplikaci procesu dolování dat je opravdu veliká. Toto tvrzení dokazují příklady uvedené v předchozím odstavci, kde je aplikován ve dvou zcela rozdílných případech užití, což vyžaduje rozdílný přístup k problému, a co víc ve dvou úplně jiných odvětvích. Metody dolování znalostí z dat použité v uvedených příkladech se jistě také do určité míry liší, i když některé se mohou navzájem překrývat nebo být použity v obou případech. Toto je další důležitou vlastností oblasti dolování dat. Běžně používané metody v procesu dolování znalostí jsou zobrazené na obrázku 1.



Obrázek 1: Metody používané pro dolování dat (Han et al., 2012)

Shrnutím výše uvedených informací dostaneme další důležitou charakteristiku data miningu – jedná se o silně tzv. „application driven“ a současně interdisciplinární disciplínu.

3.2 Dolování znalostí z multimédií (Multimedia mining)

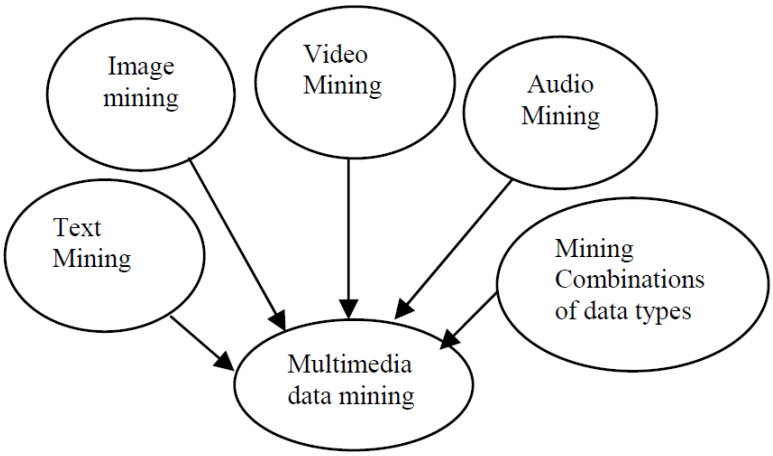
Od počátku vývoje data miningu dochází k jeho aplikacím v rozličných oblastech lidské činnosti. Zejména pak se jeho aplikace soustředily na oblasti bohaté na data, které jsou schopné známé metody zpracovávat. Taková data jsou zpravidla číselného charakteru a jsou uspořádána do pevných jasně stanovených struktur, jež je možné ukládat v běžně používaných úložištích – nejčastěji relační databáze.

Nestrukturovaná data

V posledních deseti letech, zejména díky vývoji oblasti informačních technologií, došlo k masivnímu nárůstu objemu dat uložených v různých úložištích – trend množství ukládaných dat neustále roste. S tímto trendem se logicky objevuje otázka, jsme schopni tato data nějak smysluplně využít a vytěžit z nich zajímavé informace?

Zde se objevuje problém. Umíme vydolovat informace ze strukturovaných dat, díky čemuž jsme tedy schopni zpracovat část nově generovaných dat. Ale co zbytek dat? Drtivá většina, zejména z pohledu objemu dat, je ukládána v jiných stále populárnějších formátech, jedná se zejména o video, hudbu, řeč či text.

S těmito druhy dat a jejich formáty uložení si klasický přístup k dolování znalostí neporadí. Hlavní příčinou je podoba dat – ty jsou zpravidla nestrukturovaná nebo polo strukturovaná. Pro potřeby analýzy těchto druhů dat vznikla nová podoblast, tzv. „Multimédia mining“. Možné dělení této oblasti je zobrazeno na obrázku 2.



Obrázek 2: Dělení multimedia miningu podle (Manjunath et al., 2010)

Multimédia mining je podle (Manjunath et al., 2010) definován jako oblast využívaná pro hledání vzorů, extrakci pravidel a získávání informací z multimediálních databází[2]. Tyto databáze jsou zpravidla reprezentovány různými rozšířeními relačních a objektově orientovaných databází nebo běžným souborovým systémem (Manjunath et al., 2010). Data uložená v multimediálních databázích jsou nestrukturovaná či polo strukturovaná. Jsou reprezentována bitovým proudem, a přestože si

[2]multimediální databáze – zpravidla obsahují data jako text, obrázky, video, audio, aj.

udržují jistou vnitřní strukturu, jsou i tak považována jako nestrukturovaná, jelikož tato struktura není vhodná pro konvenční úložiště. (Manjunath et al., 2010)

Zpracování mediálních dat

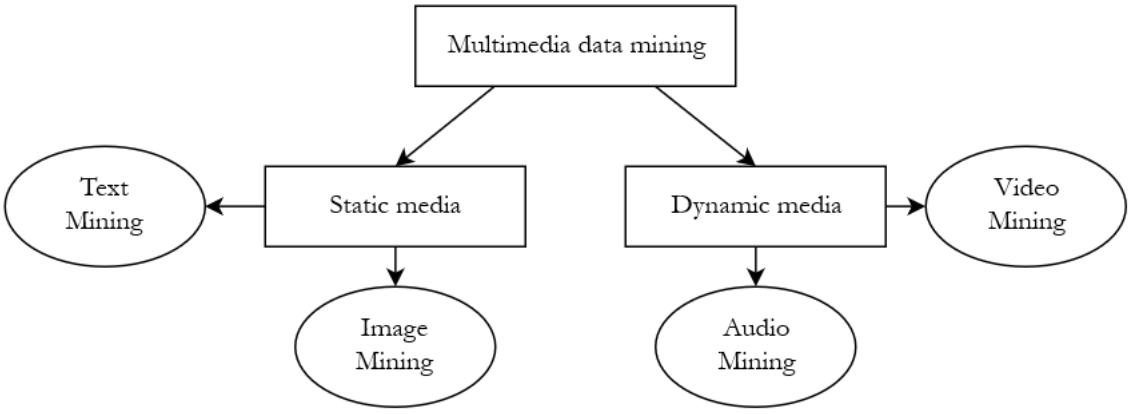
Zpracování mediálních dat, ať už nestrukturovaných či polostrukturovaných je mnohem náročnější úkol, než u dat strukturovaných. Vyšší náročnost zpracování těchto dat plyne zejména z: (Manjunath et al., 2010)

- velkého objemu dat
- vysoké variability a současné heterogenity mediálních dat
- subjektivního chápání těchto dat

Aby vůbec bylo možné mediální data zpracovávat, je nejprve nutné přistoupit k jejich transformaci do takové podoby, která nám umožní další zpracování. V oblasti zpracování velkého množství dat se v uplynulých desetiletí ověřily klasické dataminingové metody. Ty ovšem striktně vyžadují data ve strukturované podobě. Konverze nestrukturovaných dat na strukturovaná je možná, nicméně její přesný popis není možné uvést, jelikož závisí na druhu multimediálních dat. Ruku v ruce s druhem multimediálních dat jde také náročnost této konverze. (Manjunath et al., 2010; Vijayarani et al., 2015)

Rozdělení

Podle (Vijayarani et al., 2015) můžeme multimediální data rozdělit dle jejich povahy do dvou kategorií – statické a dynamické. Určení kategorií a rozdělení specializovaných data miningových oblastí z kategorií plynoucí je na obrázku 3.



Obrázek 3: Kategorie multimedia miningu (Vijayarani et al., 2015)

Speciální podoblastí média miningu, resp. audio miningu je tzv. speech mining neboli zpracování řeči – viz další část.

3.3 Analýza mluvené řeči

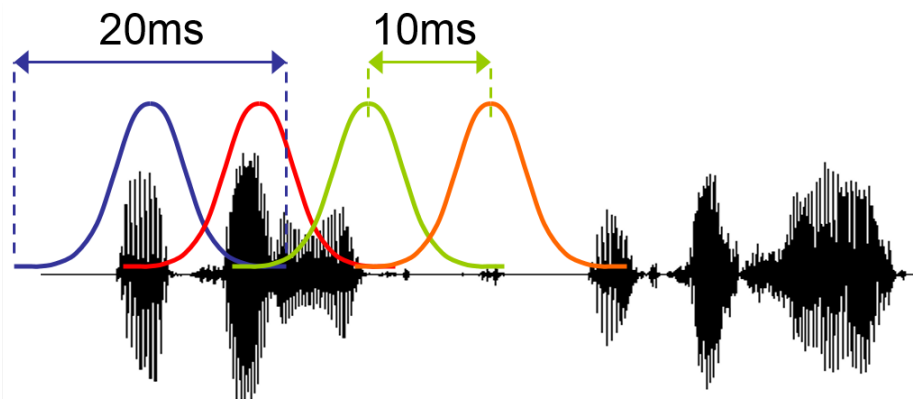
Zpracování mluvené řeči je velmi populární oblast, která postupně hledá a nachází své uplatnění v mnohých odvětvích lidské působnosti. Pod pojmem analýza řeči si většinou představíme přepis mluvené řeči na text, tzv. Speech-To-Text (STT). Ano tato oblast je asi nejdéle zájmem odborníků zabývajících se analýzou řečového signálu a tím pádem i nejznámějším případem užití této oblasti. Nicméně byla by chyba domnívat se, že přepis řeči na text je jedinou oblastí zájmu analýzy mluvené řeči. Dalšími velmi zajímavými aplikacemi vycházejícími ze zpracování řečového signálu jsou např. identifikace mluvčího (Speaker identification – SID), určení jazyka mluvčího (Language identification – LID), určení pohlaví (Gender identification – GID), odhad věku (Age estimation – AGE) a jiné. Už jen z rozsahu znalostí, jež je možné z mluvené řeči získat, je jasné, že pole užití je poměrně široké. Od přepisu diktovaného textu, přes hlasové ovládání, vzdálenou verifikaci na základě hlasu, po určení jazyka volajícího na tísňové linky a v závislosti na to přiřazení odpovídajícího operátora a mnoho dalších možných využití.

Zpracování

Prvním nezbytným předpokladem ke zpracování řečového signálu je jeho pořízení a nezbytná transformace. Zvuk je zachycován mikrofony, jejich výstupem je analogový signál popisující zachycenou řeč. Tento signál je nezbytné transformovat do digitální (číslicové) podoby, která nám následně umožní další zpracování.

Dalším společným krokem pro jakoukoli z výše zmíněných technologií zpracování řeči je tzv. feature extraction, čili extrakce příznaků. Jedná se o posloupnost kroků vedoucích k vydolování důležitých charakteristik signálu. Pokud řečový signál zkoumáme z hlediska velmi krátkých časových intervalů, v řádech milisekund, můžeme jej považovat za statický. Na základě tohoto předpokladu stavíme další analýzu.(Zpracování řečových signálů, 2016; Psutka et al., 2006; Haderlein et al., 2013)

Zvukový signál je tedy rozdělen do sekvence krátkých intervalů tzv. rámců (frames). Aby pouhým rozsekáním signálu nedocházelo ke zkreslení dat, je přistupováno k tzv. windowingu. Ten slouží k vyhlazení zvukového signálu, čehož je docíleno aplikováním vybrané funkce na každý z rámců. Nejčasněji použitou funkcí v tomto kontextu bývá tzv. „Hamming-window function“. K dosažení lepších výsledků je současně použit přístup vzájemného překrytí rámců, každý je dlouhý zpravidla 20 až 30 ms, kde posun mezi rámci činí 10 ms. Rozdělení zvukového signálu je vidět na obrázku 4. Z jednotlivých rámců jsou následně vhodně zvolenou funkcí získány důležité informace pro další zpracování. Extrahované informace, příznaky, jsou pro každý z rámců ukládány v podobě mnoharozměrného vektoru. Uvedeným postupem dojde k získání informací zásadních pro následné zpracování a současně ke značné kompresi vstupního signálu – ta nastává vypuštěním dalších dostupných, ale v tomto kroku nepotřebných dat skrytých v řečovém signálu. (Schwarz, 2008; Matějka et al., 2015; Glembek, 2012)



Obrázek 4: Princip rozdělení zvukového signálu na diskrétní úseky (Matějka et al., 2015)

Algoritmy použité pro extrakci příznaků se liší v závislosti na dalším využití získaných příznaků, tedy jejich využití pro přepis řeči na text, odhadu věku mluvčího, atd. Nicméně některé z nejpoužívanějších algoritmů jsou „Mel Frequency Cepstral Coefficients“ (MFCC), „Shifted Delta Cepstra“ (SDC), „TempoRAl Patterns“ (TRAPs), „Perception Linear Prediction“ (PLP), „Perception Linear Prediction“ (PLP), „I-vectors“ a jiné.

Proces zpracování mluvené řeči je nastíněn ve schématu na obrázku 5. Bloky „Převod do digitální formy“, „Segmentace“ a „Extrakce příznaků“ jsou společné pro všechny technologie zpracování řeči. Blok „Další zpracování“ je obecnou abstrakcí dalšího zpracování extrahovaných dat, které se vždy liší v závislosti na zvolené analýze řečového signálu. Místo tohoto bloku se tak může objevit pouze klasifikátor pro přímé určení výsledků na základě mnohadimenzionálních vektorů příznaků, či složitější struktura bloků reprezentující následné kroky zpracování v závislosti na konkrétní aplikaci. Některé z možných aplikací zpracování řečového signálu i s velmi abstraktním popisem algoritmů použitelných v rámci tohoto bloku je uvedeno níže. Zmíněné metody v daných aplikacích řečové analýzy nejsou jediným možným postupem řešení daného úkolu. Zde uvádím některé algoritmy zvolené při implementaci firmou Phonexia.

Obrázek 5: Abstraktní pohled na zpracování nahrávky řečovými technologiemi

Demografické informace

V některých případech, jako jsou například různé průzkumy prováděné kontaktními centry, je nutné dodržovat určitý poměr dotázaných respondentů. Toto se zpravidla řeší položením otázky přímo během samotného rozhovoru. Tento přístup ovšem

není příliš vyhovující, vyžaduje totiž další činnost operátora kontaktního centra – zaznamenávání těchto údajů – a navíc informace poskytnuté daným respondentem nemusí být záměrně pravdivé.

Pro tyto účely je vhodné využít technologie identifikace pohlaví (GID) a identifikace věku řečníka (AGE). Jejich nasazení následně umožní plně automatizovanou kontrolu rozložení respondentů, současně ulehčí práci operátorům a odpadne nutnost respondentů odpovídat na tyto druhy otázek. V případě GID je pro klasifikaci využíván algoritmus GMM (gaussian mixture model). U AGE je založen na regresních neuronových sítích.

Identifikace mluvčího

Elektronická komunikace, masivně rozšířená v posledních desetiletích, spolu přináší celou řadu bezpečnostních rizik. Jedním z nejkritičtějších je např. vzdálená autentizace a potažmo autorizace mluvčího. Jistě je možné ji řešit zadáváním různých pinů, ale takové řešení je velmi náchylné na prolomení. Nicméně pin může být odcizen, vyzrazen, odposlechnut,... Z toho důvodu je vhodné nalézt takový způsob ověření protistrany, který je co možná nejméně vázaný na nutnost uchování autentizačních/autorizačních klíčů či postupů. Právě tento případ může řešit technologie pro identifikaci mluvčího z řeči (SID). Hlasová charakteristika každého člověka je jedinečná, nedá se tedy odcizit či falšovat. Díky tomu nám také odpadá potřeba pamatovat si piny či jiné autentizační/autorizační postupy. Kromě využití SID pro ověření přístupových práv, existuje další široké pole pro jeho využití. Tím jsou dohledové systémy. Technologie tak může pomoci bezpečnostním složkám najít pohřešované či nežádoucí osoby. SID je založeno na PLDA (Probabilistic Linear Discriminant Analysis)

Obsah promluvy

Případů, kde je nutné zkoumat obsah mluvené řeči, existuje nepřeberné množství. Poměrně jednoduchým příkladem mohou být ovládací systémy založené na předem stanoveném omezeném počtu povelů. Jinými příklady jsou: diktování textových zpráv, automatický přepis pracovní schůzky, či automatické vyhledávání na základě nálezu určeného klíčového slova. Uvedené příklady mohou být řešeny aplikací technologií KWS (Key word spotting – hledání klíčových slov) a STT (Speech-To-Text). Obě technologie jsou založené na akustickém modelování, k němuž jsou zpravidla využívány HMM (Hidden markov models). Následuje prvek zvaný dekodér, který je pro STT prochází pomocí WFST (Weighted Finite State Transducer) mechanismu a aplikuje Viterbi algoritmus. Výsledkem je posloupnost slov. U KWS je pro hledání klíčových slov taktéž použit Viterbi algoritmus, je však aplikovaný na jednodušší model množiny slov. (Psutka et al., 2006)

3.4 Další data miningové metodologie

Nejrozšířenější metodologií je takzvaný „Statistický data mining“. Jedná se o přístup, který si většina z nás představí pod pojmem data mining. Jeho úkolem je zpracování velkého objemu, často i velmi komplexních, ale strukturovaných dat. S tímto přístupem jsou spojovány pojmy jako datové sklady, strojové učení, statistické zpracování, regrese, shlukování a mnoho dalších. Metody popsané dříve v textu patří právě do oblasti statistického data miningu.

Vizuální dolování znalostí (Visual data mining)

Lidská představivost a schopnost hledání různých podobností v libovolných vizualizacích, ať už malovaných obrazech, fotografiích, či zobrazení charakteristik zkoumaných dat, je silnou stránkou člověka. Jsme tak schopni bez jakýchkoli předchozích znalostí identifikovat obecné i některé detailní charakteristiky, které se na první pohled zdají skryté, či dokonce odhalit vzory, jež by bylo výpočetně velmi obtížné hledat. Za tuto vlastnost vděčíme našemu mozku, ten můžeme v této souvislosti chápat jako velmi výkonný, paralelně pracující výpočetní nástroj, jenž nám umožňuje ve spolupráci s očima úkoly podobné povahy řešit. Této vlastnosti se snažili využít i odborníci zabývající se dolováním znalostí z dat. Zavedli proto oblast vizuálního data miningu. (Han et al., 2012)

Podle (Han et al., 2012) můžeme vizuální dolování znalostí definovat jako proces získání implicitních a užitečných znalostí z rozsáhlých datových souborů aplikováním vizualizačních technik a to přímo na data či znalosti z nich získané.

Vizuální dolování znalostí z dat může být realizované následujícími způsoby:

- Vizualizace dat
- Vizualizace výstupů z předcházejících procesů dolování dat
- Vizualizace procesu dolování dat
- Interaktivní vizuální dolování dat

Vizualizace výstupů procesů dolování dat je jedna z oblastí, kterou se zabývá praktická část této práce. Za zmínku také stojí, že oblast vizuálního dolování není přesně vymezena. Zmínky o této oblasti můžeme nalézt v různých zdrojích. Některé z jich, např (Keim et al., 2010), ji uvádějí jako součást tzv. vizuální analytiky (Visual analytics). Definice jsou si však velmi podobné.

Audio mining

Další zajímavou metodologií přístupu k data miningu je audio mining. Jde o aplikaci procesů dolování znalostí z dat, kde výsledky těchto procesů jsou prezentovány formou audio signálů. Představme si to tak, že při analýze dat je přehráván nějaký audio signál, např. nějaká melodie. Pokud je v datech nalezen nějaký vzor, či zajímavá informace, změní se i tento signál. (Han et al., 2012)

4 Speech Analytics platform – SPAS

SPAS[3] je analytické řešení určené pro kontaktní centra vyvíjené firmou Phonexia s.r.o. (dále jen Phonexia). Hlavním cílem platformy SPAS je umožnit automatickou analýzu všech telefonních hovorů provedených kontaktním centrem. Stanovený cíl není nikterak malý, na základě průzkumu dané oblasti bylo zjištěno, že supervizoři kontaktních center jsou schopni manuálně ohodnotit pouze přibližně 1-3 % hovorů. Toto číslo je velmi neuspokojivé, jelikož každý hovor musí splňovat určité náležitosti – ať už určené legislativou, nebo interními směrnicemi kontaktního centra. Nasazení více supervizorů na kontrolu hovorů by vedlo k neúnosnému růstu nákladů. Cena hovoru a nutnost kontroly dodržování pravidel vede kontaktní centra k hledání možných řešení tohoto problému.

S řešením přichází právě platforma SPAS. Ta na základě využití řečových technologií, vyvíjených rovněž firmou Phonexia, umožňuje automatickou analýzu 100 % telefonních hovorů. Dramaticky tak přispívá ke zvyšování kvality a efektivity kontaktních center zatímco současně výrazně snižuje průměrné náklady na hovor.

4.1 Možné případy užití

Mnoho „outbound“[4] kontaktních center často řeší prodej různých produktů. Toho dosahuje uzavíráním takzvaných distančních smluv. Jednou z mnoha podmínek uzavření uvedeného druhu smlouvy po telefonu je, že musí zaznít ústní souhlas spotřebitele s nabízenou smlouvou. Pokud však souhlas nezazní a klient neplní závazky plynoucí ze smlouvy, je taková smlouva jen velmi těžce vymahatelná.

Dalším případem užití je kontrola práce a výsledků kontaktního centra vůči zadavateli telefonické kampaně. Tím se myslí hlavně splnění kvót (např. úspěšnost, cílení na určité skupiny aj.) určených pro danou kampaň.

Platforma SPAS velmi rychle a hlavně plně automatizovaně pomáhá s řešením takovýchto úkolů. V prvním případě umožňuje kontrolou 100 % telefonních hovorů detekovat případy, kdy nedošlo k udělení souhlasu. Díky tomu může kontaktní centrum pružně reagovat a zavčasu převolat takové případy a vyhnout se tak případným budoucím problémům.

V druhém případě platforma umožňuje automatickou kontrolou call scriptu[5] a dalších charakteristik identifikovat slabá místa prodeje, nedostatečně proškolené operátory a nejčastější chyby operátorů. Dovoluje také hlídat rozložení cílových skupin kontrolou demografických údajů klientů u daných hovorů. Kontaktní centrum tak může pružně reagovat, zajistit nutné přeškolení operátorů, úpravu call scriptu a plánovat směny operátorů.

[3] http://spas-solution.com/

[4] outbound – znamená, že kontaktní centrum se specializuje na odchozí hovory, tzn. operátoři obvolávají klienty

[5] call script – soupis vodítek a nutných bodů, jež má operátor během hovoru za úkol splnit. Jinak řečeno chronologická struktura procesu typického hovoru.

Uvedené příklady nejsou rozhodně finálním výčtem, platforma SPAS má mnoho dalších případů užití, vybral jsem však jen tyto dva typické příklady pro objasnění jejího úkolu a přínosu.

4.2 Technologická stránka platformy

Řešení SPAS je klasická webová aplikace typu klient-server. Jádro platformy je psáno primárně v jazyce Java EE. Při vývoji využívá také širokou škálu různých frameworků. Z těch klíčových vyjmenuji: Spring[6] pro budování jádra aplikace, MyBatis[7] pro zajištění ORM komunikace s datovým úložištěm, Wicket[8] pro budování prezentační části.

