VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

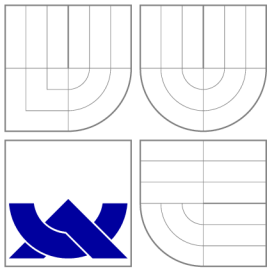
VIZUALIZACE VÝSTUPU Z ŘEČOVÝCH TECHNOLOGIÍ PRO POTŘEBY KONTAKTNÍCH CENTER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

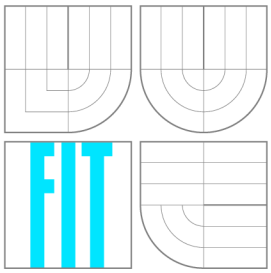
AUTOR PRÁCE
AUTHOR

OLEKSANDR ZHEZHELA

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VIZUALIZACE VÝSTUPU Z ŘEČOVÝCH TECHNOLOGIÍ PRO POTŘEBY KONTAKTNÍCH CENTER

VIZUALIZATION OF OUTPUTS FROM SPEECH TECHNOLOGIES FOR CONTACT CENTERS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — OLEKSANDR ZHEZHELA
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR SCHWARZ, Ph.D.
SUPERVISOR

BRNO 2014

Abstrakt

Diplomová práce se zabývá vizualizaci dat získaných pomocí řečových technologií pro potřeby kontaktních center. Jsou prozkoumány metody získávaní informaci z řečových signálů a existující nástroje, které řeší podobné úlohy. Je analyzován rozsah dat, která lze z řečových technologií získat. Rozebrány procesy a standardy používané v kontaktních centrech. Na základě požadavků pracovníku kontaktních center implementováno uživatelské rozhraní pro vizualizaci dat a audio přehrávač, znázorňující řečová data. Získané poznatky a řešení byli implementovány do nástroje Speech Analytics Server (SPAS).

Abstract

The thesis is aimed on visualisation of data mined by speech processing technologies. Some methods speech data extraction were studied and technologies for this task were analysed. The variety of meta data that can be mined from speech was defined. Were also examined existing standards and processes of call centres. Some requirements for the user interface were gathered and analysed. On that basis and after communication with call centre employees there was defined and implemented a concept for speech data visualization. Gained solutions were integrated into Speech Analytics Server (SPAS).

Klíčová slova

Řečové technologie, zpracování řeči, rozpoznávání řeči, rozpoznávání kličových slov, vizualizace, audio přehrávač, kontaktní centrum, kontrola kvality.

Keywords

Speech technologies, speech processing, speech recognition, keyword spotting, visualization, call center, quality control, human operator.

Citace

Oleksandr Zhezhela: Vizualizace výstupu z řečových technologií pro potřeby kontaktních center, diplomová práce, Brno, FIT VUT v Brně, 2014

Vizualizace výstupu z řečových technologií pro potřeby kontaktních center

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Petra Schwarza. Další informace mi poskytl Mgr. Milan Schwarz

.....
Oleksandr Zhezhela
28. května 2014

Poděkování

Chtěl bych poděkovat Ing. Petru Schwarzovi, Ph.D., a Mgr. Milanu Schwarzovi za veškerou odbornou pomoc a rady, které mi poskytli.

Obsah

1 Úvod **5**

2 Technologie zpracování řeči. **7**
2.1 Automatické zpracování řeči 7
2.2 Předzpracování řečového signálu 8
2.3 Rozpoznávání řeči (Speech recognition) 8
2.4 Jiné úlohy zpracování řeči 11
2.4.1 Rozpoznávání jazyka (Language identification) 11
2.4.2 Rozpoznávání pohlaví (Gender identification) 12
2.4.3 Identifikace mluvčího 12
2.4.4 Časová analýza dialogu 12
2.4.5 Negativní faktory a problémy 13

3 Scénáře použití řečových technologií **15**

4 Problematika vnitřních procesů kontaktních center **17**
4.1 Struktura kontaktního centra 17
4.2 Standardy pro CSP 17
4.3 Požadavky kontaktních center 18