Vývoj v podobě webové aplikace přináší řadu výhod a odstraňuje problémy vznikající využitím klasické desktopové aplikace. Využití desktopové aplikace by znamenalo nainstalovat SPAS na každý z počítačů, kde bude využíván. To by s sebou neslo i ohromnou režii v podobě nutnosti projít všechny počítače jeden po druhém při aktualizaci verze platformy. Dalším problémem by byl výkon každého z osobních počítačů. Všechny z uvedených a mnoho dalších problémů řeší právě využití webové aplikace. Aplikace je tak nasazena pouze na jednom webovém serveru (v případě SPASu – Apache Tomcat[9]) pro celé kontaktní centrum a všichni uživatelé k ní přistupují vzdáleně prostřednictvím webových prohlížečů. V případě vydání nové verze tak odpadá nutnost obtěžovat jednotlivé uživatele s aktualizací softwaru, vše proběhne na jednom místě, webovém serveru, a uživatel tak není nucen k další obsluze. Odpadá také problém s výkonem hardwaru, který může být slabinou desktopových aplikací. Webové servery jsou dnes v drtivé většině virtualizovány na velmi výkonných strojích, což sebou přináší i možnost velmi pružné škálovatelnosti výkonu. Další velmi podstatnou výhodou webové aplikace je možnost jejího využití na jakémkoli zařízení, které má nainstalovaný nějaký webový prohlížeč. Je tedy možné ji využívat jak na klasickém počítači či notebooku tak i na široké škále dnes velmi populárních přenosných zařízení jako např. mobilní telefony, tablety atd.

4.3 Řečové technologie

Platforma SPAS umožňuje analýzu telefonních hovorů za využití poměrně široké škály řečových technologií. Konkrétně nabízí:

- Rozpoznání klíčových slov na akustické bázi (KWS – Key word spotting)
- Přepis řeči na text (STT – Speech-To-Text)
- Identifikace mluvčího (SID – Speaker identification)

[6]Spring – https://spring.io/
[7]MyBatis – http://www.mybatis.org/mybatis-3/ (dříve známí i jako iBatis)
[8]Wicket – http://wicket.apache.org/
[9]Apache Tomcat – http://tomcat.apache.org/

- Rozpoznání jazyka, potažmo dialektu (LID – Language identification)
- Určení pohlaví mluvčího (GID – Gender identification)
- Odhad věku (AGE – Age estimation)
- Časová analýza nahrávky (TAE – Time analysis extractor)

Všechny výše uvedené patří do rodiny řečových technologií dostupných z knihovny Brno Speech Core (BSCORE) rovněž vyvíjené firmou Phonexia. Knihovna zpřístupňuje dostupné funkcionality prostřednictvím rozhraní Brno Speech Application Interface (BSAPI) (Phonexia, 2015). Vzhledem k tomu, že SPAS je psán v jazyce Java, ale jak BSCORE, tak BSAPI jsou napsány v jazyce C, je nutné pro jejich použití ve SPASu využít ještě další rozhraní, a to BSAPI Wrapper. BSAPI Wrapper je komplexní rozhraní jež obaluje celé BSAPI a vystavuje jeho funkcionalitu pro využití i v jiných jazycích, např. v Javě.

Výsledky zpracování jednotlivých nahrávek řečovými technologiemi je díky implementaci BSAPI Wrapperu ve SPASu možné poměrně jednoduše získat. Jako ukázku uvedu získání výsledků technologie časové analýzy nahrávky:

- getSpeechSpeed() – vrátí rychlost řeči
- getSpeechLength() – délka řeči v nahrávce
- getMaxReaction(int arg0, int arg1) – vrátí maximální délku ticha, respektive maximální dobu reakce (odpovědi) mezi dvěma určenými kanály
- getMaxReactionPosition(int arg0, int arg1) – vrátí pozici maximální délky ticha mezi určenými kanály
- getMinReaction(int arg0, int arg1) – vrátí minimální délku ticha, respektive nejkratší dobu reakce (odpovědi) mezi dvěma určenými kanály
- getMinReactionPosition(int arg0, int arg1) – vrátí pozici minimální délky ticha mezi určenými kanály
- getAvrgReaction(int arg0, int arg1) – průměrná délka reakce mezi dvěma kanály
- getNSpeakerTurns(int arg0, int arg1) – informace kolikrát se změnil mluvčí mezi určenými kanály
- getNCrossTalk(int arg0, int arg1) – vrátí počet skoků do řeči mezi určenými kanály
- getCrossTalkPosition(int arg0, int arg1, int arg2) – vrátí pozici určeného skoku do řeči (viz třetí parametr) v nahrávce
- getNChannels() – počet kanálů v nahráce

4.4 Analýza nahrávky

Hlavní předností řešení SPAS je výstavba analytické logiky nad řečovými technologiemi. Řečové technologie jsou samy o sobě velmi mocným nástrojem pro dolování dat z telefonních hovorů a jiných nahrávek řeči. Nicméně jejich použití je vždy omezeno pouze na jednu nahrávku. Navíc, aby uživatel mohl provést nějaké pokročilé zpracování, musí mít dostupné alespoň základní informace o nahrávce, jako počet kanálů a pořadí kanálů. Používání technologií takovýmto způsobem by bylo velmi zdlouhavé, nekonformní a hlavně neefektivní. Navíc by uživatel vždy musel řešit i výběr nahrávek, které hodlá zpracovat a které již byly zpracovány.

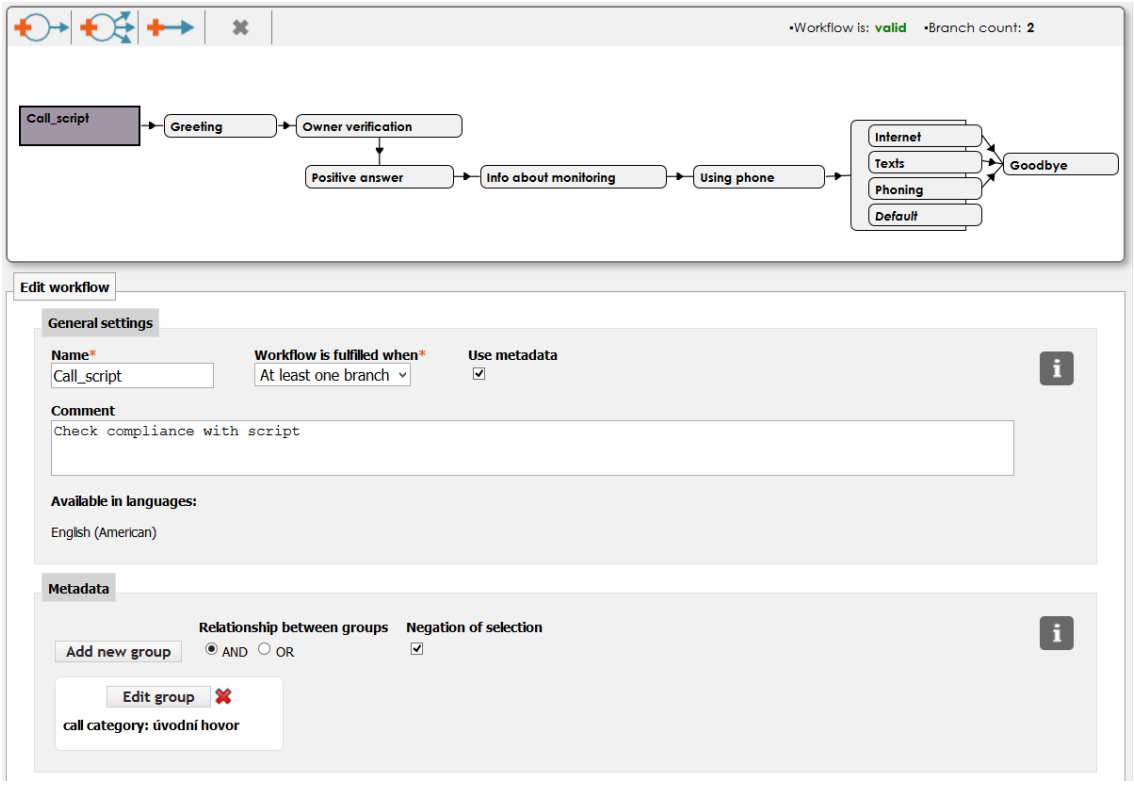
Uvedené problémy řeší platforma SPAS zavedením jednotné logiky pro zpracování nahrávek nazvané „job“. Job si představme jako jakýsi úkol, ve kterém celou řadou parametrů specifikujeme co? kdy? a jak? se má zpracovat. Základním parametrem jobu je obecné určení jaké nahrávky budou zpracovány. Toho je docíleno určením časového období, přesněji určením období, z něhož musí nahrávka pocházet, aby byla daným jobem zařazena na seznam ke zpracování. Druhým kritickým krokem je výběr technologie, jedné nebo více najednou, pomocí nichž budou dané nahrávky zpracovány. Pokud si uživatel přeje zpracovat i analýzu klíčových slov, musí v dalším kroku přiřadit ještě tzv. workflow (specifikuje jaká klíčová slova a kdy hledat – podrobněji bude popsáno níže). Následuje už jen logika ovládání jobu – spuštění, přerušení, zastavení.

Velikou výhodou je flexibilita nastavení zpracování. Konkrétní job může být spuštěn buď pouze jednou pro jednorázové zpracování sady nahrávek, nebo naplánován pro určité období – např. zpracovávat všechny nahrávky, které budou zaregistrovány do systému ve vybraném období. Třetí variantou je spuštění nekonečného automatického zpracování, což znamená, že veškeré nové nahrávky budou daným jobem zpracovány, dokud nedojde k jeho zneaktivnění.

4.5 Workflow

Nejpokročilejší částí řešení SPAS je proces zpracování a prezentace výsledků analýzy klíčových slov. Zásadní roli v tomto ohledu hraje workflow. Pomocí workflow můžeme vybudovat abstraktní reprezentaci očekávaného průběhu hovoru, vůči níž se následně hovory automaticky vyhodnocují. Lze si to představit jako vybudování elektronické reprezentace call scriptu, tedy přesné sady kroků, které operátor musí v každém hovoru dodržet. Pro jednoduchost a názornost při užití je workflow budováno v grafickém editoru. Připodobnit lze k orientovanému acyklickému grafu, kde uzly určují místa analýzy a hrany její posloupnost. Ukázka workflow na obrázku 6.

Jednou z velkých výhod workflow je možnost dodatečného upřesnění výběru nahrávek, které zpracuje. Toho je docíleno využitím metadat poskytnutých k nahrávce. U každého workflow lze specifikovat, jaká metadata musí nahrávka splňovat (jaká metadata jsou nahrávce přiřazena), aby byla zpracována tímto workflow (možné vi-



Obrázek 6: Ukázka konstrukce workflow a základního nastavení

dět ve spodní části obrázku 6). Tato pravidla dodatečné selekce se aplikují až po aplikování pravidel určených jobem.

5 Moderní webová aplikace

Webové aplikace a jejich vývoj je zejména v posledním desetiletí velmi populární oblastí. Mnoho dříve striktně desktopových aplikací se již dávno přesunulo právě do této oblasti. Výhody jsou jasné, centralizace a z ní plynoucí jednotná údržba aplikace, rychlá reakce na bezpečnostní hrozby, jednoduché doručování nových verzí aplikace, možnost využití na jakémkoli zařízení s webovým prohlížečem, dostupnost kdekoli po světě limitovaná pouze nutností připojení k internetu a mnoho dalších. Díky vysoké popularitě jsou webové aplikace také značně dynamickou oblastí. Nové trendy jak v oblasti architektury, tak v oblasti designu aplikací se poměrně rychle objevují a střídají předešlé. Příkladem může být velká pozornost věnovaná před pár lety Ruby on Rails[10], který je, jak ukazují současné trendy, nyní již spíše na ústupu ve prospěch novějším jazykům, jakým je například Node.js[11] (Varghese, 2014; Wayner, 2016). Níže v této kapitole uvádím některé z dnes velmi populárních trendů: Single-page application, API Driven development a Microservices. Při jejich určení jsem vycházel zejména z těchto zdrojů (Varghese, 2014; Winn, 2015; Stangarone, 2016; Wayner, 2016; Peham, 2016).

O webových aplikacích a jejich vývoji bylo napsáno velmi mnoho článků i publikací. Každý z nich se ovšem věnuje této problematice z jiného úhlu pohledu. Nemalá část zdrojů se věnuje metodikám životního cyklu procesu vývoje webových aplikací. Mezi takové patří například (Hurst; Reich, 2012; Lotz, 2013). Obecně se uvedené zdroje shodují na historickém vývoji a dnešních trendech – tedy postupný přechod od vodopádové metodiky, někdy zvané také jako tradiční přístup k životnímu cyklu, přes spirálový přístup a jeho různé kombinace až po dnes velmi populární agilní metodiky vývoje. Podrobnější popis je možné nalézt ve výše zmíněných zdrojích.

Největší část článků a publikací zabývajících se problematikou vývoje webových aplikací se věnuje otázce použitých technologií a jejich srovnání. Pro ukázku lze zmínit například (Laurent, 2014; Kohan). V uvedených lze identifikovat dva hlavní směry – technologie používané pro aplikační backend[12] a technologie pro frontend[13]. Nejzásadnější z hlediska frontendu jsou zejména HTML, CSS, JavaScript a frameworky nad ním postavené. Pro backend, tedy aplikační logiku, lze jmenovat programovací jazyky PHP, Java, C#, Perl, Python atd.

Další části kapitoly se věnuji krátkému popisu technologií využitých v praktické části této práce, popisu některých aktuálních trendů ve vývoji webových aplikací a nakonec výběru vhodné JavaScriptové knihovny pro tvorbu vizualizací v praktické části práce.

[10]Ruby on Rails (RoR) – http://rubyonrails.org/ – je framework určený pro vytváření webových aplikací v jazyku Ruby (https://www.ruby-lang.org/en/)

[11]Node.js – https://nodejs.org/en/ – je na JavaScriptu postavený framework/platforma určený pro vývoj rychlých událostmi řízených webových aplikací

[12]Backend – je část webové aplikace, jež se skrytě na pozadí stará o funkčnost aplikace

[13]Frontend – vizuální podoba webové aplikace, tedy vše co uživatel vidí

5.1 Technologie pro tvorbu webových aplikací

Apache Wicket

Apache Wicket je volně šiřitelný framework, vyvíjený od roku 2004, určený pro vývoj webových aplikací v Javě (Apache Wicket, 2016). Jeho hlavní charakteristikou je komponentově orientovaný přístup, na rozdíl od asi nejrozšířenější MVC architektury. Komponentová architektura se nechá připodobnit ke klasickému přístupu DOM (Document object model), kde jednotlivé části dokumentu jsou uzavřené do párových tagů, ke kterým je následně přistupováno jako k samostatnému objektu s určitými vlastnosti a obsahem. Každý z těchto objektů může obsahovat jiné objekty – části dokumentu. Celý dokument tak reprezentuje stromová struktura vzájemně zanořených objektů. Přístup Wicketu k webové stránce je analogický popsanému principu DOM. Webová stránka se skládá z komponent, které mohou obsahovat libovolné množství jiných komponent. Přičemž každá z nich si uchovává minimálně informace o tzv. rodičovské komponentě a seznam přímo podřazených komponent. To zaručuje vytvoření stromové struktury webové stránky. Komponentou v tomto smyslu rozumíme jakýkoli prvek webové stránky – např. popisek, různá tlačítka či kontejner určený pro shlukování jiných komponent.

Další charakteristikou Wicketu je striktní oddělení prezentační a business logiky (aplikační logiky). Pro prezentační logiku je využit čistý HTML kód, není tak potřeba učit se a do standardního HTML doplňovat jiné, specifické tagy, či aplikační logiku. Jednotlivé komponenty webové stránky a veškerá business logika k nim přiřazená je naopak plně v režii programovacího jazyka Java. Tím je zaručen plně objektový přístup. Objektové komponenty jsou svým HTML reprezentacím přiřazeny atributem HTML tagu nazvaným wicket:id. O datové propojení se starají tzv. Wicket modely.

Samozřejmostí je podpora AJAXu[14] při práci s jednotlivými komponentami webové stránky.

Spring

Spring je široká sada knihoven a frameworků určených k vytváření všech druhů Java aplikací. Interně se dělí na mnoho projektů. Díky využití modulární architektury je možné do vlastního projektu zvolit právě takovou kombinaci, která nám vyhovuje. Z hlediska návrhu a struktury rozsáhlých Java aplikací je klíčová implementace návrhového vzoru Inversion of Control (IoC) v hlavní části Springu. To nám umožňuje přenést vytváření a správu objektů na framework. Zmíněný koncept implementace IoC je obecně známý pod pojmem Dependency Injection (DI). Dalšími důležitými projekty jsou zejména Spring Security – pro zajištění komplexní bezpečnosti aplikace, Spring Boot a mnoho dalších. (Spring, 2016)

[14] Asynchronous JavaScript and XML (AJAX) – https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started

MyBatis

MyBatis je framework určený pro persistentní vrstvu aplikace. Slouží k propojení objektově orientovaných aplikací s různými druhy relačních databází. Využitím mapovacích XML souborů či Java anotací přímo v kódu, dosahuje mapování běžných Java objektů (POJO – Plain old Java object) na databázové záznamy. Veliká výhoda je také v tom, že toto mapování nemusí být ve stylu 1 ku 1 pro sloupce databázové tabulky a atributy Java objektu. Naopak, síla je v určení mapování atributů. Výsledný Java objekt tak může být poskládán z hodnot sloupců mnoha různých databázových tabulek. (MyBatis, 2016)

5.2 Trendy ve vývoji webových aplikací

Single-page application

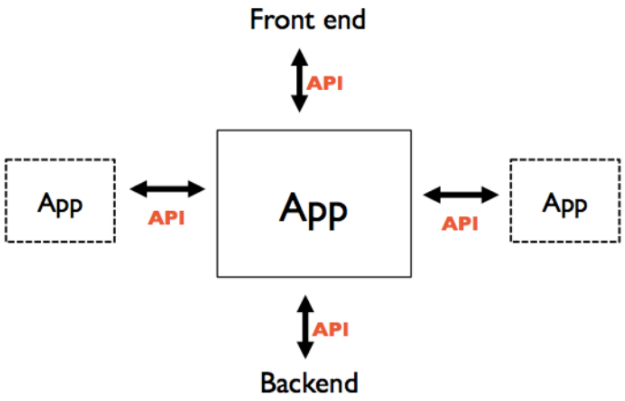
„Single-page application“ (SPA) jsou webové aplikace, které fungují na principu načtení celé struktury webové stránky při prvním požadavku a následně dochází jen k dynamickým změnám jejího obsahu, struktura zůstává neměnná. Je to velká změna oproti tradičnímu přístupu k webovým aplikacím, který spočívá ve vygenerování nové struktury stránky při každém dotazu na server a následném překreslení ve webovém prohlížeči. Aplikováním SPA přístupu dochází ke striktnímu oddělení aplikační logiky (AJAXové dotazy na server), prezentační logiky (HTML) a datové/servisní vrstvy (webový server). Veškerá interakce uživatele s grafickým rozhraním tak probíhá pouze na straně klienta využitím Javascriptu a CSS. V případě potřeby pak pomocí AJAXových dotazů aplikace získá potřebná data ze serveru, jehož role se po prvním vygenerování struktury aplikace mění na servisní/datové rozhraní. Výraznou výhodou takového oddělení logiky je pak možnost libovolné změny struktury stránky bez nutnosti zásahu do aplikační logiky, nebo úplná výměna servisní logiky bez jakékoli změny prezentační a aplikační části aplikace. (Wasson, 2013; Single page apps in depth, 2012)

K dosažení SPA architektonického přístupu je použito zejména funkcionalit dostupných z HTML5, o kterých se dočteme například v (Lubbers et al., 2011), v kombinaci s AJAXovým voláním a značným využitím JavaScriptu, respektive JavaScriptových frameworků na klientské straně.

API Driven development

Více než poslední desetiletí patří ve vývoji webových aplikací architektuře MVC (Model View Controler). Ta rozděluje vývoj aplikací do třech částí: Model – data a logika aplikace, View – prezentační vrstva, respektive uživatelské rozhraní a Controler – mezivrstva propojující Model a View. Tato architektura vyžaduje neustálou komunikaci klient server, kdy každý klientský požadavek je odeslán na server, zde je zpracován a zpět je odeslána nová webová stránka s odpovědí.

S trendem posledních let, kdy dochází k přesunu aplikační logiky na klientskou stranu, hlavně díky použití HTML5 a JavaScriptu, se přístup architektury MVC jeví značně neefektivní a do vývoje webových aplikací přichází trend zaměřený na výstavbu API, tzv. API driven development (viz. (Goteti, 2015; Anuff, 2014)). API driven development je vývojová metodologie, kde aplikační rozhraní se stává centrálním bodem zájmu vývojářů. Na mnoha různých API je totiž založena celá funkcionalita aplikace. Jedno API slouží k obsluze veškerých požadavků z prezentační vrstvy, jiné k získávání dat, další pak k vzájemné komunikaci s různými aplikacemi, atd. Architektonický nástin je na obrázku 7.

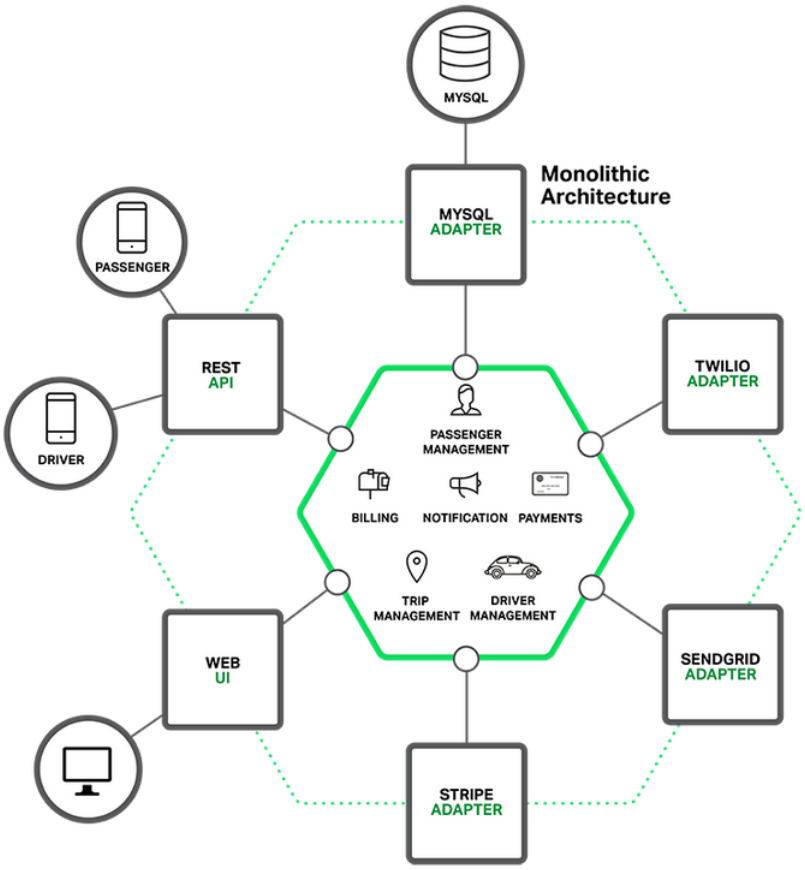


Obrázek 7: Nástin aplikace navržené podle architektury API driven development (Anuff, 2014)

Microservices

De facto standardem vývoje komplexních podnikových aplikací (tzv. enterprise applications) se v minulosti stala monolitická architektura (Richardson, 2014a). Aplikace vybudovaná na základě monolitické architektury je vždy ve výsledku považována jako jednolitá samostatná aplikace. Nic na tom nemění fakt, že v drtivé většině je aplikace bohatě vnitřně strukturovaná do mnoha modulů zastávajících rozlišné role (např. komponenty pro databázový přístup, vystavení aplikačního rozhraní, uživatelské rozhraní, atd.). I tak se navenek aplikace tváří jako jeden celek, jenž obsluhuje veškeré požadavky.

Novým trendem posledních let je architektura založená na mikroslužbách (microservices). Ta spočívá ve vývoji aplikace jako množiny malých, nezávisle spustitelných modulárních služeb, v níž každá služba funguje samostatně a komunikuje skrze dobře navržený, odlehčený mechanismus, aby splnila stanovené cíle (Huston). Rozložení komplexní podnikové aplikace do mikroslužeb má oproti monolitické architektuře řadu výhod: umožňuje samostatný vývoj jednotlivých služeb, spuštění služby jen pokud je potřeba, oddělenou škálovatelnost a mnoho dalších. Rozdíly modulární architektury a architektury mikroslužeb jsou patrné na obrázcích 8 a 9.



Obrázek 8: Monolitický návrh aplikace pro taxi službu (Richardson, 2015)

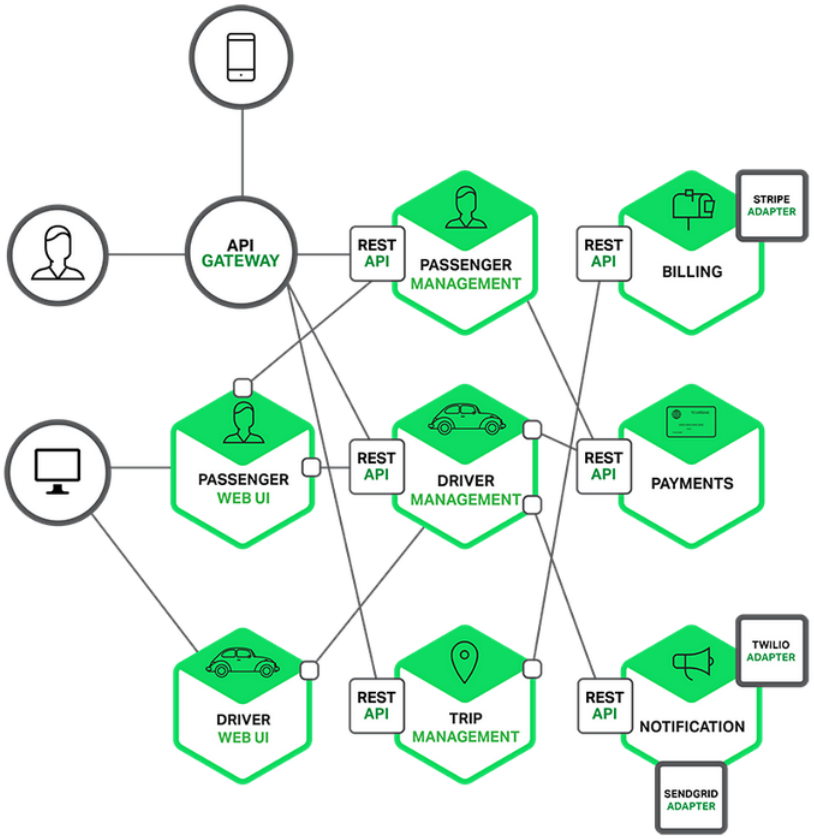
K nasazení aplikace vytvořené podle architektury mikroslužeb na webový server existuje řada přístupů. Mnoho z nich je možné nalézt v (Richardson, 2016; Richardson, 2014b). Tím asi nejpopulárnějším přístupem je dnes využití principu kontejnerů. Každá instance služby je spuštěna v samostatném kontejneru a jejich řízení je svěřeno nadřazené službě pro správu kontejnerů – například Docker[15].

Jiným, nicméně velmi podobným, přístupem k architektuře podnikových aplikací, který se rozšířil v posledním desetiletí, je architektura orientovaná na služby (Service oriented architecture – SOA).

5.3 Výběr vizualizačního nástroje

Jednou z klíčových částí navrhovaného vizualizačního modulu je právě technologie či nástroj, který použijeme pro vytváření grafických reprezentací zvolených dat. Vybraný nástroj musí umožnit jednoduchou integraci do existujícího projektu, musí být pro programátora jednoduchý, přehledný a co možná nejvíce usnadňovat práci

[15]Docker – https://www.docker.com/ – nástroj pro správu aplikačních kontejnerů



Obrázek 9: Návrh aplikace pro taxi službu na základě architektury založené na mikroslužbách (Richardson, 2015)

při vytváření vizualizací. Současně by měl být využívaný, tzn. existují referenční implementace.

Nástrojů pro vizualizaci dat dnes existuje nepřeberné množství, je proto potřeba postupovat obezřetně při výběru vhodného kandidáta. V případě této práce provádím výběr na základě splnění několika základních kritérií, subjektivního hodnocení naplnění kritérií popsaných v úvodu této části dokumentu a konzultace s vedoucím vývoje projektu. Zvláštní důraz jsem kladl na zmíněnou možnost, co nejjednodušší integrace do řešení SPAS a to zejména z pohledu času programátora i architektonického návrhu. Přihlížel jsem také k různým doplňkovým funkcionalitám některých knihoven.

Základními kritérii, které musí vybraný nástroj bezpodmínečně splňovat, jsou:

1. licenční požadavky – volný na použití v komerčním prostředí
2. schopnost svižně pracovat s větším množstvím dat (řádově tisíce hovorů)
3. interakce (možnost ovládání) – posouvání os, po najetí myší schopnost zobrazit popis dat, události po klinutí do grafu aj.
4. přizpůsobení (customizace) – popisky os a jiné

D3.js

D3.js (D3.js – Data-Driven Documents) je populární vizualizační JavaScriptová knihovna postavená na práci s dokumenty založenými na datech. Základním kamenem jejího úspěchu je, jak již naznačuje název D3 (Data-Driven Documents), daty řízený přístup k práci s dokumenty reprezentujícími webovou stránku, tzv. DOM[16]. K vizualizaci dat tato knihovna používá zejména HTML, CSS3 a SVG[17]. Díky kombinaci třech zmíněných přístupů k vizualizaci je D3.js velmi rychlá i pro rozsáhlé datasety. Vstupní data mohou být ve formátu JSON[18], či XML.

Výhodou pro některé vývojáře může být její značná podobnost s dalšími DOM frameworky – jQuery[19] a Prototype[20], což může z počátku velmi usnadnit orientaci v kódu a značně urychlit učící křivku a tedy i její využití. Další výhodou je značná volnost v upravování a výstavbě vlastních grafů – je možné určit vlastní specifické nastavení hlavní struktury grafu, jako jsou rozměry os, ale i pro nejmenší detaily jako typ, barva, či hustota mřížky na pozadí grafu. D3.js umožňuje vytvářet interaktivní vizualizace, jejichž chování je možné prostřednictvím JavaScriptových callbacků upravovat vzhledem k potřebám uživatele.

Mezi hlavní nevýhody patří zejména problémy s kompatibilitou ve starších prohlížečích a fakt, že knihovna nenabízí žádné předem připravené grafy. Pokud tedy chceme vykreslit libovolný graf, musíme jej vždy celý vyspecifikovat. Z toho plyne značná náročnost na objem psaného kódu pro vytvoření vizualizace a tedy i výraznou časovou náročnost. D3.js je šířený pod licencí BSD 3-Clause.

Google Charts

Google Charts (Charts: Interactive charts for browsers and mobile devices) je JavaScriptová knihovna určená pro vizualizaci dat na webových stránkách. K vykreslení grafů využívá HTML5, SVG a pro potřeby zpětné kompatibility starších prohlížečů i VML[21]. Google Charts nabízí širokou paletu předpřipravených grafů a značnou volnost v jejich upravování dle přání uživatele. Ke každému grafu také vystavuje několik JavaScriptových událostí, které umožní implementovat vlastní interakci s některými prvky grafu. Možnost vizualizace i větších objemů dat je zajištěna díky využití HTML5. Nicméně standardním dotazováním na Google Charts API je množství dat omezeno maximální velikostí URL. Pro obejití tohoto omezení je nutné využít dotazování na datový zdroj prostřednictvím Google Visualization API.

[16]DOM – Document Object Model – podrobný popis https://www.w3.org/DOM/
[17]SVG – Scalable Vector Graphics – https://www.w3.org/Graphics/SVG/About.html
[18]JSON – JavaScript Object Notation – http://www.json.org/
[19]jQuery – https://jquery.com/
[20]Prototype – http://prototypejs.org/
[21]VML – Vector Markup Language – https://www.w3.org/TR/NOTE-VML

Licenční politika je zaštítěna prostřednictvím Google APIs Terms of Service, které určuje, že s výjimkou jasně vyspecifikovaných okolností je zdarma pro komerční i nekomerční užití.

I když tato knihovna splňuje všechna předem stanovená základní kritéria, byla nakonec z užšího výběru vhodných kandidátů vyřazena. Důvodem vyřazení byla dvě klíčová fakta. Knihovna vyžaduje neustálé připojení k internetu, kvůli potřebě komunikace s Google Charts API pro vygenerování grafů – to je v rozporu strategie platformy Speech Analytics, která je zpravidla nasazena ve vnitřních uzavřených sítích zákazníka. Druhým faktem je vynášení dat mimo uzavřenou síť – pro vytvoření grafu je potřeba poslat vizualizovaná data na servery Googlu, viz. Google Charts API zmíněné výše.

Flot

Flot (Flot: Attractive JavaScript plotting for jQuery) je čistě JavaScriptová knihovna určená pro tvorbu vizualizací ve webových aplikacích využívajících jQuery. Jedná se o minimalistickou knihovnu zaměřenou na jednoduchost použití a rychlost, jak vykreslení, tak psaní kódu pro vizualizaci základních grafů. Knihovna umožňuje mnoho možností uživatelského přizpůsobení – od uživatelského vylepšení stávajících grafů, přes úpravu procesu vykreslení, až po možnost napojení na vlastní kód pro obsluhu uživateli vyvolaných událostí. Dále dovoluje volitelné rozšiřování funkcionality prostřednictvím napojení na řadu externě, či komunitou připravených modulů. Práce s velkým množstvím dat také není díky využití HTML5 „canvas“ problémem, i když pro opravdu masivní datasety bude potřeba využít některý z dostupných pluginů pro předzpracování dat. Licenční politika je řízena licencí MIT.

Flot je velmi zajímavým kandidátem na využití, nicméně jeho hodnocení značně sráží fakt nutnosti využití řady doplňků a potřeby vlastního návrhu a implementace architektury pro řízení uživatelských událostí. To vše sebou nese značné nároky na obsluhu kódu a čas programátora.

Plotly.js

Plot.ly je rozsáhlý projekt zabývající se vizualizací dat (Plot.ly project). Přesněji je to online analytický nástroj nabízející cloudové řešení pro vizuální analýzu dat. Nástroj je možné ovládat buď skrze webový prohlížeč, či RESTové API rozhraní. Nicméně toto řešení je komercionalizované a poskytované za úplatu. Velké překvapení přišlo koncem loňského roku (tj. 2015), kdy se Plot.ly rozhodlo uvolnit k volnému užití, pod licencí MIT, své know-how v oboru vizualizace dat – JavaScriptovou knihovnu Plotly.js.

Plotly.js je vysokoúrovňová deklarativní JavaScriptová knihovna určená pro vytváření grafů ve webovém prostředí. V základu přichází s poměrně širokou paletou předpřipravených grafů, jež je možné využívat. Vytváření grafů je založeno na

knihovnách D3.js, jež byla popsána výše, a na stack.gl[22]. Stack.gl je volně použitelný softwarový ekosystém určený pro vykreslování 2D a 3D grafiky ve webových prohlížečích. Použití stack.gl dovoluje také implementaci vysoké interaktivity vizualizovaných dat. Plotly.js pro vykreslování grafů primárně používá SVG, to je ale limitováno zejména výkonem při vykreslování velmi vysokého počtu objektů – v těchto případech může Plotly.js přepnout vykreslovací logiku a použít WebGL integrovaný skrze stack.gl.

Možnost uživatelské interakce je zajištěna interním JavaScriptovým API vystaveným pro několik různých druhů událostí, jež vznikají uživatelskou interakcí. Toto API umožňuje vlastní implementaci požadovaného chování na základě uživatelem vyvolané události. Asi největší výhodou a přínosem je zmíněný deklarativní přístup ke grafům. To znamená, že nemusíme vytvářet složité JavaScriptové konstrukce, nebo psát zdlouhavé úseky kódu pro vykreslení jednoho grafu. Vše, co je potřeba, je vytvořit jeho JSON popis. Každou jednotlivou část, od typu grafu, přes použité barvy, až po legendu grafu, popíšeme samostatnými JSON objekty. Výsledný graf je pak reprezentován komplexní JSON strukturou složenou z těchto menších objektů. Tu pak jednoduše vezmeme a předložíme knihovně k vykreslení.

Zvolený nástroj

Pro účely vytvoření vykreslovacího jádra vizualizačního modulu navrženého a vytvářeného v rámci této práce jsem se rozhodl použít poslední popsanou knihovnu – Plotly.js. Knihovna nejenže splňuje všechny stanovené základní požadavky, ale navíc nabízí zajímavý deklarativní přístup k práci s grafy. Tento přístup programátorovi umožňuje jednodušší implementaci bez nutnosti přesouvat velkou část aplikační logiky ze serverové do klientské části aplikace. Programátor tak může připravit strukturu grafu a potřebná data přímo ve zvoleném programovacím jazyku na straně serveru a vykreslovacímu jádru pak předat pouze JSON objekt k vykreslení. To vše bez nutnosti psaní velkého objemu JavaScriptového kódu.

[22]stack.gl – http://www.stack.gl

6 Analýza požadavků

Současná podoba platformy Speech Analytics je silným analytickým nástrojem určeným k vyhodnocení hovorů kontaktních center. Dokáže z hovorů získat mnoho statistických údajů (délky nahrávek, změny mluvčího, skoky do řeči, rychlost řeči, rychlost reakce mluvčího, …) a je velmi silný v hledání klíčových slov a analytikou nad nimi. Všechny tyto údaje je schopen přehledně shrnout a zobrazit pro každou z nahrávek. Nicméně někteří uživatelé platformy SPAS poukázali na fakt, že získané údaje je sice možné zobrazit u nahrávky, ale chybí souhrnné zobrazení.

V poslední době, také do řešení SPAS, přibyla podpora nových řečových technologií, konkrétně GID (identifikace pohlaví mluvčího) a AGE (odhad věku mluvčího). Z diskuze s uživateli vyplývá, že i tyto mají značný potenciál pro analýzu hovorů. Problém ovšem je, že výsledky zpracování hovorů nově doplněnými technologiemi zatím nejsou nikde v aplikaci dostupné.

Na základě uvedeného vyvstal požadavek na vytvoření vhodné, uživateli co nejsrozumitelnější, prezentace dostupných výsledků. Při návrhu takového řešení by měl být brán ohled i na první připomínku, tedy umožnit hromadné zobrazení výsledků.

K vytvoření návrhu prezentace výsledků řečových technologií bylo nutné další komunikaci řídit a usměrnit ke konstrukci uceleného obrazu o požadavcích uživatelů. Toto bylo dosaženo zodpovězením dvou zásadních otázek:

1. S jakými daty se bude pracovat?
2. Co od prezentace dat uživatelé očekávají? (resp. V čem by jim měla pomoci?)

První otázku zodpovídají předcházející odstavce. Co se týče druhé otázky, výsledkem její diskuze nebyl jediný závěr, který by jednoznačně stanovil, jaké je očekávání. Naopak na základě konstruktivních návrhů užití formulováno několik otázek, na které by řešená prezentace výsledků měla být schopna odpovědět. Těmito otázkami jsou:

- Kdo je mým zákazníkem? (odchozí hovory)
- Kdy určité skupiny zákazníků volají? (příchozí hovory)
- Kontrola operátora – Průměrný počet skoků do řeči? (odchozí + příchozí hovory)
- Kontrola operátora – Průměrná délka ticha na konci hovoru? (odchozí + příchozí hovory)

7 Návrh řešení

Otevřenou diskuzí mezi uživateli řešení a dalšími zaměstnanci firmy Phonexia nad stanovenými požadavky platformy SPAS se došlo k závěru nutnosti vytvořit univerzální vizualizační nástroj určený pro prezentaci výsledků řečových technologií. Takový nástroj by měl být přímočarý a intuitivní na ovládání. Současně by měl umožnit jednoduché vizualizace dostupných dat formou vhodně zvolených grafů. Z pohledu implementace by diskutovaný nástroj měl být implementován formou modulu do platformy Speech Analytics. Rámec využitelnosti vizualizačního nástroje pak určují zejména otázky: „Jaká data budou vizualizována?“ a „Co se od vizualizace očekává?“, jež byly zodpovězeny v předcházejících odstavcích.

Z uživatelského hlediska by zamýšlený nástroj měl pracovat v několika krocích:

1. Stanovení obecných filtrovacích kritérií
2. Výběr typu grafu
3. Výběr dat
4. Volba agregační funkce
5. Zobrazení výsledků

7.1 Návrh ukázkového případu užití

Představme si situaci, kdy se kontaktnímu centru ozve zadavatel kampaně na průzkum spokojenosti s určitým druhem produktu. Zadavatel požaduje ověření plnění předem stanovených podmínek zastoupení věkových skupin v dané kampani.

V takovém případě přijde na řadu supervizor týmu, který volá danou kampaň. Supervisor se běžným způsobem přihlásí do aplikace SPAS a přepne na stránku s vizualizačním nástrojem. Zde nejprve vyplní sekci obecných filtrovacích pravidel – provede nastavení požadované kampaně a stanoveného časového omezení pro výběr nahrávek. Dále provede určení kanálu nahrávek, který nás zajímá – v tomto případě klientský kanál. Následuje volba typu výsledného grafu – vybereme sloupcový graf. Další krok vede k určení skupiny dat, jež budou zobrazena v grafu – zde zvolíme věk. Posledním krokem je volba vhodné agregační funkce, která bude použita pro zobrazení výsledků – požadujeme kontrolu zastoupení volaných klientů, zvolíme tedy „počet“. Následuje už jen zobrazení grafu s výsledky.

7.2 Návrh grafického řešení

Při návrhu grafického uživatelského rozhraní (GUI) jsem vycházel z výše sepsané posloupnosti kroků, jež reprezentuje celý proces vizualizace dat. Klíčové bylo navrhnout rozhraní a jeho ovládání tak, aby odráželo logickou návaznost kroků zmíněného procesu vizualizace a současně bylo co nejvíce uživatelsky přívětivé, zejména přehledné, jednoduché a lehce ovladatelné. Z toho důvodu jsem při návrhu rozdělil

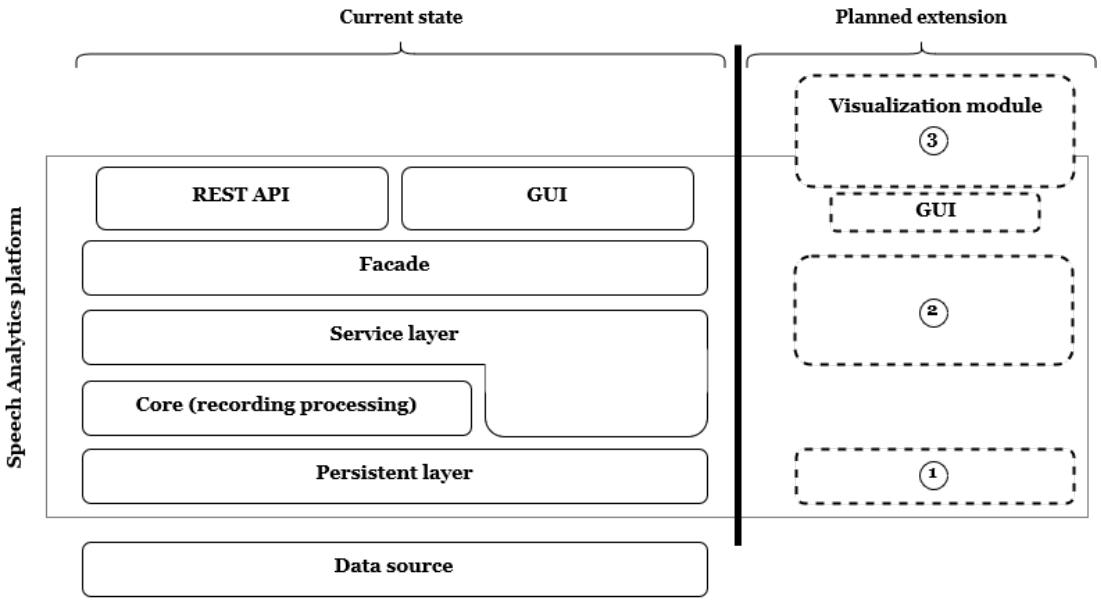
grafické rozhraní do třech pomyslných částí, které reprezentují jednotlivé fáze procesu vizualizace dat. Výsledkem je návrh rozhraní, který je možné vidět na obrázku 22 v příloze A.

První část uživatelského rozhraní by měla sloužit k výběru typu grafu a volitelnému nastavení filtrovacích kritérií pro výběr nahrávek, jejichž výsledky zpracování řečovými technologiemi budou zařazeny do množiny hodnot pro zobrazení v grafu. Druhá část slouží pro volbu druhu dat. Druhem dat je myšlen konkrétní výsledek zpracování řečovou technologií, tedy např. věk mluvčího, délka nahrávky, atd. Současně bude možné v tomto kroku provést i detailnější nastavení povahy zobrazených dat. Například volbu možného seskupení na základě zadané hodnoty. Další možností nastavení v druhé části by měla být kombinace více druhů dat v jednom grafu (to umožní například porovnání věku volajícího na základě pohlaví). Přechodem do třetí fáze získáme výslednou vizualizaci grafu na základě nastavených kritérií v předcházejících krocích. Pouhá vizualizace ovšem není jediný úkol třetí fáze, naopak. Hlavním účelem je zde prostřednictvím jednoduchého ovládání grafu dovolit uživateli hledat potenciálně zajímavé informace v datech.