5 Navrch řešení **21**
5.1 Kvantitativní analýza 21
5.2 Kvalitativní analýza 21
5.2.1 Přehrávání audiozáznamů 22
5.2.2 Kritická místa v hovoru 23
5.2.3 Přepis řeči 23
5.2.4 Identifikace mluvčího 23
5.3 Shrnutí 23

6 Řešení SPAS **24**
6.1 Speech Analytics Server (SPAS) 24
6.1.1 Webová aplikace 24
6.1.2 Spring 25
6.1.3 Apache Wicket 27
6.1.4 Základní funkcionalita a možnosti SPAS 28
6.2 Vizualizace řečových informací ve SPAS 30

7 Implementace **34**
7.1 Dynamické tabulky . . . 34
7.1.1 Zvýraznění kritického obsahu . . . 36
7.1.2 Dynamické zobrazování sloupců . . . 37
7.2 Audio přehrávač . . . 39
7.2.1 Použité technologie . . . 39
7.2.2 Architektura . . . 41
7.2.3 Ovládání procesů přehrávaní . . . 43
7.2.4 Zobrazování waveformu . . . 44
7.2.5 Zobrazování kritických míst . . . 45
7.2.6 Zobrazování přepisu nahrávky . . . 47
7.2.7 Verifikace operátora . . . 48
7.2.8 Zapouzdření do Wicket kontejneru . . . 48

8 Diskuze s koncovými uživateli **50**
8.1 Zpětná vazba . . . 50
8.2 Implementace dodatečné funkcionality . . . 50

9 Závěr **51**

A Obsah CD **53**

B Zdrojové kódy **54**
B.1 Implementace komponenty ActionPanel uvedené v příkladu kódu na stráně 35 54
B.2 Implementace sloupce obsahující výsledek KWS s dynamickým přiřazováním stylu. . . . 54
B.3 Implementace třídy CustomDataTableWithColumnOptions rozšiřující základní funkcionalitu Wicket tabulek s použitím vlastních lišt. . . . 55
B.4 Implementace vlastní komponenty lišty pro použití v tabulce s dynamickým zobrazováním sloupce. . . . 56
B.5 Implementace třídy-callbacku pro vykreslování obrázku waveformu pomocí knihovny BSAPI wrapper. . . . 56
B.6 Metoda, která formuje JavaScript funkci inicializaci modulu SpasPlayer . . 58

Seznam obrázků

2.1 Schéma rozpoznávání izolovaných slov ze slovníku [13] 8
2.2 Schéma procesu rozpoznávání klíčových slov [13] 9
2.3 Schéma procesu přepisu řeči na text [13] 9
2.4 Příklad výstupního grafu hypotéz (lattice) [6] 10
2.5 Příklad confusion network 10
2.6 Základní schéma akustické metody rozpoznávání jazyka [10] 11
2.7 Základní schéma fonotaktické metody rozpoznávání jazyka [10] 12

3.1 Media Mining Client, Saillabs 15
3.2 Tovek Analytical deSktop Kit, Tovek 16

5.1 Vztahová sít 22
5.2 Příklad waveformu 23

6.1 Modulární struktura rámce Spring 25
6.2 Zadaní parametrů pro zpracování audio záznamů 29
6.3 Zadání parametrů pro zpracování KWS 29
6.4 Statistické údaje 30
6.5 Vizualizace výstupů řečových technologií pomocí grafů 31
6.6 Vizualizace výstupů řečových technologií pomocí tabulky (AjaxFallbackDefaultDataTable) 32
6.7 Stránka Detail nahrávky 32
6.8 Stránka Hodnocení nahrávky 33