Navržené uživatelské rozhraní je výsledkem iterativního procesu návrhu a diskuze s uživateli platformy Speech Analytics. Při návrhu jsem musel zohlednit i současný vizuální styl platformy SPAS, do které je modul zasazen. To zejména proto, aby nevznikl modul zcela vybočující z jednotného stylu. Dále byly zohledněny také některé poznatky získané během vývoje datové logiky modulu. Nicméně stále se jedná o jednu z prvních verzí návrhu, která velmi pravděpodobně dozná značných změn na základě zpětné vazby po testovacím spuštění v řešení SPAS.

7.3 Abstraktní návrh architektury

Je jasné, že zamýšlený modul bude poměrně rozsáhlým dílem. Při jeho architektonickém návrhu bude potřeba zohlednit mnoho faktorů jako: univerzální návrh datového nosiče, zpracování dat různé povahy, transformace dat, grafické rozhraní, prezentace výstupů, interakce atd. Nicméně než se pustíme do podrobného návrhu samotného modulu, je potřeba si uvědomit, že je nutné brát v potaz i vnější okolí navrhovaného modulu. Vnějším okolím v tomto ohledu myslím jeho zasazení do platformy SPAS. Na její straně totiž bude potřeba implementovat řadu nutných funkcionalit. Jmenujme ty hlavní – vytvoření jednotné datové struktury pro výsledky zpracování nahrávek řečovými technologiemi, rozšíření persistentní vrstvy pro získávání dat z datového zdroje a logika pro konverzi jednotné datové struktury do univerzální podoby vyžadované navrhovaným modulem. Abstraktní návrh celkové architektury je možné vidět na obrázku 10.



Obrázek 10: Obrázek poskytuje abstraktní pohled na strukturu aplikace znázorňující současný stav, plánované rozšíření i zasazení vizualizačního modulu.

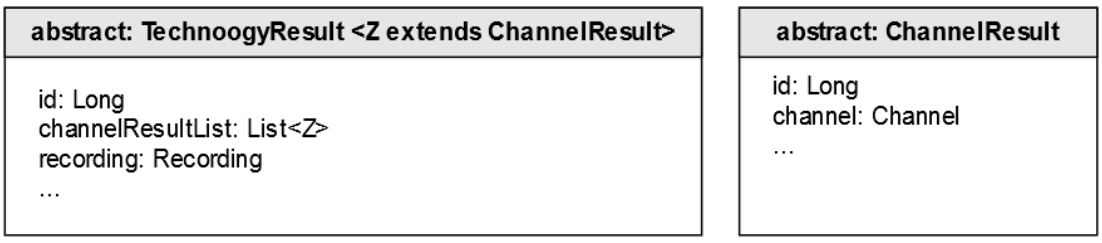
Níže je uvedeno vysvětlení číslování na obrázku 10:

1. Perzistentní vrstva a datová struktura
   Rozšíření persistentní vrstvy. Jak už bylo dříve zmíněno v textu, platforma SPAS disponuje řadou technologií pro dolování znalostí z mluvené řeči. Každá z těchto technologií poskytuje výsledky, které jsou přímo mapovány na různá místa datového úložiště. Nejinak je tomu v aplikaci. Pro každou řečovou technologii existuje v různých balíčcích speciálně určený objekt. V dosavadním přístupu, kdy aplikace zpravidla pracovala vždy pouze s výsledky jedné z technologií, toto nebyl problém. Ovšem ambice navrhovaného vizualizačního modulu toto mění. Požadavek na umožnění práce s výsledky více řečových technologií

najednou a s ním spojená potřeba jednotné datové základny potřebné pro další zpracování vede k nutnosti návrhu univerzální struktury. Taková struktura musí zachovat klíčové charakteristiky identifikující výsledek řečové technologie vůči nahrávce. Těmi jsou:

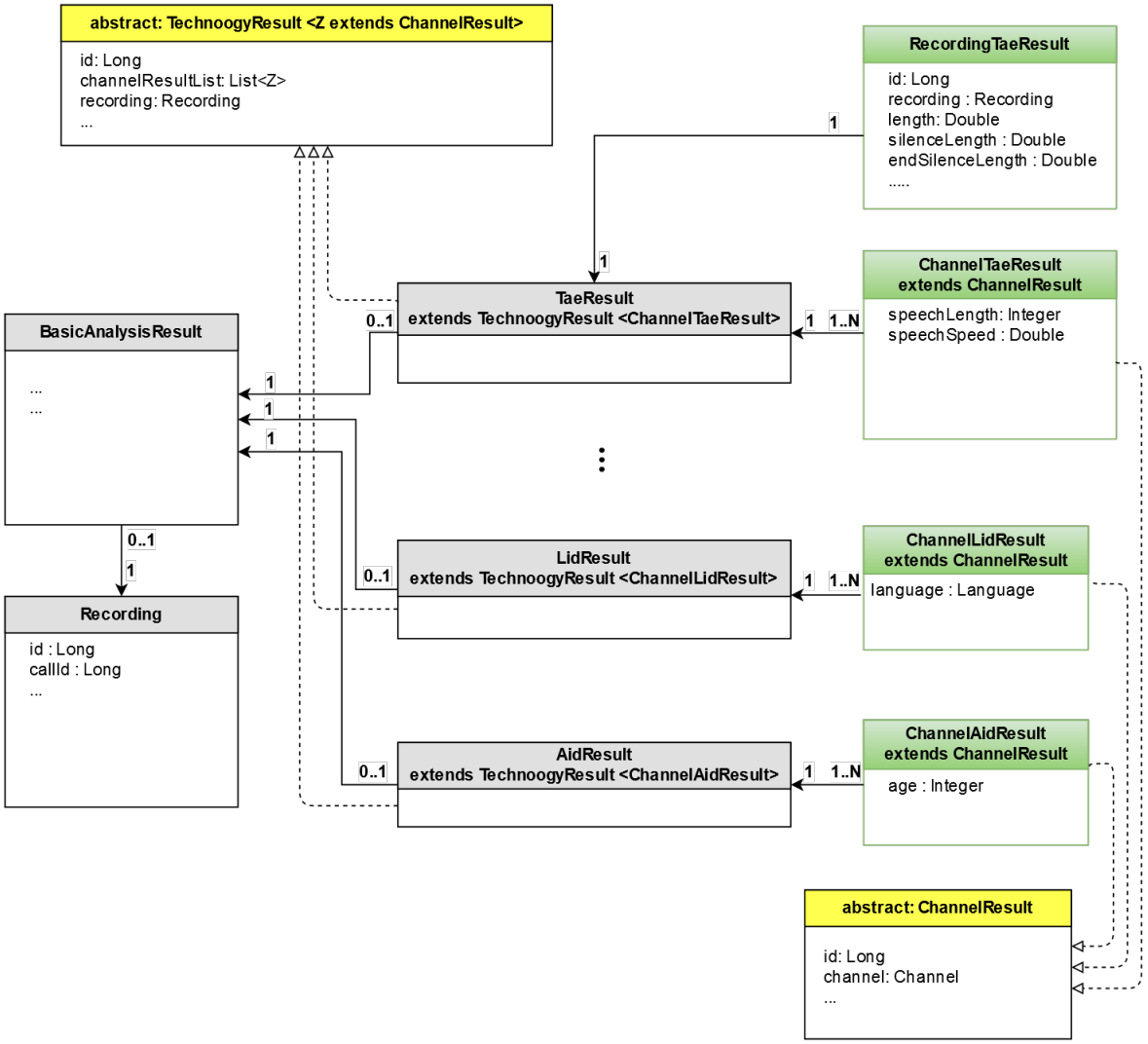
- přiřazení kanálů nahrávce
- určení druhu kanálu
- provázanost výsledků řečových technologií na odpovídající kanály
- přiřazení obecných výsledků řečových technologií na nahrávku jako celek
- možnost práce se specifickými výsledky některých řečových technologií

Dalším požadavkem je jednoduchost a jednotnost práce s výsledky různých technologií. Návrh takové struktury vzhledem k nutnosti značné univerzálnosti dané stanovenými požadavky se zprvu zdál být poněkud problematický. Nicméně díky pestrým možnostem, jež na poli návrhu objektů nabízí jazyk Java, se taková struktura povedla úspěšně navrhnout. Dva základní objekty tvořící kostru požadované struktury jsou vidět na obrázku 11. Základem pro navržení těchto obecných objektů je využití Abstraktních tříd ve spojení s genericitou a dědičností. Abstraktní třída ze své povahy slouží jako vzor pro objekty, které z ní dědí. Její výhodou je možnost nařídit svým potomkům nutnost implementace předem stanovených metod. Současně však umožňuje i implementaci obecných funkcionalit, které její potomci zdědí. Genericita zajistí univerzálnost návrhu. Neklade totiž nutnost určovat typ objektu již u společného předka, určí pouze, že proměnná či kolekce bude daného generického typu. Povinnost přesně specifikovat datový typ přenechává programátorovi implementujícímu třídu určenou generickým typem. Konečně dědičnost v navržené struktuře zaručí předání společných vlastností objektu a typovou bezpečnost. Konkrétně typovou bezpečnost generického prvku Z, jemuž určuje podmínku, že musí být potomkem objektu ChannelResult. Vše bude patrné z následujícího příkladu, použití objektové struktury, viz obrázek 12.



Obrázek 11: Předpis abstraktních tříd TechnologyResult a ChannelResult

Na obrázku 12 je viditelná jasná návaznost nahrávky (Recording) na výsledek řečové technologie (např. TaeResult). Objekt TaeResult využitím genericity současně říká, že kolekce kanálů dané nahrávky musí být typu ChannelTaeResult.



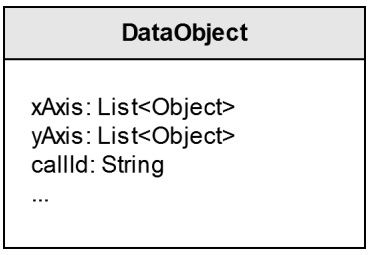
Obrázek 12: Ukázka navržené jednotné objektové struktury výsledků řečových technologií

Toto bez problémů kompilátorem projde, jelikož objekt ChannelTaeResult splňuje podmínku generického typu z objektu TechnologyResult a to, že musí být potomkem objektu ChannelResult. Podobným způsobem bychom implementovali objekty pro výsledky dalších řečových technologií.
Uvedený příklad ukazuje sílu objektově orientovaného přístupu Javy. I poměrně jednoduchými objekty s využitím možností nabízených jazykem lze vytvořit komplexní, typově bezpečnou a současně univerzální datovou strukturu.
Dalším úkolem, jenž se objevuje ve spojitosti s návrhem nové datové struktury je její plnění daty. Tedy správné mapování dat dostupných v datovém úložišti na navrženou strukturu. Touto otázkou se bude zabývat části 8.2.

2. Příprava dat
   Dalším nutným krokem na straně platformy SPAS bude návrh a implementace logiky konverze dat do univerzální datové struktury, jež bude základním datovým zdrojem pro navrhovaný vizualizační modul. Na tomto místě okamžitě vystává otázka, zda-li je nutná další konverze datové struktury. Vždyť předchozí část se věnuje konstrukci jednotné struktury pro výsledky řečových technologií. Odpověď je jasná: ano, konverze je nutná. Hlavním odůvodněním je požadavek na co možná největší zachování nezávislosti navrhovaného modulu na platformě Speech Analytics. Modul je sice v této práci do platformy implementován, ale vzhledem k rozvoji aplikací vyvíjených firmou Phonexia je nutné brát v potaz i možnost jeho budoucího přesunutí, či znovu využití v jiných aplikacích.
   V této části textu však přesný postup konverze popisovat nebudu. Té se bude věnovat sekce 8.3. Zde naopak abstrahuji od detailů a zaměřím se na návrh datové struktury pro vizualizační modul.
   Z pohledu vizualizačního modulu je nutné navrhnout takovou strukturu, která bude reflektovat veškeré jeho požadavky a specifika nutná k docílení výsledné funkcionality. Mezi takové požadavky patří:

   - zachování nezávislosti modulu na datovém zdroji
   - předaná data musí být jednoznačně přiřazena k hovoru, tzn. každý datový prvek musí být samostatně identifikovatelný (zejména kvůli zamýšlené uživatelské interakci s grafem)
   - možnost přenášet dodatečná, v grafu nezobrazená, data navázaná na jednotlivé datové prvky
   - nezávislé spravování dat pro více os grafu (nutnost zajistit i provázanost těchto údajů)
   - datový prvek musí umožnit přenášet více hodnot pro stejnou osu

   Veškeré zmíněné požadavky splňuje objekt navržený na obrázku 13.



Obrázek 13: Objekt pro přenos dat do vizualizačního modulu

Svou povahou je DataObject klasickou Java třídou. V návrhu obsahuje dvě kolekce xAxis, yAxis a proměnnou callId. xAxis a yAxis slouží pro uchování všech hodnot pro zobrazení vztažených k jedné nahrávce a to vzhledem k ose,

na kterou chceme data zobrazit. Kolekce jsou využity z důvodu požadavku přiřazovat jedné nahrávce více dat na vybranou osu. Například k nahrávce budeme chtít na osu X zobrazit současně informaci o věku a pohlaví mluvčího. Omezující podmínkou, nebo spíše povinností přenesenou na programátora při vkládání více dat na jednu osu, je nutnost striktně hlídat pořadí vkládaných hodnot do listu vybrané osy u všech datových prvků. Rozhodnu-li se, že na prvním indexu listu bude informace o pohlaví a na druhém místě informace o věku, není žádoucí pořadí u některých prvků prohodit, protože by docházelo ke zkreslení výsledků, či dokonce nefunkčnosti modulu z důvodu nekonsistence datových typů. Nahrávky, které nemají dostupný některý z požadovaných výsledků, jsou vyřazeny už ve fázi komunikace s datovým úložištěm. Tím předejdeme možné komplikaci s prázdnými prvky listu, či jejich posunutím vůči další hodnotě na zpracovávané ose.
Proměnná callId obsahuje unikátní identifikátor nahrávky v systému. V pozdějších fázích zpracování objektu a vizualizace výsledků se využívá k přiřazení jakékoli hodnoty k odpovídají nahrávce.
V tomto návrhu je uvažováno využití pouze 2D grafů, proto jsou použity pouze kolekce xAxis a yAxis. Pokud bychom chtěli přidat více dimenzí, stačí pouze do třídy DataObject přidat další kolekce, např. zAxis. Analogický postup platí i pro doplňková data. Pokud tak k přenášeným datovým hodnotám chceme zaznamenat další informace, lze postupovat podle stejné logiky jako u datových os. To znamená přidat do DataObject požadovanou proměnnou a postarat se o její naplnění. Je nutné si také uvědomit, že hodnoty zobrazované na osy jsou zpravidla agregované. Proto je potřeba odpovídajícím způsobem přistoupit i k volbě doplňovaných proměnných.

3. Vizualizační modul
Blok vizualizačního modulu bude hlavním jádrem celé práce. Zjednodušeně řečeno, jeho úkolem bude celkové řízení procesu zkoumání dat. Na základě nastavení požadavků na vizualizaci vytvořených uživatelem si zažádá datový zdroj o patřičná data. Získaná data následně dle potřeby zpracuje. Dále provede konstrukci kostry grafu, tu naplní předzpracovanými daty a následně, za využití zvolené JavaScriptové knihovny, provede vykreslení výsledků v uživatelově webovém prohlížeči. V souvislosti s interními pravidly vývoje pro platformu SPAS je kladen důraz na udržení co možná nejvíce kódu aplikační logiky v Javě. Toho bude dosaženo návrhem a implementací vlastního Java wrapperu pro využívanou JavaScriptovou knihovnu. Wrapper využitím zpětných callbacků zaručí přesun implementace logiky do serverové části.
Podrobný popis, včetně přehledného diagramu, celého modulu a jeho funkcionality bude uveden v sekci 8.4.

7.4 Využití optimálních datových struktur a přístup ke zpracování dat

Jedním ze základních předpokladů navrhovaného vizualizačního modulu je práce s většími objemy dat. Počítejme řádově desetitisíce objemných objektů pro zobrazení týdenních záznamů hovorů. V případě měsíců se jedná o statisíce objektů. Důležité je uvědomit si nejen náročnost na paměť, způsobenou hromadným zpracováním takových objemů dat v paměti, ale také náročnost na čas procesoru. Tu můžeme uvažováním značného zjednodušení rozdělit na dvě části – výpočet cílové hodnoty a nutná režie. První jmenovanou, za předpokladu optimálního návrhu algoritmu pro výpočet, nelze nijak zvlášť ovlivnit, pomineme-li možnost rozšíření výpočetních zdrojů. Druhou jmenovanou je nutná režie. Kromě opravdu nutné režie způsobené interním řízením logiky zpracování určené mnoha aspekty práce procesoru, do jejichž diskuze se zde pouštět nebudu, bývají hlavní příčinou nadměrné režie programátorské chyby. Chyb, jichž se programátoři mohou dopustit, ať už při návrhu, nebo přímo při psaní kódu, je možné dohledat opravdu mnoho. Nás zajímá zejména nevhodná volba použitých datových struktur.

Jednoduchým příkladem je vhodný výběr kolekcí pro práci s daty. Jejich výběr je nutné podřídit zejména očekávaným operacím nad prvky kolekce. Je jasné, že pokud očekáváme častý náhodný přístup k jednotlivým prvkům, nebude vzhledem k algoritmické složitosti optimální spojový seznam, kde by složitost dosahovala minimálně O(n), přičemž volbou indexované kolekce se můžeme dostat na složitost blížící se O(1) (viz např. (Java Collections, 2011)).

Jak bylo zmíněno, ohled je kladen zejména na práci s velkým množstvím dat. Ale co v daném kontextu znamená práce s daty? Ta představuje několikanásobnou iteraci nad kolekcemi dat za účelem jejich seskupování, či výpočtu různých agregačních funkcí. Při těchto operacích nás nezajímá rychlost náhodného přístupu na vybraný index, naopak je nutné velmi rychlé sekvenční procházení a také vkládání nových elementů. Důležitou charakteristikou je možnost práce se stejnými prvky – ve vstupních datech se může vyskytovat mnoho identických hodnot.

Uvedené věty celkem jasně stanovují podmínky na výběr vhodného typu kolekce. V jazyce Java, který bude základním kamenem modulu, existuje pro zmiňovanou charakteristiku jasný kandidát – LinkedList. LinkedListy jsou široce využívány zejména ve fázi přípravy dat pro vizualizační modul, či pro uchování dat v rámci modulu.

7.5 Struktury pro zpracování dat

V přechozí sekci odůvodňuji využití vhodné datové struktury pro práci s daty. Vzhledem k popsaným požadavkům byl zvolen LinkedList, který umožňuje velmi rychlé sekvenční procházení i přidávání nových prvků. Pokud se ovšem s dosaženou rychlostí zpracování nesmíříme a nadále budeme hledat jiné postupy vedoucí ke zrychlení

aplikace, narazíme na zmínky o balíčku s názvem Stream, někdy označovaný jako Java Stream API. Ale co vlastně Stream je a k čemu slouží?

Jedná se o značně inovativní balíček, který zavádí mnoho nových funkcionalit a vylepšení, jimiž v mnoha směrech posouvá Javu o velký krok kupředu. Z těch nejdůležitějších jmenuji zejména: funkcionální styl programování, pipelining, automatická vnitřní iterace, nový přístup ke zpracování množiny dat a široká podpora agregačních funkcí. (Urma, 2014; Winterberg, 2014a; Java Streams; Tutorialspoint:Java 8 – Streams, 2016)

Zavedením Stream API dochází ke změně případů užití kolekcí při psaní běžného aplikačního kódu. Kolekce díky svému zaměření na způsob uložení obsažených prvků a vzhledem k efektivnosti k jejich přístupu jsou i nadále vhodným způsobem pro uchovávání datových prvků, objektů. Naopak streamy se zaměřují na provádění agregačních operací nad prvky svého datového zdroje. Přicházejí dokonce s řadou předem připravených funkcionalit k ještě většímu usnadnění rutinní práce programátora, kterou musel před jejich zavedením řešit. Díky tomu nyní můžeme na Stream API delegovat řadu případů užití, k nimž byly kolekce dříve využívány.

Příkladem je třeba výpočet sumy nad proměnnou X obsaženou v objektech uložených v kolekci. Dříve musel programátor explicitně iterovat celou kolekci a pro každý prvek zjistit hodnotu proměnné X, přičíst ji k nějaké globální proměnné a po projití všech prvků vrátit hodnotu globální proměnné. S využitím streamu, stačí pouze ukázat na datový zdroj, určit proměnnou a zavolat předem připravenou funkci – to vše na jednom řádku kódu. Následuje ukázka takového výpočtu. V úvodu je inicializovaná kolekce intList, která níže slouží jako zdroj dat pro stream.

```
1  // initialize list of Integers
2  List<Integer> intList = Arrays.asList(1, 12, 18, 6);
3
4  // Convert list to stream and summarize its values
5  int sum = intList.stream().mapToInt(Integer::new).sum();
```

Už z popisu je jasná první výhoda, výrazná úspora kódu i času programátora. Další podstatnou výhodou použití Stream API je automatická iterace. Ta díky implementaci přímo tvůrci programovacího jazyka zaručí velmi dobrou algoritmickou složitost a tím urychlí aplikační kód. Jinou výhodou je tzv. pipelining. Ten umožňuje zřetězení operací prováděných nad streamem. Viz následující příklad.

```
1  // initialize list of Strings
2  List<String> values = Arrays.asList("a1", "b2", "a3", "a2", "b1")
       ;
3
4  // Convert list to stream, change all characters to upper case,
5  // sort them alphabetically and print output
6  values.stream().map(String::toUpperCase).sorted().forEach(System.
       out::println);
7
8  // out = "A1", "A2", "A3", "B1", "B2"
```

V ukázkovém kódu se nejprve inicializuje kolekce řetězců `values`. Poté následuje vytvoření streamu a pipelining zvolených operací. Slovně popsáno to znamená: každý prvek streamu se převede na kapitálky, následně seřadí podle abecedy a nakonec se na standardní výstup vypíše výsledek. V závěru je uvedena ukázka výstupu.

Zde je vidět velmi jednoduchá ukázka síly Stream API. Dříve by totiž programátor musel psát složitější kód pro opakovanou iteraci kolekce `values` pro každou z operací. Nyní vše napíše zkráceným zápisem na jedné řádce.

Lambda výrazy

Stream API je sám o sobě velmi silným nástrojem pro výpočty nad daty, ale ve spojení s tzv. Lambda výrazy (mimochodem také zavedené v Javě 8) dosahuje zcela nových možností. Informace o Lambda výrazech lze dohledat např. v těchto zdrojích (Lambda expressions, 2015; Winterberg, 2014b; Lambda Quick Start). Lambda výrazy, zjednodušeně řečeno, umožňují předávání funkcionality jako parametr. Toto bylo již dříve možné a to použitím anonymních vnitřních tříd, tak proč se zavádí něco co „jen supluje“ již existující postupy? Důvodů lze v různých zdrojích najít několik. Těmi nejčastěji uváděnými je výrazné zjednodušení a zpřehlednění kódu.

Mnoho anonymních vnitřních tříd je dnes v Javě konstruováno pouze s jednou abstraktní metodou, častým příkladem můžou být EventListenery či Comparator. Z každého takového rozhraní se v Java 8 stává tzv. funkční rozhraní. A ke zjednodušení a zpřehlednění implementace těchto rozhraní poslouží právě Lambda výrazy. V následujícím úseku kódu je zobrazeno porovnání původního zápisu ActionListeneru pomocí anonymní vnitřní třídy v Javě 7, se zkráceným zápisem pomocí Lambda výrazu v Javě 8.

```
1  // traditional example of ActionLister in Java 7
2  clickMeBtn.addActionListener(new ActionListener(){
3
4      @Override
5      public void actionPerformed(ActionEvent ae){
6          System.out.println("Clicked !!");
7      }
8  });
9
10 // example of ActionListener in Java 8 using Lambda expression
11 clickMeBtn.addActionListener(ae -> System.out.println("Clicked !!
       "));
```

Java 8 přichází s celou řadou předem připravených funkčních rozhraní. Těmi nejdůležitějšími jsou: Comparator, Predicate, Consumer, Function, Supplier, UnaryOperator, BinaryOperator.

Následující úsek kódu zobrazuje příklad propojení Lambda výrazu a Stream API. Nejprve je opět inicializován list řetězců `values`. Z toho je vytvořen stream. Na něj je aplikován filtr, který jako parametr přebírá implementaci funkčního rozhraní Predicate. Predicate navrací hodnotu pravda/nepravda na základě vnitřního

vyhodnocení vstupního parametru. V tomto příkladě je implementace rozhraní Predicate provedena jednoduchým lambda výrazem `s -> s.startsWith("a")`, který vrátí pravda, pokud řetězec začíná písmenem `"a"`. Konečný výpis na standardní výstup je proveden pouze pro prvky splňující podmínky filtru.

```
1  // initialize list of Strings
2  List<String> values = Arrays.asList("a1", "b2", "a3", "a2", "b1")
       ;
3
4  // Convert list to stream, filter elements, only those starting
5  // with "a" will go through, finally print output
6  values.stream().filter(s -> s.startsWith("a")).forEach(System.out
       ::println);
7
8  // out = "a1", "a3", "a2"
```

Další, pokročilé, ukázky lze nalézt v kapitole Implementace.

Stream API v kombinaci s Lambda výrazy, hraje klíčovou roli při zpracování dat ve vizualizačním modulu.

7.6 Jazykové mutace

Speech Analytics je platforma se světovou působností. Tento fakt se odráží i do jisté potřeby co nejvíce se přiblížit uživatelům. Toho se snaží, mimo jiné, dosáhnout i jazykovou mutací ovládacích prvků celé platformy. V současné době lze v nastavení platformy zvolit kterýkoli z následujících jazyků: čeština, angličtina, němčina a ruština.

Potřeba jazykových mutací se tak stává dalším bodem, který bude nutné v rámci této práce řešit. K dosažení tohoto cíle využiji standardní funkcionality tzv. ResourceModelů zabudovaných ve Wicket frameworku, viz. sekce 5.1. Ty fungují na bázi properties souborů, které jsou uloženy ve stejném balíčku jako Java třída reprezentující danou webovou stránku. Pro každý jazyk je určený jeden zvláštní properties soubor.

8 Implementace řešení

8.1 Použité technologie

Níže je vyjmenován seznam použitých programovacích jazyků, frameworků a balíčků. Jejich podrobný popis je možné nalézt v sekci 5.1 a zdrojích v této sekci uvedených.

- Java SE/EE 8
- Apache Wicket (v6.21)
- MyBatis (v3.3.0)
- JaxB + Eclipse Link MOXy
- Spring (v4.1.1)
- Maven
- MySQL
- Plot.ly (v1.9.0)
- JavaScript + jQuery

8.2 ORM mapování do struktury výsledků technologií

V sekci 7.3 na straně 39 je popisována univerzální struktura Java objektů určená pro jednotnou práci s výsledky řečových technologií. Její podoba je patrná na obrázku 11. S návrhem takovéto struktury také musíme řešit otázku naplnění daty. Struktura je sice navržena tak, aby pojmula výsledky všech dostupných řečových technologií najednou, nicméně při běžném používání bude k tomuto stavu docházet jen zcela výjimečně. Vzhledem k povaze navrhovaného modulu bude nutné požadované hodnoty z datového zdroje načítat dynamicky na základě uživatelských požadavků.

Dynamické vytváření objektů nepůsobí žádný problém v programovacím jazyce, ovšem obtíže přicházejí v části získávání dat z datového zdroje. V našem případě je datovým zdrojem relační MySQL databáze a ta ve své podstatě nijak nepodporuje vytváření dynamických SQL dotazů na volitelné sestavení požadovaných výstupů. K tomu účelu bude potřeba použít nadřazenou vrstvu, ve které dynamické vytvoření SQL dotazu přesně podle našich požadavků proběhne. Vytvořený dotaz pak mezivrstva předá databázi a ta jej vykoná jako běžný dotaz. Volba, který framework řešící danou problematiku v práci použít, byla jasná – MyBatis. Platforma Speech Analytics jej již používá, nebyl tedy důvod přidávat do projektu další podobný framework. MyBatis nabízí dvojí přístup k vytváření těchto dotazů – anotace Java kódu, či popis dotazu v XML souboru.

Pod odstavcem je ukázkový kód vytváření dynamického SQL dotazu za využití MyBatis. K definici dotazu byl v ukázkovém příkladu použit přístup založený na

XML souboru. Jak je na první pohled patrné, konstrukce dotazu je založena na kombinaci běžného SQL dotazu s frameworkem definovanými ovládacími konstrukcemi. Ty vychází z XML tagů a slouží právě pro dynamickou úpravu dotazu.

```
1  <select id="getResults" resultMap="generalResultMap">
2    SELECT
3
4    <!-- choose which results select -->
5    <foreach item="elem" collection="axis">
6        <choose>
7            <when test="elem eq 'AGE'">
8                <include refid="aidResult" />,
9            </when>
10           <when test="elem eq 'GID'">
11               <include refid="gidResult" />,
12           </when>
13           <when test="elem eq 'LID'">
14               <include refid="lidResult" />,
15           </when>
16           <when test="elem eq 'TIME_STAMP'">
17               r.createdTime AS r_createdTime,
18           </when>
19   ...
```

Popis vybraných řádků kódu:

1. `select` je ovládací tag frameworku, určující, že se jedná o `SELECT` příkaz jazyka SQL. Atribut `id` je identifikátorem použitým k odkazování v rámci frameworku. Slouží také jako klíč k propojení do Java kódu. Mapování výsledků dotazu na Java objekty je specifikováno v tzv. ResultMap, jež je určena stejnojmenným atributem. Popis funkce mapování bude uvedeno dále v textu.

2. Běžné SQL

5. Frameworkem určená konstrukce pro cyklus nad všemi prvky kolekce `axis`. Ta je předána jako vstupní parametr dotazu. Jednotlivé elementy této kolekce jsou nadále v kódu identifikovány zástupným řetězcem určeným atributem `item`

6. a 7. Tagy `choose` a `when` jsou dalšími ovládacími konstrukcemi myBatis. Slouží k podmíněnému výběru jedné z možností určených obsahem tagů `when`. Funkce tagů `choose` a `when` je identická k příkazu pro řízení toku Switch známým z jazyka Java.
   Atribut test v tagu `when` slouží ke specifikaci podmínky určující, zda bude tělo tohoto tagu zpracováno.

8. `include` je dalším řídícím tagem. Jak jeho název napovídá, slouží pro vložení určité části kódu specifikované v jiné části dokumentu, či dokonce úplně v jiném dokumentu. Parametr `refid` určuje identifikátor vkládané části kódu.

17. Tento řádek zmiňuji jako ukázku, že i řídicí tagy `when` mohou obsahovat standardní SQL kód

Uvedený kód reprezentuje jen úvodní část sestavovaného SQL dotazu. Ta představuje dynamickou konstrukci výsledné sady hodnot vrácených SQL dotazem. Konkrétně tento uvedený příklad má za úkol vybrat pouze výsledky řečových technologií, jejichž název je obsažen v listu axis. List axis je jedním ze vstupních parametrů pro MyBatis při konstrukci dotazu. Velmi podobným způsobem, za využití podmínkových příkazů dochází ke konstrukci i ostatních částí SQL dotazu. Tedy propojení tabulek i finální podmínkovou část dotazu. Pro ilustraci byl uveden jen úvodní příklad, další části jsou analogické.

Předcházející řádky se zabývají způsobem dynamické konstrukce SQL dotazu v MyBatis. Díky takovým konstrukcím jsme schopni optimalizovat dotaz i jeho náročnost za účelem získat z datového zdroje jen požadovaná data. To ovšem není vše, co je pro nás MyBatis schopný udělat. Další důležitou rolí, kterou zastává, je objektové mapování výsledků SQL dotazů. Mapování opět probíhá na základě vytvoření XML dokumentu, jenž definuje vzor, podle kterého bude celý proces probíhat. Ukázka způsobu mapování je naznačena v následujícím úryvku kódu.

```
1   <resultMap id="gidResultMap" type="GidResult">
2       <id column="r_id" />
3       <collection property="channelResultList" column="ch_id"
            resultMap="channelGidResultMap" ofType="ChannelGidResult"
            />
4   </resultMap>
5
6   <resultMap id="channelGidResultMap" type="ChannelGidResult">
7       <id column="ch_id" />
8       <association property="channel" column="ch_id" resultMap="
            channelTmpResultMap" />
9       <association property="gender" column="gender_id" resultMap="
            genderResultMap" />
10  </resultMap>
```

Uvedená část kódu popisuje mapování výsledků technologie pro určení pohlaví mluvčího uložených v databázi do Java objektu `GidResult`. Jednoznačnost mapování objektů je zaručena použitím tagu `<id>`. Proměnná jakéhokoli neprimitivního objektu je mapována tagem `<association>`, který atributem property určí název proměnné v mapovaném objektu (zde `GidResult`), atributem `column` určí odpovídající sloupec z výsledku SQL dotazu a atributem `resultMap` odkaz na další podrobné mapování tohoto objektu. Pro přemapování kolekcí objektů se využívá tag `<collection>`. Ten má stejné atributy jako tag `<association>` a navíc k nim přidává atribut `ofType`, jenž říká, jakého typu jsou objekty dané kolekce.

8.3 Příprava datové kolekce pro vizualizační modul

Otázka nutnosti procesu extrakce dat z univerzální struktury výsledků a vytvoření datové struktury pro vizualizační modul byla řešena v sekci 7.3 na straně 42. Ve zmíněné části textu byl také navržen a popsán objekt DataObject, jenž je základním objektem pro vložení dat do vizualizačního modulu. Zde na zmíněnou sekci naváži popisem způsobu implementace tohoto procesu.

Jádro procesu extrakce požadovaných dat je postavené na využití Java Stream API a Lambda výrazech. Přínosy těchto balíčků společně s odůvodněním jejich využití je zmíněno v sekci 7.5 na straně 44. Ukázka implementace extrakce výsledků technologie odhadu věku mluvčího je předmětem následujícího úryvku kódu.

```
1  private LinkedList<Integer> getAgeResultList(List<Recording>
       rawDataList, ChannelCode channelCode) throws
       DataExtractionException {
2      LinkedList<Integer> res = null;
3
4      try {
5         res = rawDataList.stream().map(
6                   x -> x.getBasicAnalysisResult().getAidResult()
7                        .getChannelResultByCode(channelCode).getAge()
8               ).collect(Collectors.toCollection(LinkedList::new));
9
10     } catch (Exception e) {
11        throw new DataExtractionException();
12     }
13
14     return res;
15 }
```

Metoda `getAgeResultList` přebírá dva vstupní parametry: `rawDataList` a `channelCode`. `rawDataList` je kolekce nahrávek a výsledků řečových technologií k nim přiřazených a získaných z datového zdroje na základě požadavku uživatele. Parametr `channelCode` identifikuje z jakého kanálu nahrávky se výsledky budou extrahovat (nahrávka zpravidla obsahuje dva kanály, operátor a klient).

Tělo metody obsahuje zmíněnou kombinaci Stream API a Lambda výrazu. Nejdříve se `rawDataList` převede na Stream objekt (stream A). Na ten se zavolá tzv. intermediate operace `map`, která říká, vytvoř objekt Stream (stream B) z výsledků zpracování původního objektu Stream (stream A). Následuje hlavní fáze zpracování (řádky 6 a 7). Zde je využit Lambda výraz k získání věku mluvčího pro vybraný kanál každé z nahrávek. Přesněji řečeno se zde používá zkrácená implementace funkcionálního rozhraní Function zapsaná pomocí Lambda výrazu.

Poslední částí celého procesu je zavolání tzv. terminální operace `collect`. Ta slouží k provedení redukce výstupního streamu (stream B) na parametrem určený objekt, v našem případě `LinkedList`.

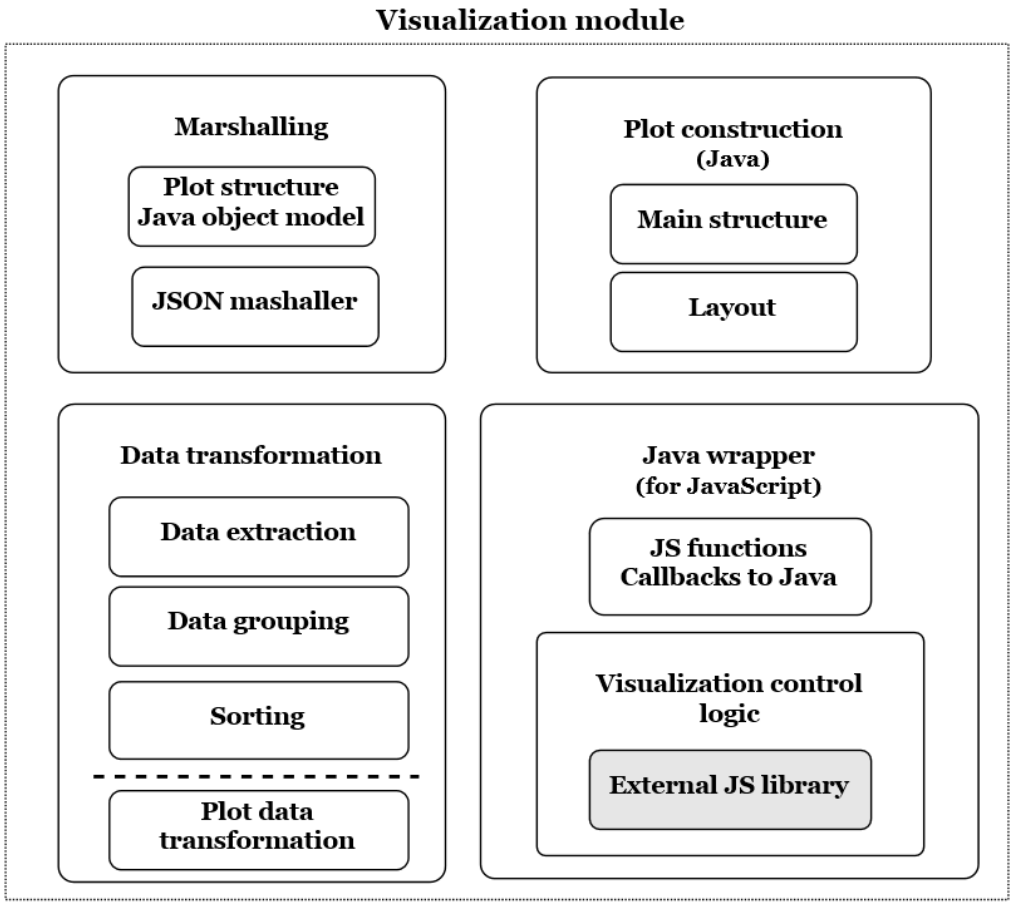
Výsledkem celého zpracování je tak kolekce číselných hodnot. Ty reprezentují věk mluvčího z vybraného kanálu každé z nahrávek. To vše přehledně zapsané pouze

na jediném řádku a to díky využití Stream API a Lambda výrazů.

Popsaný postup získání výsledků samozřejmě nezahrnuje celý proces extrakce a přípravy dat pro vizualizační modul. Mezi další problémy, které je nutné řešit, patří implementace logiky výběru druhu výsledků, jež je potřeba z dostupné datové struktury extrahovat, logika jejich přiřazení vybrané ose grafu v objektu DataObject, konverze mezi různými datovými typy i řízení nestandardních situací výstavbou vlastní struktury výjimek.

8.4 Vizualizační modul

Jak již bylo zmíněno v sekci 3 na straně 43, vizualizační modul je klíčovou částí celé práce. Jeho úkolem je na základě požadavků uživatele získat potřebná data, automatizovaně nad nimi provést veškeré potřebné transformace či agregace a nakonec zkonstruovat a vykreslit uživateli výsledný graf. Z tohoto krátkého popisu je jasné, že modul je poměrně komplexním dílem a to muselo být zohledněno i do jeho architektury. Její návrh je na obrázku 14.



Obrázek 14: Struktura vizualizačního modulu

Modul je rozdělen do čtyř hlavních částí: Data transformation, Java wrapper, Marshalling a Plot construction. Popis jednotlivých částí a ukázky z jejich implementace jsou předmětem dalších odstavců této sekce.

Transformace dat (Data transformation)

Blok transformace dat se skládá z dalších čtyř částí: extrakce, seskupovaní, řazení a finální transformace podoby dat. Extrakcí se v této části rozumí získání všech požadovaných hodnot pro vybranou osu z kolekce vstupních objektů DataObject. Samotná extrakce a zobrazení jejích výstupních dat by však pro požadovanou funkcionalitu modulu nebyla dostačující. Představme si příklad, kdy si uživatel přeje zobrazit, jaké je rozložení věku klientů volajících do kontaktního centra. Uživatel intuitivně očekává výsledky v nějaké sumarizované podobě, například že pro každou hodnotu věku bude vrácen počet hovorů jemu odpovídajících. Samotná extrakce by ale vrátila pouze řadu číselných hodnot představující vždy věk klienta pro daný hovor. Proto je potřeba zapojit seskupování dat.

V základním nastavení jsem v této práci zvolil pro přípravu dat seskupování na základě unikátních hodnot. Vrátím-li se k příkladu se zobrazením rozložení věku klientů volajících do kontaktního centra, zjistím, že seskupením na základě unikátních hodnot (v tomto případě věk mluvčího) dosáhnu přesně požadovaného cíle. Výstupem procesu extrakce a seskupení by tak měl být seznam párů hodnot (mapa hodnot neboli asociativní pole), kde první hodnota (klíč) je věk a druhou hodnotou je kolekce hovorů, které provedli klienti se stejným věkem (z kolekce pak jednoduše zjistíme počet).

Ukázka extrakce hodnot pro požadovanou osu se základním seskupením je zobrazena v následující části kódu.

```
1  private TreeMap<?, LinkedList<DataObject>> defaultGroup(
       LinkedList<DataObject> datalist, PlotAxisEnum plotAxisEnum,
       int axisIndex) throws DataExtractionException {
2
3      try {
4         return datalist.stream().collect(
5            Collectors.groupingBy(
6              x -> x.getAxisData(plotAxisEnum, axisIndex),
7              TreeMap::new,
8              Collectors.toCollection(LinkedList::new))
9            );
10
11     } catch (Exception e) {
12        throw new DataExtractionException(e.getMessage());
13     }
14 }
```

Jak je z ukázky patrné, k extrakci a seskupení dat je opět využito Java Stream API v kombinaci s Lambda výrazy, jež je popsáno v části 7.5. Rozšířením oproti předchozím příkladům je využití funkce `groupingBy` z třídy Collector, která imple-

mentuje mnoho klíčových operací pro redukci výstupu zpracování streamu. Konkrétně `groupingBy` slouží k vytvoření mapy (asociativní pole), kde klíč je výsledkem klasifikační funkce předané prvním parametrem funkce a hodnota přidružená ke klíči je definována tzv. collectorem předaným ve třetím parametru. Collector převezme všechny vstupní prvky, v našem případě všechny objekty se stejným klíčem, a vloží je do zvoleného kontejneru. V příkladu je jako kontejner pro výsledek zvolen `LinkedList`.

Seskupování dat na základě unikátní hodnoty je klíčovým faktorem práce modulu. Nicméně v některých případech se tento přístup uživateli příliš nehodí a raději by si zvolil vlastní způsob seskupení. K ilustraci toho případu lehce modifikujeme výše probíraný příklad zobrazení věku volajících do kontaktního centra. Uživatel i nadále bude chtít zobrazit věkové rozložení, ale nepotřebuje zacházet do přílišného detailu a zobrazovat počty hovorů pro všechny možné hodnoty věku mluvčího. Naopak mu stačí zobrazit počty pro určité věkové skupiny, např. po deseti letech.

A přesně toto je další implementovaná funkcionalita. Na základě uživatelem definovaného vstupu lze seskupit data do různě velkých skupin. Uvedené platí primárně pro číselné datové typy, ale i pro některé předem definované varianty seskupení data či času. Následuje ukázkový kód seskupení hodnot pro celočíselné datové typy.

```
1  private TreeMap<?, LinkedList<DataObject>> groupInt(LinkedList<
       DataObject> datalist, PlotAxisEnum plotAxisEnum, int axisIndex
       , int groupingStep) throws DataExtractionException {
2
3      try {
4         return datalist.stream().collect(
5            Collectors.groupingBy(
6              (x -> intValGroupingFce.apply(
7                (Integer) x.getAxisData(plotAxisEnum, axisIndex),
                     groupingStep)
8               )
9              TreeMap::new,
10             Collectors.toCollection(LinkedList::new))
11           );
12
13     } catch (Exception e) {
14        throw new DataExtractionException(e.getMessage());
15     }
16 }
17
18 private BiFunction<Integer, Integer, Integer> intValGroupingFce =
19   (BiFunction<Integer, Integer, Integer> & Serializable)
20   (x, y) -> {
21     return x - (x % y);
22   };
```

Celkově zpracování probíhá stejným způsobem jako v předchozích příkladech. Jediným a klíčovým rozdílem je změna funkce na určení hodnoty klíče pro seskupení hodnot. Tu najdeme na řádcích 7 a 8. Jedná se o Lambda výraz, který pro výpo-

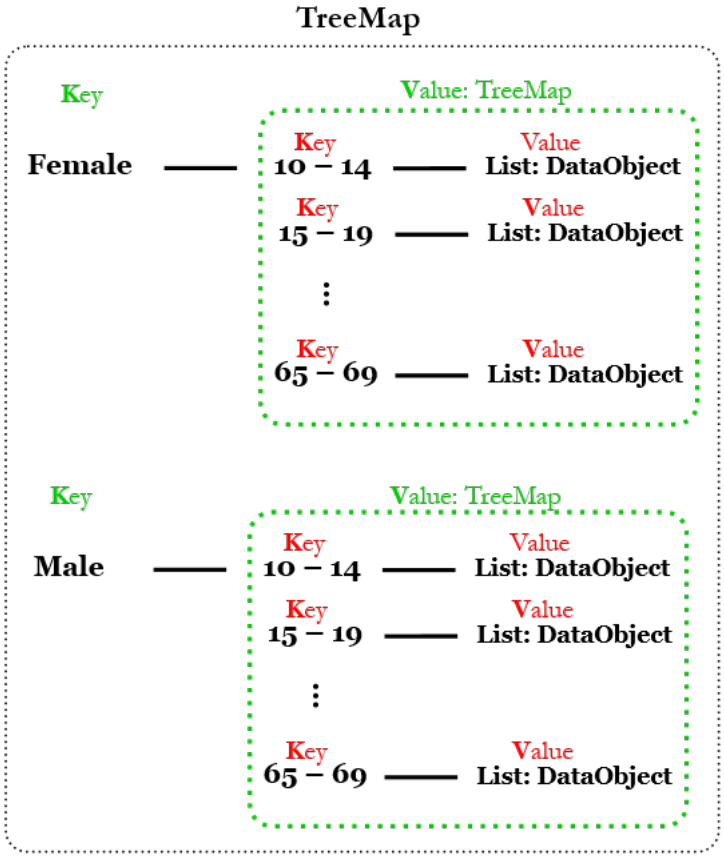
čet hodnoty (klíče ve výsledném asociativním poli) volá funkční rozhraní odkazované prostřednictvím proměnné `intValGroupingFce`. Definice `intValGroupingFce` je uvedena ve spodní části ukázky. Jedná se o implementaci abstraktní funkce z funkčního rozhraní `BiFunction`. Pro jiné datové typy je implementace analogická.