7.1 Příklad AjaxFallbackDefaultDataTable 36
7.2 Výsledek provedené úpravy 37
7.3 Výsledek provedené úpravy 37
7.4 Standardní hlavička tabulky Wicket 38
7.5 Rozšířená hlavička tabulky Wicket 38
7.6 Hlavička tabulky se skrytými sloupci 38
7.7 Implementace modálního okna s vlastnostmi viditelnosti sloupců 39
7.8 Spodní lišta tabulky pro výběr počtu záznamů na stránce 39
7.9 Podpora přehrávaní různých formátu audio v prohlížečích 43
7.10 Ovládací prvky 44
7.11 Rozšíření informace o nahrávce 44
7.12 Realizace waveformu 45
7.13 Realizace waveformu s překreslením 45
7.14 Vizualizace časové osy pomocí BSAPI-wrapper 46
7.15 Vizualizace časové osy pomocí Canvas 47

7.16 Vizualizace kritických míst 47
7.17 Vizualizace titulků 48
7.18 Vizualizace verifikace operátora 48
7.19 Vizualizace verifikace operátora 49

8.1 Výsledná podoba SpasPlayer 50

Kapitola 1

Úvod

V historii lidstva hrála komunikace významnou roli v procesu lidského rozvoje, vzdělávání a pokroku. Mluvená řeč je bezpochyby nejpoužívanějším prostředkem komunikace. Poté, co v roce 1876 Alexander Graham Bell vynalezl telefon, ústní forma komunikace do jisté míry nahradila předávání informací písemnou podobou. Uplynulo už více než 20 let od doby,[12] kdy se telefon stal mobilním zařízením, a tím ještě více používanějším. Podle zprávy ITU (International Telecommunication Union)[8] z roku 2013 používalo 67,5 ze 100 obyvatel evropských zemí aktivně mobilní telefon, 27 obyvatel ze 100 stále aktivně používalo pevnou linku. Telefon byl a zůstává prostředkem rychlého získávání a předávání informací, přivolání pomoci a svůj význam má i při řízení podniků a fungování služeb.

Řada firem si uvědomuje, že jeden z klíčových faktorů jejich úspěšnosti spočivá v jednání s klienty. Nemalá pozornost je proto věnovaná kvalitní zákaznické podpoře. Pro účely efektivní komunikace s klienty jsou již od 50. let zřizována tzv. kontaktní centra neboli call centra [1].

Za více než 60 let své existence prošla kontaktní centra cestou vývoje a velkých změn, a to jak z hlediska zařízení a hardwaru, tak z hlediska jejich managementu a řízení lidských zdrojů. Podle serveru ihned.cz [12] je v současné době v České republice v call centrech zaměstnáno více než deset tisíc lidí a jenom v jednom kontaktním centru se ročně provolá více než 11 milionů minut. Zároveň s růstem počtu zaměstnanců, počtu hovorů a se zvýšením objemu odvedené práce vznikají problémy managementu. Jedním z nejzávažnějších je problém kontroly kvality práce operátorů.

Dodnes se kontrola kvality práce operátorů v kontaktních centrech dělá „manuálně“. Supervizor poslouchá náhodně vybranou nahrávku hovoru a hodnotí práci operátora. Efektivita takového postupu je velmi nízká (kontrolována jsou cca 3% hovorů). Zde vzniká velký prostor pro automatizaci tohoto procesu.

Řečové technologie nám umožňují automatizovaně zpracovávat nahrávky a získávat data potřebná pro analýzu a hodnocení hovorů. Jedním z úkolů této práce je prostudovat existující standardy procesů, běžících v kontaktním centru, ověřit možnosti získávání dat z nahrávek hovorů pomocí řečových technologií a analyzovat požadavky supervizorů kontaktního centra na vizualizaci těchto dat. Na základě této analýzy bude vytvořen návrh uživatelského rozhraní pro účinnou a přehlednou vizualizaci dat, která bude hodnotit supervizor. Tento návrh bude implementován a následně integrován do existujícího softwarového řešení, které se již používá v kontaktních centrech České republiky.

Kapitola 2 obsahuje seznámení s řečovými technologiemi jako takovými. Popisuje zásadní pojmy a metody, které se používají při zpracování řeči. V této kapitole uvedeme hlavní úlohy, které řečové technologie řeší, a také popíšeme rozsah výstupních dat procesů

zpracování řeči. Příklady existujících řešení na trhu IT uvedené v kapitole 3, používajících řečové technologie, pomůžou určit základní scénáře použití daných technologií v praxi. Na základě těchto znalostí navrhneme analýzu a vizualizaci dat, získaných z řeči a jejich aplikace v prostředí kontaktních center.

V kapitole 4 rozebereme strukturu kontaktních center a použití řečových technologií pro zvýšení efektivnosti práce v úseku kontroly kvality. Rozebereme problémy, vznikající kvůli neefektivnímu procesnímu řízení. Definujeme a zdůvodníme požadavky na uživatelské rozhraní pro vizualizaci dat pro potřeby kontaktních center. Na základě těchto údajů navrhneme možná řešení a odůvodníme je v kapitole 5.

Ve zkoumaném kontaktním centru se již používá platforma určená pro analýzu řečových dat. Její základní funkcionalita a architektura spolu s popisem používaných vizualizačních prostředků je popsána v kapitole 6. Určíme části existujícího nástroje, které potřebují rozšíření a zlepšení. Rozšíříme používanou platformu o moduly a funkce, které definujeme na základě komunikace se zaměstnanci kontaktního centra. Proces implementace je obsahem 7. kapitoly. Názor koncových uživatelů na implementovanou funkcionalitu popíšeme v kapitole 8.

Kapitola 2

Technologie zpracování řeči.

2.1 Automatické zpracování řeči

Automatické zpracování řeči umožňuje hlasovou komunikaci mezi lidmi navzájem (kódování) nebo mezi člověkem a strojem.[13]

Do procesu automatického zpracování řeči se zapojuje více věd z různých oborů: technické vědy (matematika, informační technologie — automatizace, odchytávání a zpracování signálů, vzorkování, analýza apod.), přírodní vědy (fyziologie — znalosti o hlasovém ústrojí, akustika — popis fyzických vlastnosti řečových signálů) a humanitní vědy (znalosti o jazyku, stavbě slov, syntaxi apod.)[11]. Využití či nevyužití poznatků jednotlivých věd v procesu zpracování signálů záleží na konkrétní aplikaci a úloze. Každá z těchto věd analyzuje a zpracovává řečový signál na své vlastní úrovni, např. zachycuje a kóduje signál, rozkládá slovo na fonémy, určuje jejich význam. Výsledky takové mnohovrstevnaté analýzy obsahují obrovské množství výchozích dat, která jsou přímo získaná z řeči a jsou k dispozici pro další zkoumání anebo jejich přímou prezentaci. Ve skutečnosti mozek člověka běžně provádí podobnou analýzu při percepci mluvené řeči – převádí řečový signál na srozumitelnou sadu dat. Řečové technologie umožňují realizaci tohoto procesu na stroji, a tím pádem i jeho automatizaci.

Technologie zpracování řeči se používají jak pro ukládání a přenos řečových signálů, tak i pro rozpoznávání řeči (převod řečového signálu na text) nebo rozpoznávání mluvčího[13]. Samostatnou kapitolou je syntéza řeči neboli převedení psaného textu do mluvené řeči, což je opačný proces. V rámci této práce budeme aplikovat řečové technologie jen pro účely rozpoznávání řeči, případně identifikaci mluvčího.

Pro účel získávání dat pomocí řečových technologií a pro jejich následnou vizualizaci je v této práci použita knihovna Brno Speech Core (BSCORE) s rozhraním Brno Speech Application Interface (BSAPI). Brno Speech Core (BSCORE) je sada základních komponent pro řešení úloh rozpoznávání řeči. Implementuje širokou škálu algoritmů např.: čtení řečových dat ze souborů a mikrofonu, zpracovávání seznamu souborů, klasifikaci, dekódování, rozpoznávání fonémů, detekci klíčových slov, rozpoznávání jazyka a rozpoznávání řečníka. Aplikační rozhraní pro BSCORE je objektově orientované. Každý algoritmus má svoji třídu. Třídy jsou přístupné prostřednictvím rozhraní.[4].