Více druhů výsledků na ose

Další důležitou funkcionalitou pro umožnění pokročilejšího zkoumání dat je možnost kombinace více druhů dat na jedné ose. Co si pod kombinací dat představit, popíši opět na ilustrativním příkladu.

Představme si, že jsme si zobrazili počty hovorů jednotlivých věkových skupin klientů volajících do kontaktního centra. Toho dosáhneme použitím dříve popsané funkcionality. Nyní si však řekneme, že by se nám hodilo upřesnění výsledků třeba informací o pohlaví jednotlivých klientů. Výsledkem by tedy měl být graf, jenž zachová dříve určené rozložení věkových skupin, ale namísto jednoho sloupce pro každou skupinu zobrazí vždy sloupce dva – počet žen a počet mužů z dané věkové skupiny.

V grafickém rozhraní si tuto možnost uživatel jednoduše zvolí, složitější část ovšem nastává ve fázi implementace. Logickou úvahou je jasná nutnost znovupoužití asociativních polí. Základní princip zůstane stejný, klíč reprezentuje skupinu shodných dat, ale hodnota odpovídající klíči již nebude pouze kolekce, ale další vnořené asociativní pole, které slouží pro vnitřní rozdělení hodnoty schované pod klíčem. Pro přehlednost je popsaná struktura map zobrazena a popsána na obrázku 15.



Obrázek 15: Princip vnoření asociativních polí pro stupňovité dělení hodnot

Při zvýšené pozornosti si lze na obrázku všimnout rozdílu v pořadí zpracování datových balíčků (výsledků řečových technologií) v porovnání s popsaným příkladem. V něm je řečeno, že uživatel si nejdříve zobrazí dělení věkových skupin a ty upřesní pohlavím mluvčího. Logicky by tak na první úrovni mapy měly být věkové skupiny a až ve vnořené úrovni rozdělení podle pohlaví. Nicméně zobrazené prohození je účelné a vyplývá zejména z potřeb podoby dat přijímané knihovnou Plot.ly (JavaScriptová knihovna použitá pro vykreslování grafů, viz sekce 5.3).

Z pohledu implementace je popsaný princip analogický dříve popsané extrakci. Dochází pouze k rozšíření řídící logiky, kde je nutné zohlednit hlavně opačné pořadí zpracování vstupních dat a odpovídající konstrukci vnořených asociativních polí.

Agregace nad výsledky

Veškeré předchozí kroky provedené ve fázi transformace vstupních dat byly zaměřeny na přípravu dat zobrazených na ose X vybraného grafu. Proto nyní následuje krok určení hodnot osy Y. Tyto hodnoty jsou dány aplikací uživatelem zvolené agregační funkce na skupiny dat vytvořené v přechozích krocích. Modul umožňuje použít tyto agregační funkce: počet, průměr, součet, minimum a maximum.

Ukázka výpočtu kolekce hodnot pro osu Y s aplikací funkce pro výpočet průměru je v následujícím bloku kódu. Vstupem pro výpočet je mapa seskupených hodnot pro osu X.

```
1  // Data type of dataMap
2  // TreeMap<?, LinkedList<DataObject>> dataMap
3
4  dataMap.values().stream().mapToInt(
5    x -> ((Double)
6      (
7        x.stream().mapToInt(
8          y -> (int) y.getAxisData(plotAxisEnum, axisIndex)
9        )
10       .average().getAsDouble()
11     )
12 ).intValue()
13 ).boxed().collect(Collectors.toCollection(LinkedList::new))
```

Výpočet je opět proveden za využití Java Stream API v kombinaci s Lambda výrazy. Postup však, oproti předchozím ukázkám, už není tak přehledný a přímočarý. Nejdříve se vytvoří první stream (řádek 4), který prochází všechny hodnoty spojené s klíči vstupní mapy (řádek 5). Tyto hodnoty jsou však kolekcemi. Proto dochází k použití dalšího streamu za účelem projití všech hodnot vnořené kolekce (řádek 7). Z jednotlivých prvků je vybrána požadovaná hodnota (řádek 8), která je vstupem pro funkci `average` (řádek 9). Ta postupně shromáždí všechny hodnoty daného streamu a po jeho ukončení vrátí výsledek (průměr z kolekce hodnot pro daný klíč mapy `dataMap`). Tento proces se cyklicky opakuje, dokud není ukončen i první stream. Hodnota průměru pro každý z klíčů `dataMap` je vždy zabalena do datového typu `Integer` (řádek 12). Nakonec je výstup, dříve popsaným postupem, převeden do `LinkedListu` (řádek 13).

Řazení dat

Vzhledem k absenci automatického řazení vstupních hodnot na straně knihovny Plot.ly bylo nutné řešit otázku řazení ještě před odesláním dat do vizualizační knihovny. To jsem se rozhodl vyřešit využitím struktury TreeMap ve fázi přípravy dat. TreeMap je implementace asociativního pole, které automaticky řadí obsažené prvky na základě hodnoty klíče. Využívá k tomu principu Red-black stromu[23].

Konstrukce grafu a marshalling (Plot construction, Marshalling)

K vykreslování požadovaných grafů je v rámci této práce použita knihovna Plot.ly. Podrobný popis této knihovny je uveden v sekci 5.3. V následujících odstavcích vycházím hlavně z jejího deklarativního přístupu ke konstrukci grafu.

[23]Red-black strom – samo vyvažující se binární vyhledávací strom – viz https://www.cs.auckland.ac.nz/software/AlgAnim/red_black.html

Deklarativní přístup znamená, že celá struktura grafu včetně veškerých nastavení je tvořena komplexní jasně určenou objektovou strukturou, kde každý objekt odpovídá za nějakou část grafu. Ke změně vlastností jednotlivých částí pak stačí pouze změnit hodnotu proměnných odpovídajícího objektu. V případě Plot.ly je použitou objektovou strukturou JSON.

Běžně by mohla konstrukce grafové struktury pomocí JSON objektů probíhat v JavaScriptové části kódu, tedy na straně klienta. Nemuselo by tak docházet k posílání veškerých požadavků na podobu grafu, krom požadavků na data, na server. Nicméně jedním z požadavků při vývoji vizualizačního modulu bylo zachování co možná nejvíce aplikační logiky na straně serveru, v Javě. Použil jsem proto postup, kdy veškeré požadavky klienta na podobu grafu i data jsou po stisknutí tlačítka na vykreslení grafu odeslány na server, zde jsou připravena data, zkonstruována struktura grafu a vše je v jednom balíčku odesláno zpět do klientského prohlížeče k vykreslení.

Ke konstrukci grafu na straně serveru jsem přistoupil stejným způsobem, jenž prosazuje Plot.ly. Na základě referenční dokumentace dostupné z (Plot.ly project) jsem vytvořil zrcadlový obraz JSON struktury v Javě. Tento přístup sice klade značné časové nároky při prvotní implementaci, nicméně později umožňuje jednoduchou tvorbu grafu na základě práce s běžnými Java objekty tak, jak je programátor zvyklý.

Po vytvoření grafu ve formě Java objektů je nutné zajistit finální převod do JSON struktury. K tomu jsem využil kombinaci proprietárních Java JAXB anotací v kombinaci s EclipseLink MOXy marshallerem. JAXB je využit pro anotování všech objektů tvořících strukturu grafu i objektů k nastavení jeho stylu. Anotace pak slouží MOXy marshalleru jako přesný návod, který objekt a jak převést do JSON. Výsledkem procesu marshallování je tak 1 ku 1 kopie struktury grafu z podoby Java objektů do JSON struktury.

```
1  var data = [{
2    x: ['car', 'bus', 'train'],
3    y: [20, 14, 23],
4    type: 'bar'
5  }];
6
7  var layout = {
8        title : "Testing example",
9        autosize: true,
10       xaxis : {
11         title: "Transport type"
12       },
13       yaxis : {
14         title: "Count"
15       }
16 };
```

Java wrapper

Vykreslení výsledků práce vizualizačního modulu je v této práci prováděno pomocí JavaScriptové knihovny Plot.ly. Ta samozřejmě vyžaduje ke svému ovládání a interakci použití JavaScriptu. Jak už jsem ale několikrát v práci zmínil, požadavkem je udržet logiku ovládání v serverové části kódu. Navrhl jsem proto jednoduchý Java wrapper, který slouží k obalení JavaScriptového kódu a jeho zpřístupnění v Javě.

Wrapper funguje na principu vystavení unikátní URL adresy (tzv. callback adresa) pro každou JavaScriptovou funkci, kterou si přejeme implementovat v serverové části kódu. Tyto adresy jsou při inicializaci webové stránky předány JavaScriptovému dokumentu, jenž obsluhuje vykreslovací knihovnu. Ten se postará o jejich předání odpovídajícím funkcím. Zvolená funkce poté při svém spuštění zašle AJAX request na získanou adresu. Request je odchycen v serverovém kódu, zavolána určená obslužná metoda a výsledek je odeslán zpět.

Popsaný princip tak jednoduše umožňuje přenesení implementace zvolené aplikační logiky do serverového kódu. Rozsah JavaScriptového kódu je pak redukován pouze na vytvoření jednoduchých metod, které se skládají pouze z hlaviček a volání AJAX requestu.

8.5 Proces zpracování požadavku na vizualizaci dat

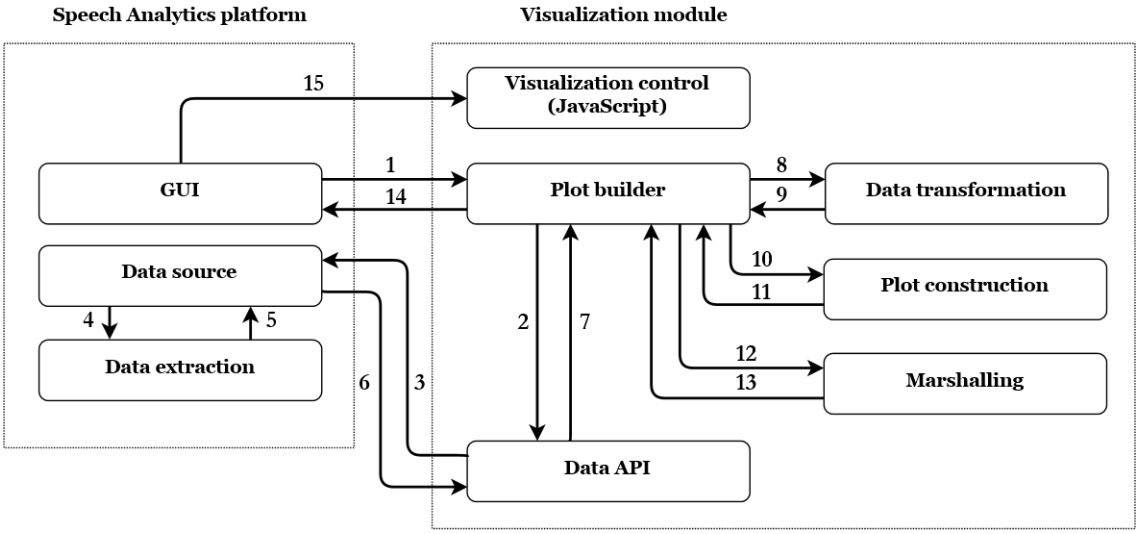
V předchozích částech textu jsem se zaměřoval na detailní implementace některých částí modulu. Snažil jsem se vybrat důležitá místa zpracování požadavku, u nichž jsem vždy v popisu vysvětlil problém a na ilustrativním příkladu popsal jeho řešení. Nicméně popisované části byly vždy vytrženy z kontextu a řešeny jako zcela samostatné celky.

V této sekci bych chtěl sumarizovat veškeré poznatky popsané v přecházejících částech dohromady. Tedy uvést jednotlivé části do logických souvislostí a vytvořit chronologickou linii návaznosti procesů v kontextu zpracování požadavku na vizualizaci dat vzneseného uživatelem aplikace.

Číslované schéma komunikace jednotlivých částí modulu a platformy Speech Analytics znázorňující ucelený proces zpracování požadavku na vizualizaci dat je zobrazeno na obrázku 16. Následuje číslovaný seznam s popisem jednotlivých šipek ze schématu:

1. Uživatel v grafickém rozhraní vybere své požadavky na vizualizaci výsledků řečových technologií. Těmi jsou např. typ grafu, řečová technologie, seskupení, časové rozmezí a další filtrovací kritéria pro výběr nahrávek. Tyto informace jsou odeslány do vizualizačního modulu, konkrétně do části Plot builder.

2. Plot builder je hlavní část vizualizačního modulu starající se o komplexní logiku zpracování požadavků. Konkrétně v tomto kroku pouze převezme uživatelský požadavek a přesměruje jej do části Data API.



Obrázek 16: Schéma zpracování požadavku na vizualizaci dat

3. Data API extrahuje z uživatelského požadavku veškeré informace týkající se specifikace hledaných dat, transformuje je do stanovené podoby a odešle požadavek na data do části Data source.
4. Data source převezme požadavek a na jeho základě dynamicky vytvoří dotaz do datového úložiště a obdrží požadované výsledky. V dalším kroku žádá o vytvoření struktury dat požadované vizualizačním modulem.
5. Data extraction blok na základě vstupních parametrů provede transformaci požadovaných dat z univerzální struktury výsledků do struktury požadované vizualizačním modulem. Výsledek pošle zpět do Data source.
6. Data source převezme výsledky a odešle je zpět do Data API vizualizačního modulu jako výsledek jeho požadavku na data.
7. Získaná data pouze přesměruje do bloku Plot builder.
8. Plot builder shromáždí potřebné informace z uživatelského požadavku a na jejich základě zažádá o transformaci dat.
9. Data transformation blok provede seskupení dat pro osu X a aplikací agregačních funkcí vypočte hodnoty pro osu Y. Nakonec výsledek odešle zpět.
10. Žádost o vytvoření objektové struktury uživatelem požadovaného grafu.
11. Plot construction blok na základě požadavku vytvoří dvě objektové struktury. První z nich reprezentuje datové položky k zobrazení v grafu. Jejich součástí je i základní nastavení struktury grafu. Druhá struktura slouží k definici vizuální podoby výsledného grafu, např. popisky os, použité barvy aj.

12. Plot builder převezme objektové struktury reprezentující vytvářený graf a zažádá o jejich transformaci.
13. Blok Marshalling provede transformaci vstupních struktur Java objektů do JSON.
14. Celkový výsledek zpracování – JSON reprezentace grafu i s daty je odeslána zpět jako odpověď na požadavek na vizualizaci dat.
15. Během vykreslování webové stránky s výsledky je odeslán JavaScriptový příkaz do klientského prohlížeče na vykreslení grafu.

Uvedené body rozhodně nejsou vyčerpávajícím popisem všech detailů jednotlivých částí procesu zpracování požadavků. Dochází k vypuštění některých méně kritických částí. Také abstrahuje od řešení nestandardních situací, které se prolíná všemi fázemi procesu zpracování požadavku na vizualizaci. Účelem tohoto popisu bylo vytvořit abstraktní náhled na ucelený proces zpracování uživatelského požadavku.

9 Výsledky

V předchozích částech textu je popisován celkový návrh vizualizačního modulu a postup implementace jeho jednotlivých částí. Zde se naopak zaměřím na výsledek práce z uživatelského pohledu, zejména na popis grafického uživatelského rozhraní, princip jeho ovládání a následně uvedu několik příkladů užití vizualizačního modulu.

Výsledné grafické rozhraní implementovaného modulu i se zasazením do platformy Speech Analytics je zobrazené na obrázku 23 v příloze A. V porovnání s jeho návrhem uvedeným v sekci 7.2 dostálo jen několika drobných změn. Došlo k přeuspořádání polohy ovládacích prvků. Nyní je tak veškeré ovládání týkající se výběru grafu a omezujících pravidel pro vstupní data v horní části webové stránky. Na levé spodní straně je umístěn posuvný blok určený k výběru a nastavení dat, jež budou ve výsledku zobrazena v grafu. Hlavní část obrazovky, vpravo dole, je určena k zobrazení výsledků. Toto rozložení zaručí zobrazení celého nastavení i výstupu na jedné obrazovce. Účelem je, aby uživatel nebyl nucen při změnách vstupních hodnot skrolovat mezi výsledným grafem a jeho nastavením.

9.1 Práce s modulem

Důležitou charakteristikou, jež vzhledem k návrhu zůstala zachována, je princip rozložení ovládání modulu do třech fází: výběr typu grafu a stanovení omezujících podmínek, výběr dat a jejich nastavení, zkoumání výsledků (zobrazeného grafu). Zmíněné body odráží i rozložení výše popsaného grafického rozhraní.

V první fázi, po výběru typu grafu, uživatel může volit několik omezujících podmínek na vstupní data. Těmi jsou: kampaň[24], časové rozmezí pořízení nahrávky (v základním nastavení je nastaveno rozmezí jeden měsíc dozadu od současného data) a typ kanálu nahrávky (operátor či klient). V druhé fázi uživatel zvolí typ dat zobrazených na ose X, například věk. Dalším nastavením typu dat (po kliknutí na ozubené kolečko pod typem dat) může určit, že hodnoty budou seskupovány do volitelně velkých skupin, například u věku skupiny po pěti letech. Jinou volbou nastavení v této části je dodatečné upřesnění vstupních dat. To probíhá na základě přidání dalšího druhu dat do výběru. Budeme-li uvažovat, že prvním typem dat je věk, můžeme přidat např. ještě pohlaví. Tím zajistíme, že každá věková skupina bude vždy upřesněna pomocí pohlaví (rozdělena na muže a ženy). Veškeré doposavad zmíněné pro fázi dva je nastavení pro osu X. Osa Y je určena aplikací zvolené agregační funkce na hodnoty dat z osy X. Agregační funkci uživatel zvolí výběrem z rozbalovacího menu se stejnojmenným nadpisem[25]. V základním nastavení je před vybrána funkce „Počet“. Po projití prvních dvou fází je nutné kliknout na tlačítko

[24] kampaň – seskupení nahrávek podle účelu hovoru

[25] pokud uživatel zvolí některou z funkcí Minimum, Maximum, Průměr, Součet, ale data vybraná v předchozích krocích nejsou číselného charakteru, např. pohlaví, je výběr ignorován a je aplikována funkce „Počet“

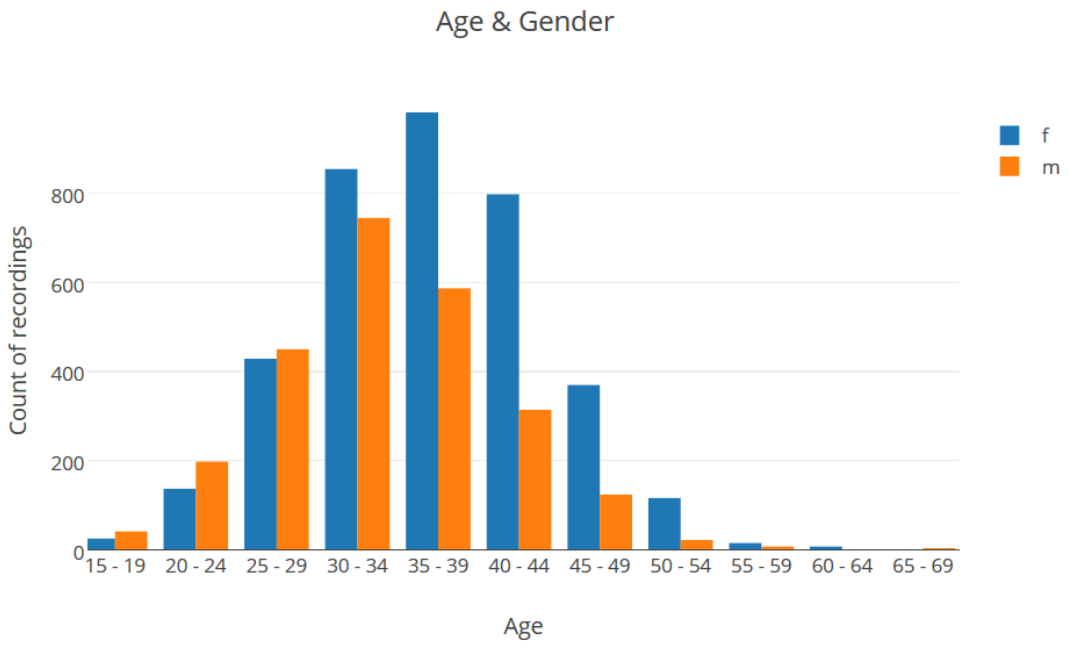
„Vykresli graf“ umístěné v levé spodní části obrazovky. To spustí proces získání dat z datového zdroje, jejich zpracování a vykreslí v pravé části obrazovky výsledný graf.

Ve třetí fázi může uživatel vizuálně zkoumat vizualizovaný graf. To mu usnadňují dostupné funkce: přibližování/oddalování částí grafu, posouvání os, zobrazení nejbližších či porovnání hodnot po najetí myší. V případě potřeby si může uživatel zobrazený graf exportovat a stáhnout ve formátu PNG.

9.2 Ukázkové příklady užití

Demografické rozložení

Prvním praktickým příkladem užití je kontrola demografického rozložení klientů volaných operátory kontaktního centra. Zajímá nás tedy rozložení klientů do věkových skupin po pěti letech upřesněné informacemi o pohlaví klienta. Praktické využití takové informace je například pro kontrolu plnění podmínek cílené reklamní kampaně ze strany zadavatele. V případě potřeby pouze operátor nastaví požadovaná kritéria a výstup přepošle zadavateli. Výsledný graf je na obrázku 17.



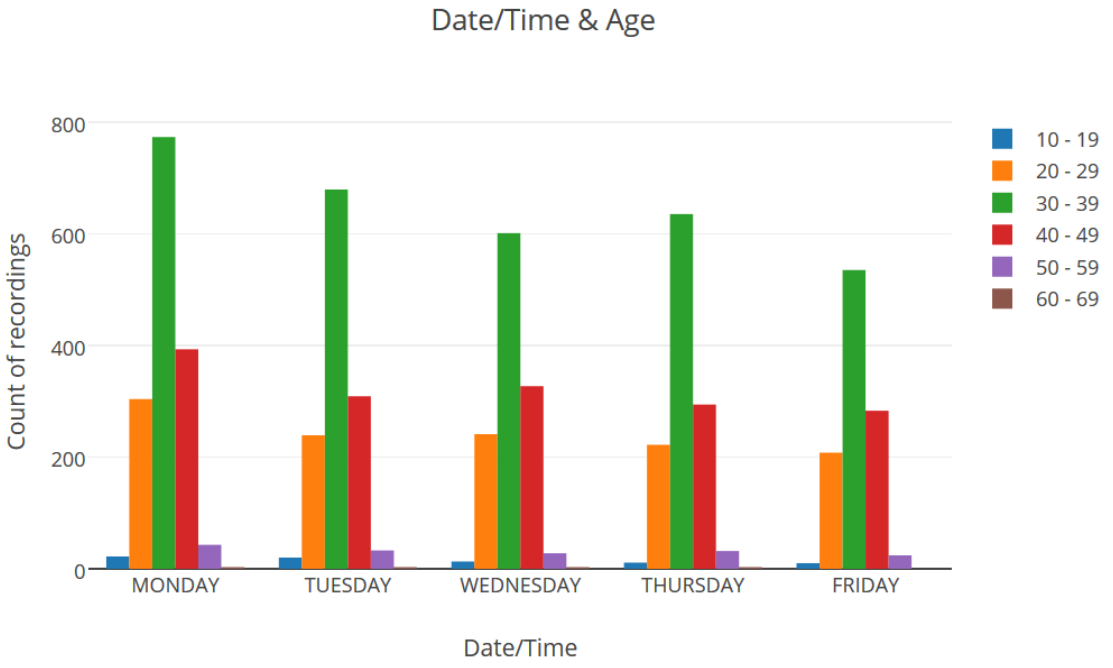
Obrázek 17: Demografické informace nahrávek

Tento příklad užití odpovídá na otázku „Kdo je mým zákazníkem?“, stanovenou v kapitole 6 Analýza požadavků, která patří mezi otázky, na něž by vizualizační modul měl být schopen dodat odpověď.

Jak je patrné, tento případ užití je cílený na odchozí hovory (analogicky je aplikovatelný i na hovory příchozí).

Kdy klienti volají

Jiným zajímavým příkladem užití může být zjištění rozložení příchozích hovorů. To nás může zajímat zejména z důvodu plánování směn operátorů a to nejen z pohledu počtu operátorů, ale i snahy optimálního párování vzhledem k věkovým skupinám. Data této povahy najdeme v obrázcích 18 a 19. Obrázek 18 zobrazuje rozložení počtu hovorů na jednotlivé dny týdnu s upřesněním na věkové skupiny. Obrázek 19 pak ukazuje počty hovorů jednotlivých věkových skupin nyní ale sumarizované vhledem k průběhu pracovního dne.



Obrázek 18: Rozložení věkových skupin po dnech

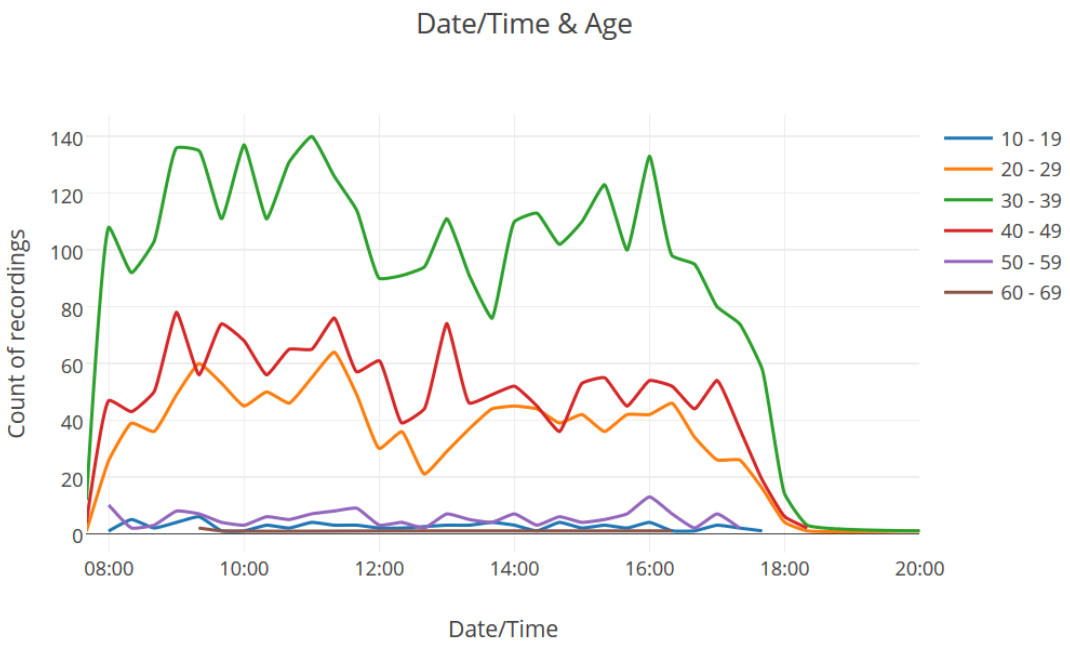
Grafy v tomto příkladu jsou pouze ilustrativního charakteru, jelikož příklad je cílený na příchozí hovory do kontaktního centra, ale data v nich zobrazená pocházejí z hovorů odchozích.

Příklad za předpokladu dostupnosti vhodných dat ilustruje odpověď na otázku „Kdy určité skupiny zákazníků volají?“ stanovenou v kapitole 6.

Kontrola operátora

Přístupů ke kontrole operátora na základě automatického vyhodnocení nahrávek může být několik. Jmenuji třeba rychlost řeči (měřená ve fonémech za vteřinu), porovnání délky řeči operátora vůči délce řeči klienta, délka ticha na konci hovoru[26],

[26]pokud klient položí telefon, ale operátor v systému hovor neukončí, tak se operátor jeví jako stále volající a není mu přiřazen další klient



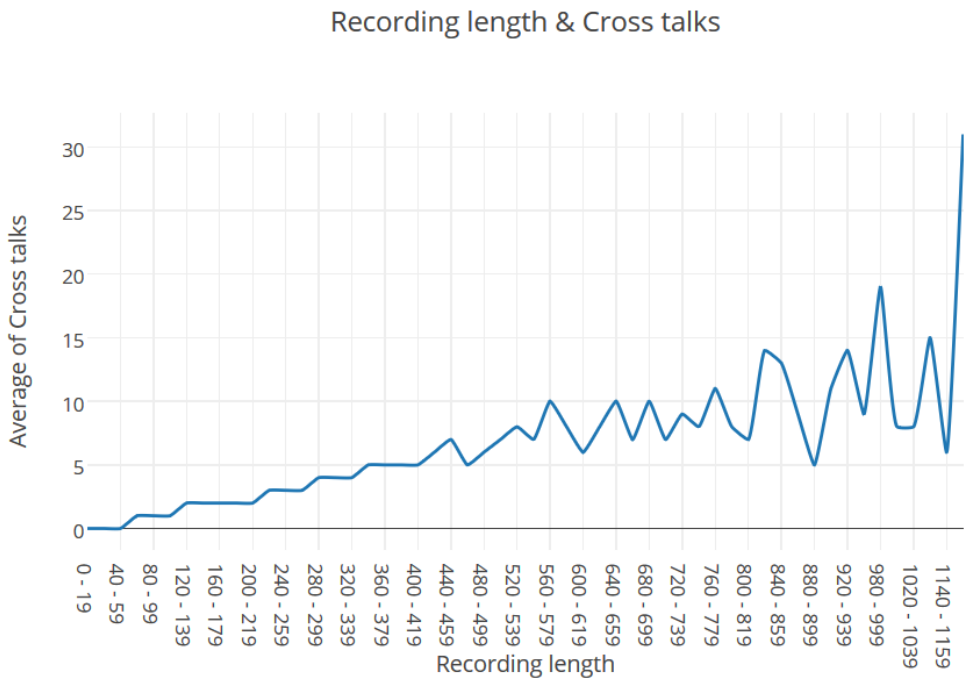
Obrázek 19: Rozložení věkových skupin během dne

počet skoků do řeči a jiné. Vzhledem ke kontrole operátora je nutné si uvědomit, že všechny zmíněné charakteristiky nelze brát jako zcela správné a rozhodující. Je potřeba k nim přistupovat pouze jako k indikátorům možných problémů, které je nutné jinými kanály ověřit.

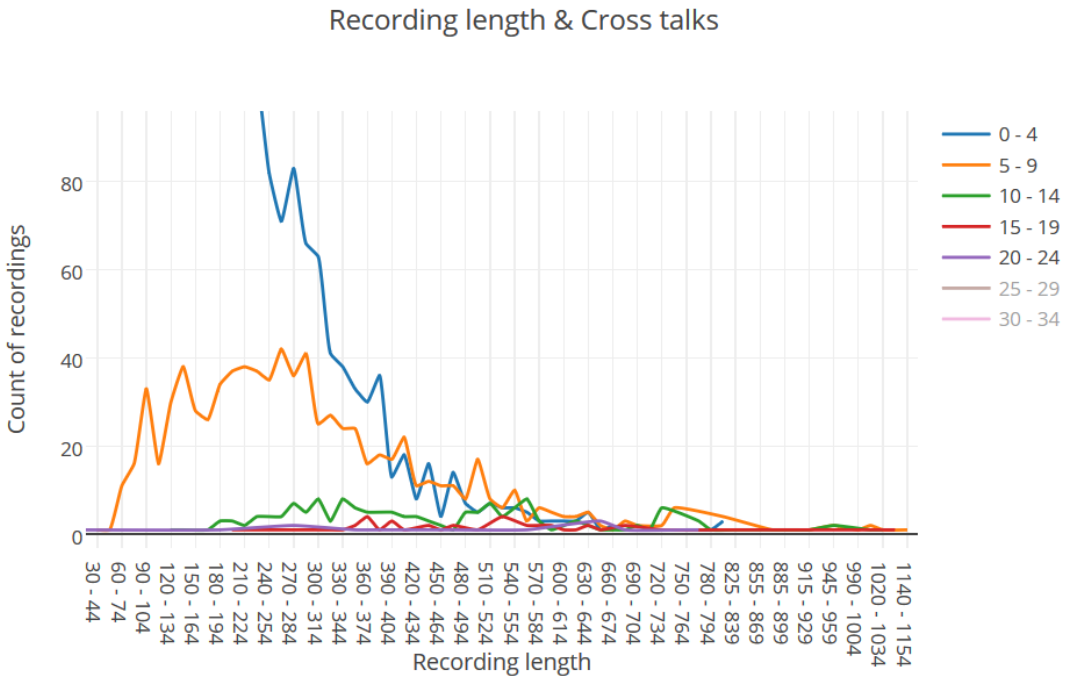
Následující grafy mohou sloužit jako nástin možné kontroly na základě počtu skoků do řeči. Na obrázku 20 je zobrazen průměrný počet skoků do řeči (ze strany operátora) vůči délce nahrávky (ve vteřinách). Je zde vidět postupný trend růstu počtu skoků do řeči s délkou nahrávky. Další pohled poskytuje graf na obrázku 21, který na ose Y zobrazuje počty nahrávek vzhledem ke skupinám určujících počty skoků do řeči (viz legenda) a to vše vůči délce nahrávky na ose X.

Interpretaci výsledků zmíněných grafů záměrně neuvádím, jelikož by se jednalo o zcela subjektivní názor založený na odhadu. Odpovídající vyhodnocení je na supervizorovi kontaktního centra, jenž se velmi dobře orientuje v dané problematice.

Tento příklad opět odpovídá na jednu z otázek stanovených v kapitole 6 Analýza požadavků na něž by měl odpovědět vizualizační modul.



Obrázek 20: Průměrný počet skoků do řeči



Obrázek 21: Skupiny skoků do řeči vůči délce nahrávky

10 Diskuze

10.1 Řešené problémy

Během vývoje rozsáhlejší aplikace se běžně objevují různé problémy bránící v implementaci řešení podle předem navrženého schématu. Takové problémy je nutné co nejdříve identifikovat a najít způsob jejich řešení. Zpravidla přitom platí pravidlo, že čím později se problém objeví, tím větší dopad má na celý projekt. Každé řešení znamená prodloužení času vývoje, odvíjí se od závažnosti problému, s čímž jde ruku v ruce zvýšení nákladů.

Nejinak tomu bylo i při vývoji vizualizačního modulu zpracovávaného v rámci této práce. Níže uvedu některé z problémů, jež bylo nutné v různých fázích implementace řešit.

Řazení dat

Během testování vývojové verze vizualizačního modulu, již plně implementovaného do platformy Speech Analytics, se objevil problém se špatným zobrazením dat – některé grafy byly nepřehledné a hodnoty pomíchané. Tento jev se objevoval pouze u kombinace některých druhů dat a byl výrazně pozorovatelný zejména u spojnicového grafu. Analýzou zmíněného se došlo ke zdroji problému, kterým byla neseřazená data předkládaná knihovně Plot.ly.

Plot.ly na vstupu očekává sadu dat, tzv. „trace“, která je již seřazena do požadovaného pořadí zobrazení dat podle osy X. Bylo tak nutné zastavit se a pozměnit značnou část procesu přípravy dat. Jelikož řazení až výsledných dat k zobrazení by kladlo značné nároky na vývoj mnoha druhů komparátorů, rozhodl jsem se řazení přesunout již do fáze seskupování dat. Pro vytváření asociativních polí jsem tak použil jednu ze speciálních implementací umožňující řazení prvků na základě klíčů asociativního pole, konkrétně TreeMap. Tímto krokem se vyřešilo řazení, nicméně potřeba mnoha druhů komparátorů přetrvala, protože klíče v asociativním poli neobsahovaly hodnoty, ale popisky os, např. řetězec „10–14“. Tento problém jsem vyřešil přesunutím vytváření popisků z části seskupování dat až do závěrečných kroků přípravy dat pro graf.

Objevení problému s řazením dat až v pozdější fázi vývoje zapříčinila souhra několika faktů. Zejména používání ideálního vzorku dat (seřazený a bez chybějících hodnot) pro testování v úvodních fázích vývoje. Výsledkem bylo poměrně značné zdržení způsobené potřebou reimplementace některých důležitých částí procesu transformace.

Problém s řazením byl bohužel objeven až v pokročilé fázi vývoje, a tak jeho řešení výrazně ovlivnilo již existující řešení.

Špatné zobrazení dat při práci s více skupinami dat na jedné ose

Dalším z řešených problémů bylo nestandardní zobrazování hodnot na ose X při vložení více druhů dat na tuto osu. Výsledný graf se v takovém případě choval zcela nestandardně. V některých případech docházelo k prohození pořadí zobrazení hodnot, a to i když vstup byl patřičně řazený. Často se při testech některé hodnoty vůbec nezobrazily v grafu i přesto, že ve vstupním souboru byly obsaženy. Sérií následných testů jsem zjistil, že Plot.ly si neporadí se správným zobrazením dat v případě, kdy první předaná sada dat není zcela kompletní, např. neobsahuje některé klíče s hodnotami, jež jsou v dalších sadách dat.

Popsaný problém jsem vyřešil přidáním mezikroku před předání dat k zobrazení vizualizační knihovně. Konkrétně bylo nutné projít všechny sady dat, vybrat všechny obsažené klíče k zobrazení na osu X a chybějící hodnoty přidat do v pořadí první sady dat. Ostatní sady tento krok nevyžadují, Plot.ly se orientuje podle první předané sady. Aby přidáním chybějících klíčů nedošlo ke zkreslení hodnot v grafu, bylo nutné každému chybějícímu klíči přiřadit tzv. prázdnou hodnotu. Tím se zaručí, že Plot.ly je o takovém klíči na ose X informován ale nebude s ním pracovat při zobrazení dané sady dat.

Další problémy

Při implementaci řešení vizualizačního modulu se postupně vyskytlo mnoho problémů, některé z nich drobné, jiné vážnějšího charakteru, jako např. problémy popsané výše. Mezi další, které stojí za to zmínit, patří problematická předání datumových hodnot do Plot.ly, nemožnost práce pouze s časovými údaji v Plot.ly, předávání informací o datových typech při převodu struktury grafu do JSON atd.

10.2 Zhodnocení a přínosy

Vizualizační modul vyvinutý v této práci přidal platformě Speech Analytics zcela nový pohled na analýzu nahrávek. Před jeho přidáním totiž platforma dokázala u nahrávek zobrazovat pouze číselné hodnoty jako výsledky analýzy některých řečových technologií. Problematické bylo i souhrnné zobrazení výsledků pro více nahrávek. V číselných hodnotách se obecně uživatel špatně orientoval.

Implementovaný modul doplňuje platformu Speech Analytics o možnost práce s výsledky analýzy nahrávek ve vizuální podobě. Už jen možnost zobrazit výsledky v grafové podobě namísto čísel, je velkým přínosem. Hodnoty se ihned stávají uživateli přehlednější a srozumitelnější.

Dále možnost v grafech vizualizovat různé souhrnné charakteristiky je velkým krokem kupředu. Přehledný graf uživateli mnohem rychleji nabídne požadované porovnání na rozdíl od rozsáhlé sady číselných hodnot.

Grafické zobrazení dat s sebou nese také možnost efektivně zkoumat potenciální vzory či závislosti v datech a zpřístupňuje tyto úkoly i běžným uživatelům neznalým

metod statistické analýzy dat. Zmíněné podporuje jednoduché grafické rozhraní aplikace umožňující uživateli rychle navolit jaká data, v jakých kombinacích, z jakého období a v jakém grafu přehledně zobrazit.

10.3 Možnosti rozšíření

Ačkoli vyvinutý vizualizační modul je poměrně rozsáhlým projektem, je z celkového pohledu pro platformu Speech Analytics stále jen náhledem do možností vizuální analytiky dat.

Už během fáze testování se objevilo mnoho připomínek a návrhů a to nejen k rozšíření ale i celkovému konceptu modulu. Řada z nich byla již zapracována ale i nadále zůstává rozsáhlý seznam dalších návrhů a doporučení. Některé z nich nastíním v následujících řádcích.

Jedním z nejdiskutovanějších návrhů bylo rozšíření modulu o statistické grafy, zejména histogram a prstencové/koláčové grafy. Vzhledem k frekventovanosti tohoto návrhu jsem uvažoval o jeho zařazení do této práce, nicméně detailnější analýzou úkolu jsem zjisti, že existují zásadní otázky pro které v době diskuzí nebyla odpověď. Např. Jaké statistické údaje zobrazovat? V jaké fázi budou statistiky počítány? Bude se o to starat modul, nebo budou předpočítané? apod.

Mezi další návrhy patřily otázky na vizualizaci výsledků hledání klíčových slov v nahrávkách. Zde bude potřeba se zamyslet celkově nad formou podání výsledků. Možné nápady jsou: V jaké části se nejčastěji dané slovo/fráze nachází? Jaké je procentuální zastoupení skupin slov v nahrávkách?

Jinou otázkou je umožnění práce s časovými údaji. V tomto případě bude pravděpodobně nutný vlastní zásah do JavaScriptové knihovny Plot.ly.

Z uživatelského hlediska bych také v další fázi vývoje chtěl implementovat možnosti odložené práce s grafy. Tím myslím nabídnout ukládání současného stavu navrženého grafu a pozdější načtení. Taková funkcionalita odstraní nutnost potřeby opakovaně nastavovat veškeré charakteristiky požadovaného zobrazení dat v případě žádosti o znovu získání stejných dat.

Z mého pohledu je nejzajímavějším návrhem na rozšíření modulu doplnění samostatného panelu pro práci s předem připravenými verzemi často používaných grafů. Do takového panelu by se uživatel přepnul po příchodu na stránku vizualizačního modulu. Ten by mu následně zobrazil velmi rychlý přehled současného stavu pomocí předem nadefinovaných grafů.

10.4 Jiná řešení

V této části práce je vhodné zmínit i další podobná řešení a uvést možnosti jejich případného srovnání. Hned zpočátku je nutné říci, že srovnání v rámci platformy Speech Analytics není možné, protože žádné podobné řešení vizualizace dat platforma nenabízí. Pro samotnou vizualizaci dat existuje celá řada za úplatu, či volně

dostupných knihoven. Ty však nelze s řešením vytvořeným v této práci porovnávat, protože jsou zpravidla určené pouze na vykreslení dodaných dat, zatímco práci s nimi (např. agregace, transformace a jiné) nenabízí. Nástrojů, které skloubí jak vizualizaci, tak práci s daty existuje také poměrně velké množství. Takové nástroje ale bývají velmi komplexní a zpravidla nabízeny jako samostatné produkty bez možnosti jejich integrace do vlastních softwarových řešení. Navíc jsou často provozovány jako cloudové služby bez možnosti nasazení v interní síti a vlastní správy.

Celkově tak nezbývá než konstatovat, že nástrojů pro zpracování i vizualizaci dat existuje mnoho, ale žádný z nich nesplňuje stanovená specifická kritéria (možnost integrace do vlastního řešení, vlastní správa, možnost úpravy, lokální práce s daty a další), a proto v rámci této práce vzniklo jedinečné rozšíření platformy Speech Analytics připravené na míru jejím potřebám.

10.5 Ekonomické zhodnocení

Pro výpočet hodnoty vyvíjeného softwarového díla, respektive určení nákladů na jeho vývoj, neexistuje jednotný standardizovaný postup. Každá firma k tomuto úkolu přistupuje odlišně. Výpočet je často podřízen specifickým potřebám té či oné společnosti. Často se lze setkat s velmi hrubými odhady na základě propočtu ceny na hodinu vývoje programátora vynásobené počtem odpracovaných (či plánovaných) hodin. Tento přístup ovšem obsahuje zcela zásadní zjednodušení, jehož důsledek může být ve výsledku pro firmu fatální. Tím zjednodušením je opomenutí všech ostatních nákladů firmy, jež stojí na pozadí vývoje – tzv. provozní náklady. Hodina vývojáře sice může firmu vyjít na řádově stovky korun, je nutné si ale uvědomit potřebu i mnoha dalších lidí a procesů starajících se o chod firmy, čímž programátorovi umožňují provádět jeho činnost. Poslední složkou nutnou započítat do hodnoty vývoje je marže, čili požadovaný zisk z hodiny vývoje.

Na základě výše uvedených řádků jsem postupoval i při výpočtu nákladů na vývoj softwarové komponenty vzniklé v rámci této práce. Hodnoty uváděné ve výpočtu vycházejí z interních odhadů pro platformu Speech Analytics, a to pro první čtvrtletí kalendářního roku 2016.

Celková odhadovaná hodnota hodiny vývoje vychází na 2045 Kč. Uvedená částka zahrnuje: přímé náklady (zejména náklady na práci zaměstnanců), nepřímé náklady (elektřina, nájem, vytápění, hardwarové vybavení a další) a stanovenou procentuální marži.

Následuje výpočet celkové hodnoty vývoje softwarové komponenty. Na tomto místě je nutné zdůraznit, že se jedná pouze o odhadované náklady na vývoj, nikoli o přesnou hodnotu.

$$13 \times 2045 + 273 \times 2045 = 584\,870\text{Kč}$$

Pro přehlednost jsem uvedl rozdělení hodin strávených na projektu na konzultace a hodiny vývoje.

11 Závěr

Cílem této diplomové práce bylo navrhnout a implementovat modul webové aplikace, který rozšíří stávající možnosti prezentace výsledků analýzy nahrávek mluvené řeči v platformě Speech Analytics vyvíjené firmou Phonexia. Ta díky využití vlastních řečových technologií umožňovala extrahovat širokou škálu údajů z jednotlivých nahrávek, postrádala však možnost hromadné, uživateli přívětivé prezentace získaných dat.

V prvních kapitolách se věnuji zejména odůvodnění účelu této práce a podrobnému nastínění metodického postupu. Následuje zpracování teoretických podkladů. Zde se zabývám přístupy k získávání znalostí z dat. Nejprve obecně a následně zmiňuji specifika a postupy k získání znalostí ze záznamů mluvené řeči.

Následující kapitola je věnována platformě Speech Analytics. Obsahem je především představení platformy jako takové, tedy její účel, doména užití a funkcionalita. Vzhledem k nutnosti implementovat modul řešený v rámci této práce do platformy Speech Analytics se věnuji i její technologické stránce, nabízeným řečovým technologiím a principu zpracování nahrávek.

V poslední kapitole teoretické části práce představuji používané technologie a přístupy k tvorbě webových aplikací. Součástí kapitoly je také výběr vhodné knihovny pro vykreslování grafů, jež je použita při implementaci vizualizačního modulu.

Praktická část práce se zpočátku zaměřuje na samotnou analýzu současného stavu platformy a stanovení požadavků. Následuje část – návrh řešení. Zde je postupně na základě uvedeného ukázkového případu užití navržena architektura celého modulu a způsob jeho začlenění do platformy Speech Analytics. Zmíněny jsou také specifické datové struktury vzhledem k nutnosti práce s velkými objemy dat.

Další kapitola se věnuje samotné implementaci řešení. Představuje přístupy zvolené k získávání dat, jejich několikanásobnou transformaci do požadovaných struktur a zejména proces zpracování na základě uživatelských požadavků.

Následuje kapitola popisující výsledky této práce. Nejprve představí grafické rozhraní a popíše princip jeho ovládání. Vzápětí na několika příkladech ilustruje praktické využití implementovaného modulu.

Kapitola diskuze obsahuje zhodnocení celé práce a její přínosy, zmiňuje některé z řešených problémů v implementační fázi práce a nastiňuje možná zlepšení a rozšíření. V závěru pak uvádí ekonomické zhodnocení práce.

Výsledkem této práce tak je modul rozšiřující platformu Speech Analytics o možnost jednoduché a rychlé vizualizace výstupů řečových technologií formou vybraných typů grafů. Svými možnostmi nastavení, seskupení a kombinace různých druhů dat pak umožňuje uživateli vizuálně zkoumat data získaná z telefonních hovorů a hledat v nich potenciální vzory či závislosti. Dále také vytvořený modul zavádí možnost zkoumat demografické informace získané z nahrávek, které doposavad nebyly v platformě nikde dostupné. Vším zmíněným modul přispívá zejména k usnadnění celého

procesu analýzy dat telefonních hovorů a slouží jako další zdroj informací pro podporu rozhodování supervizora. Závěrem tak mohu konstatovat, že práce bez výhrady splnila všechny stanovené cíle.

12 Reference

ANUFF, Ed. API-Centric Architecture: All Development is API Development. *Apigee* [online]. 2014 [cit. 2016-03-23]. Dostupné z: http://apigee.com/about/blog/technology/api-centric-architecture-all-development-api-development.

Apache Wicket User Guide – Reference Documentation. THE APACHE SOFTWARE FOUNDATION. *APACHEWICKET: Free Online Guide for Apache Wicket framework* [online]. 2016 [cit. 2016-04-15]. Dostupné z: https://ci.apache.org/projects/wicket/guide/6.x/guide/single.html.

Brno Speech Application Interface Documentation. *Phonexia* [online]. https://www.phonexia.com, 2015 [cit. 2016-03-13]. Dostupné z: https://www.phonexia.com/docs/bsapi/index.html.

D3.js - Data-Driven Documents [online]. [cit. 2016-03-25]. Dostupné z: https://d3js.org//.

Flot: Attractive JavaScript plotting for jQuery [online]. [cit. 2016-03-25]. Dostupné z: http://www.flotcharts.org/.

GLEMBEK, Ondřej. *Optimization of Gaussian Mixture Subspace Models and Related Scoring Algorithms in Speaker Verification*. Brno, 2012. Doctoral thesis. Brno University of Technology, Faculty of Information Technology.

GOTETI, Harish. CA TECHNOLOGIES. *API Driven Development, Bridging the gap between Providers and Consumers* 2015. Dostupné také z: http://rewrite.ca.com/content/dam/rewrite/files/White-Papers/CATX%20API%20article%20by%20Harish%20Goteti%20FINAL%20Sept.%202015.pdf.

HADERLEIN, Tino a Elmar NÖTH. *Interakce člověk–počítač v přirozeném jazyce (ICP): Příznaky*. 2013. Dostupné také z: http://www.kiv.zcu.cz/studies/predmety/icp/ICP_LS13/icpkap03.pdf.

HAN, Jiawei, Micheline KAMBER a Jian PEI. *Data mining: concepts and techniques*. 3rd ed. Boston: Elsevier, c2012. Morgan Kaufmann series in data management systems. ISBN 9780123814791.

HURST, Jim. *Secure Coding. Practical steps to defend your web apps.: Comparing Software Development Life Cycles* [online]. [cit. 2016-04-13]. Dostupné z: https://software-security.sans.org/resources/paper/cissp/comparing-software-development-life-cycles.

HUSTON, Tom. What is Microservices Architecture? *Smartbear* [online]. [cit. 2016-03-24]. Dostupné z: https://smartbear.com/learn/api-design/what-are-microservices/.

Charts: Interactive charts for browsers and mobile devices. *Google Developers* [online]. [cit. 2016-03-25]. Dostupné z: https://developers.google.com/chart/.

Java Collections – Performance (Time Complexity). *Information Technology Gems: Sharing ICT/InfoSec information for free in a simple, precise and hopefully enjoyable way!* [online]. 2011 [cit. 2016-04-18]. Dostupné z: http://infotechgems.blogspot.cz/2011/11/java-collections-performance-time.html.

Java SE 8: Lambda Quick Start. ORACLE. *ORACLE* [online]. [cit. 2016-04-21]. Dostupné z: http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Lambda-QuickStart/index.html#.

Java Streams – Java Stream Introduction [online]. [cit. 2016-04-20]. Dostupné z: http://www.java2s.com/Tutorials/Java/Java_Stream/index.htm.

Java 8 – Streams. *Tutorialspoint: SIMPLYEASYLEARNING* [online]. 2016 [cit. 2016-04-20]. Dostupné z: http://www.tutorialspoint.com/java8/java8_streams.htm.

KEIM, Daniel, Jörn KOHLHAMMER, Geoffrey ELLIS a Florian MANSMANN (eds.). VISMASTER. *Mastering the information age: solving problems with visual analytics*. Goslar: Eurographics Association, 2010. ISBN 9783905673777. Dostupné také z: http://www.vismaster.eu/wp-content/uploads/2010/11/VisMaster-book-lowres.pdf.

KOHAN, Bernard. Guide to Web Application Development: Guides, Resources, and Best Practices. COMENTUM®CATCH THE WORLD. *Views about enterprise web application development research and reports* [online]. [cit. 2016-04-14]. Dostupné z: http://www.comentum.com/guide-to-web-application-development.html.

LAURENT, Simon St. Web application development is different (and better): On both front and back end, the Web challenges conventional wisdom. O'REILLY®. *Radar: INSIGHT, ANALYSIS, AND RESEARCH ABOUT EMERGING TECHNOLOGIES* [online]. 2014 [cit. 2016-04-14]. Dostupné z: http://radar.oreilly.com/2014/01/web-application-development-is-different-and-better.html.

LOTZ, Mary. Waterfall vs. Agile: Which is the Right Development Methodology for Your Project? *SEGUEIII® TECHNOLOGIES* [online]. 2013 [cit. 2016-04-13]. Dostupné z: http://www.seguetech.com/blog/2013/07/05/waterfall-vs-agile-right-development-methodology.

LUBBERS, Peter, Brian ALBERS a Frank SALIM. *HTML5: programujeme moderní webové aplikace* Vyd. 1. Brno: Computer Press, 2011. ISBN 9788025135396.

MANJUNATH, T.N, S Hegadi RAVINDRA a G K. A Survey on Multimedia Data Mining and Its Relevance Today. *IJCSNS International Journal of Computer Science and Network Security*, 2010(11), 165–170. Dostupné také z: http://paper.ijcsns.org/07_book/201011/20101127.pdf.

MATĚJKA, Pavel a Oldřich PLCHOT. *Speaker Recognition*. Brno, 2015. Dostupné také z: www.fit.vutbr.cz/study/courses/ZRE/public/pred/11_sid_lid/lid.ppt.

MATĚJKA, Pavel. *Phonotactic and Acoustic Language Recognition*. Brno, 2008. Doctoral thesis. Brno University of Technology, Faculty of Electrical Engineering and Communication.

Modern web applications: an overview. *Single page apps in depth: a.k.a Mixu's single page app book* [online]. 2012 [cit. 2016-03-22]. Dostupné z: http://singlepageappbook.com/goal.html.

MyBatis [online]. 2016 [cit. 2016-04-15]. Dostupné z: http://www.mybatis.org/mybatis-3/index.html.

PEHAM, Thomas. 12 web development trends you must know about in 2016. *Usersnap Blog* [online]. 2016 [cit. 2016-03-17]. Dostupné z: http://usersnap.com/blog/web-development-trends-2016/.

PIVOTAL SOFTWARE *Spring* [online]. 2016 [cit. 2016-04-15]. Dostupné z: https://spring.io/.

Plotly.js: The open source JavaScript graphing library that powers plotly. PLOTLY. *Plot.ly* [online]. [cit. 2016-03-25]. Dostupné z: https://plot.ly/javascript/.

PSUTKA, Josef, Luděk MÜLLER, Jindřich MATOUŠEK a Vlasta RADOVÁ. *Mluvíme s počítačem česky*. Vyd. 1. Praha: Academia, 2006. Česká matice technická, roč. 111, č. spisu 502. ISBN 8020013091.

REICH, Michael. Agile v. Waterfall: How to Approach your Web Development Project. *CommonPlaces: An energetic Digital Agency striving to push the*

limits in Design, Marketing, and Development. [online]. 2012 [cit. 2016-04-13]. Dostupné z: http://www.commonplaces.com/blog/agile-v-waterfall-how-to-approach-your-web-development-project/.

RICHARDSON, Chris. Choosing a Microservices Deployment Strategy. *NGINX* [online]. 2016 [cit. 2016-03-24]. Dostupné z: https://www.nginx.com/blog/deploying-microservices/.

RICHARDSON, Chris. Introduction to Microservices. *NGINX* [online]. 2015 [cit. 2016-03-24]. Dostupné z: https://www.nginx.com/blog/introduction-to-microservices/.

RICHARDSON, Chris. Pattern: Monolithic Architecture. *Microservice architecture* [online]. 2014 [cit. 2016-03-24]. Dostupné z: http://microservices.io/patterns/monolithic.html.

RICHARDSON, Chris. Pattern: Service instance per container. *Microservice architecture* [online]. 2014 [cit. 2016-03-24]. Dostupné z: http://microservices.io/patterns/deployment/service-per-container.html.

SCHWARZ, Petr. *Phoneme recognition based on long temporal context*. Brno, 2008. Doctoral thesis. Brno University of Technology, Faculty of Information Technology.

STANGARONE, Joe. 5 big enterprise application development trends of 2016. *Mrc's Cup of Joe Blog: Save money, time, and increase business productivity* [online]. 2016 [cit. 2016-03-17]. Dostupné z: http://www.mrc-productivity.com/blog/2016/01/5-big-enterprise-application-development-trends-of-2016/.

The Java™ Tutorials: Lambda Expressions. ORACLE. *ORACLE: Java Documentation* [online]. 2015 [cit. 2016-04-21]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html.

URMA, Raoul-Gabriel. Processing Data with Java SE 8 Streams, Part 1. ORACLE. *ORACLE* [online]. 2014 [cit. 2016-04-20] Dostupné z: http://www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html.

VARGHESE, Shiju. Web Development Trends For 2015 And Beyond. *Medium* [online]. 2014 [cit. 2016-03-16]. Dostupné z: https://medium.com/@shijuvar/web-development-trends-for-2015-and-

beyond-c2d3c1ef5718#.dxwgd8iyv.

VIJAYARANI, S a A SAKILA. Multimedia Mining Research – An Overview. *International Journal of Computer Graphics & Animation* [online]. 2015, 5(1), 69–77 [cit. 2016-03-11]. DOI: 10.5121/ijcga.2015.5105. ISSN 22313591. Dostupné z: http://airccse.org/journal/ijcga/papers/5115ijcga05.pdf.

WASSON, Mike. ASP.NET – Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. *Msdn.microsoft.com* [online]. 2013 [cit. 2016-03-22]. Dostupné z: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx.

WAYNER, Peter. 21 hot programming trends – and 21 going cold. *InfoWorld* [online]. 2016 [cit. 2016-03-17]. Dostupné z: http://www.infoworld.com/article/3039935/application-development/21-hot-programming-trends-and-21-going-cold.html.

WINN, Chris. 7 Web Development Trends for 2015. *Treehouse Blog* [online]. 2015 [cit. 2016-03-17]. Dostupné z: http://blog.teamtreehouse.com/7-web-development-trends-for-2015.

WINTERBERG, Benjamin *Java 8 Stream Tutorial* [online]. 2014 [cit. 2016-04-20]. Dostupné z: http://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/.

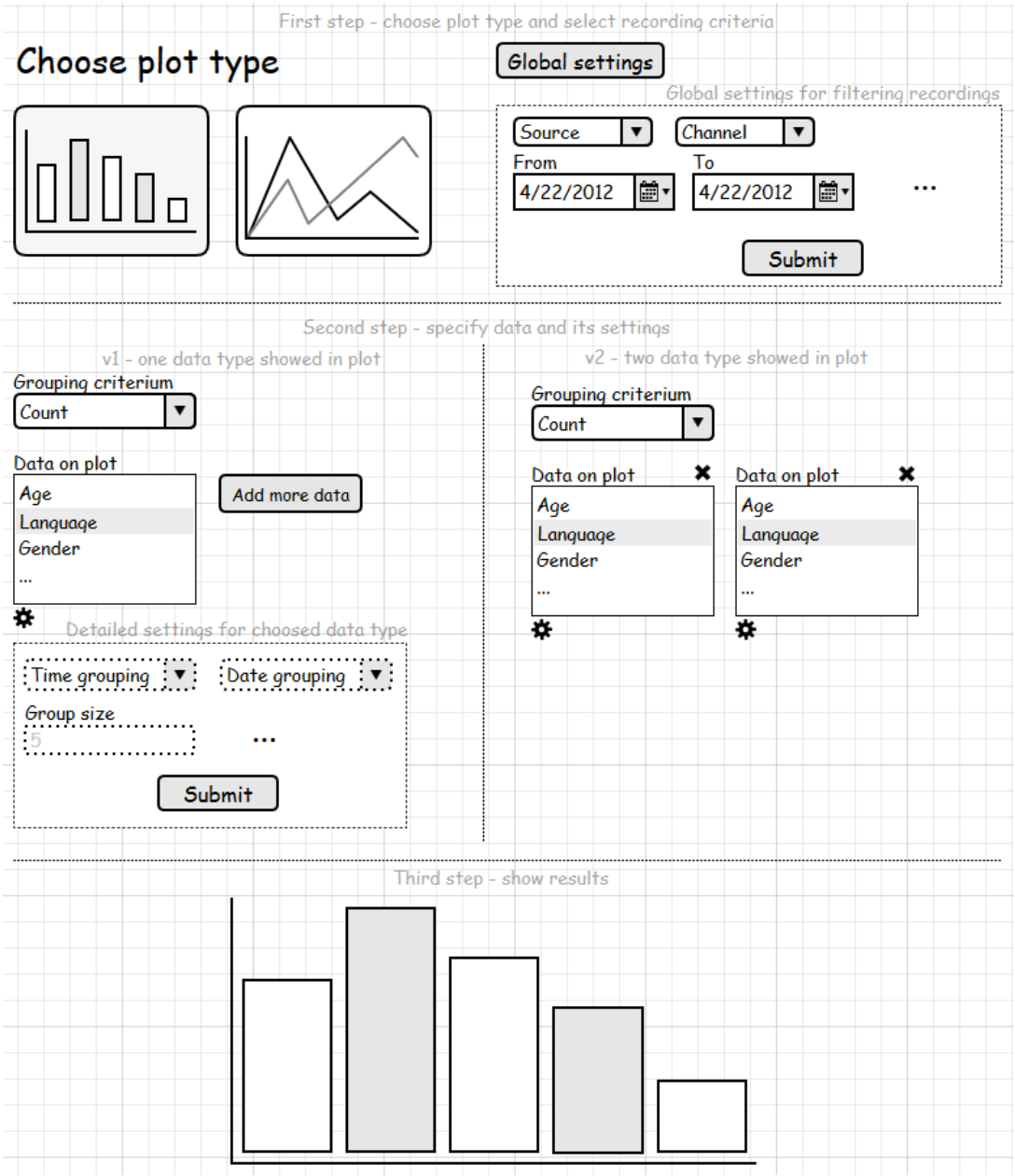
WINTERBERG, Benjamin *Java 8 Tutorial* [online]. 2014 [cit. 2016-04-21]. Dostupné z: http://winterbe.com/posts/2014/03/16/java-8-tutorial/.

WITTEN, I, Eibe FRANK a Mark A HALL. *Data mining: practical machine learning tools and techniques*. 3rd ed. Burlington: Morgan Kaufmann, c2011. Morgan Kaufmann series in data management systems. ISBN 9780123748560.

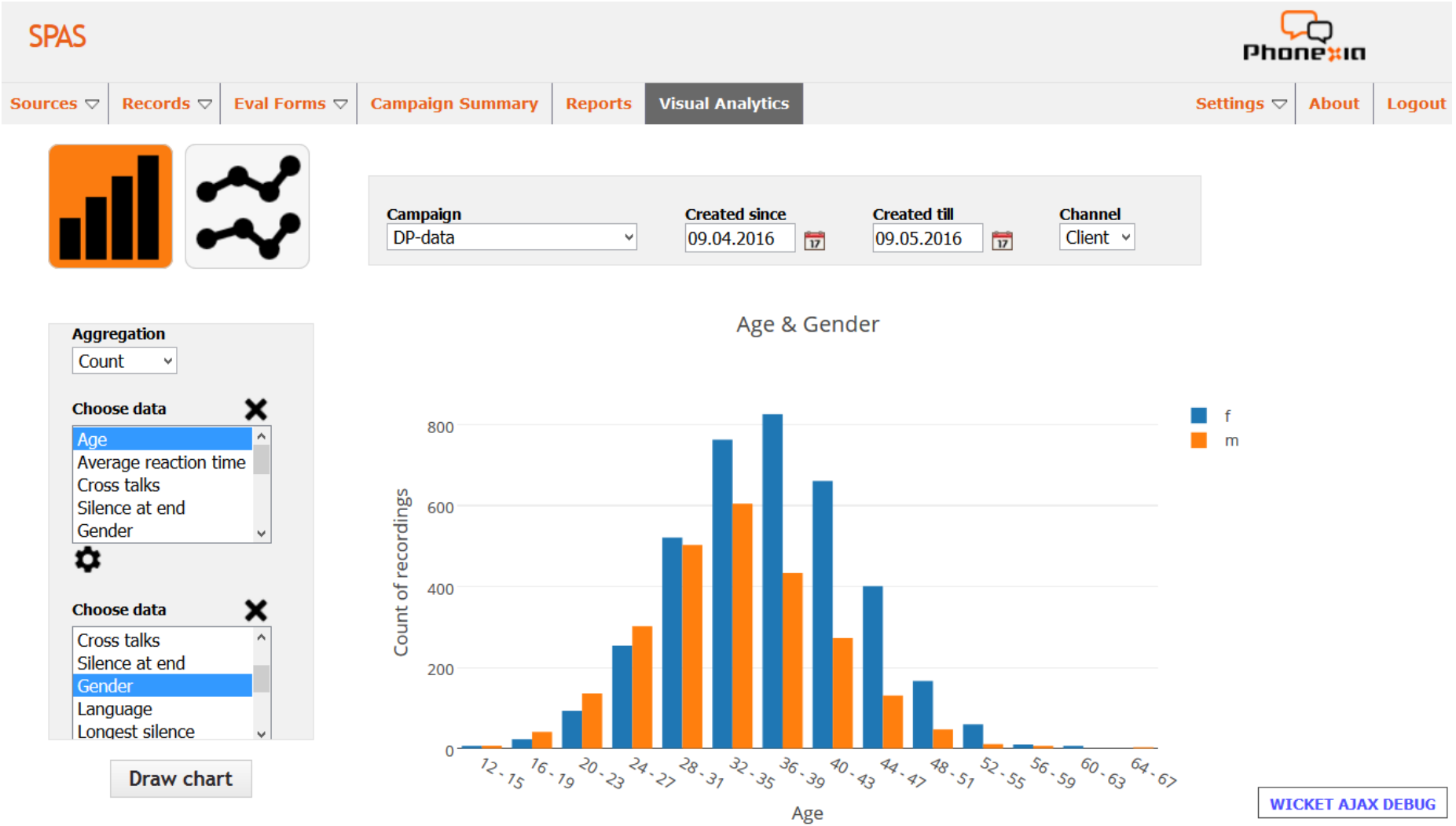
ZRE: Zpracování řečových signálů – ZRE 2014/15. *Fit.vutbr.cz* [online]. Brno, 2016 [cit. 2016-03-31]. Dostupné z: http://www.fit.vutbr.cz/study/courses/ZRE/public/.

Přílohy

A Grafické uživatelské rozhraní



Obrázek 22: Pracovní návrh grafického uživatelského rozhraní



Obrázek 23: Grafické rozhraní modulu integrovaného do platformy Speech Analytics

B Datové CD

Přiložené datové CD obsahuje:

- zdrojové kódy
- samotná práce ve formátu PDF