Mimo základní knihovnu BSCORE a rozhraní BSAPI, které jsou implementované v jazyce C, existuje rozhraní BSAPI Wrapper, jež slouží k použití dané knihovny v jiných jazycích, například v Javě nebo C#. Pro případy zpracování nahrávek na vzdáleném serveru je k dispozici i REST rozhraní.

2.2 Předzpracování řečového signálu

Ve své podstatě je řečový signál zachycený mikrofonem analogový. Pro zpracování počítačem se musí analogový signál převést do digitální podoby. Nejdřív se spojitý analogový signál převede do posloupnosti úseků diskrétních v čase – vzorků. Poté je hodnota každého analogového vzorku aproximována na číslicovou hodnotu z konečného počtu. Tím se vytvoří kódovaná reprezentace. Tak ze spojitého signálu dostáváme posloupnost údajů v číslicovém tvaru[11].

Z předzpracovaného signálu se extrahují řečové příznaky. Cílem tohoto procesu je zbavit se zbytečných dat a šumu, zvýraznit a odfiltrovat nejzajímavější data a obdržet parametrizovaný signál, totiž popsat signál omezeným počtem hodnot neboli parametrů (features). Tyto parametry budou dále tvořit základ pro následující zpracování řeči, proto je jejich správný výběr zásadní.

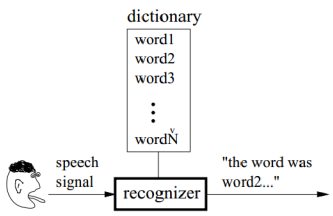
2.3 Rozpoznávání řeči (Speech recognition)

Rozpoznávání řeči je automatický převod mluvené řeči do textu. Rozpoznávání řeči lze rozdělit do tří hlavních skupin podle náročnosti úlohy [13]:

- rozpoznávání izolovaných slov
- rozpoznávání spojených slov z omezeného slovníku
- rozpoznávání plynulé řeči s velkým slovníkem

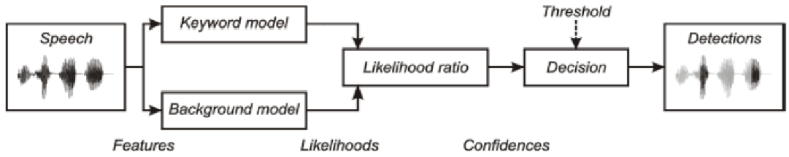
V dnešní době existují dva základní postupy pro rozpoznávání řeči. Jedná se o metody založené na porovnání daného slova s referenčními vzorky a metody založené na použití skrytých Markovových modelů (HMM – Hidden Markov Models).

První skupina metod byla populární už od sedmdesátých let a používala se nejčastěji pro detekci izolovaných slov. Při použití techniky rozpoznávání slova na základě porovnání se vzory je obraz každého rozpoznávaného slova porovnáván jako celek s obrazy slov ve slovníku. Slovo A je zpracováno a spojeno se slovem A* ve slovníku tehdy, když je možné určit slovo A* ze slovníku takové, ke kterému má rozpoznávané slovo A nejmenší vzdálenost, tedy kterému je nejvíce podobné. Určení vzdáleností (míry podobnosti) mezi dvěma obrazy slov se řeší hledáním takové nelineární transformace časové osy jednoho z obrazů, při níž bude hledaná vzdálenost porovnávání obou obrazů nejmenší. Tato transformace a přizpůsobování časové odlišnosti slov se řeší technikou dynamického borcení časové osy (DTW – Dynamic Time Warping). DTW eliminuje časové rozdíly mezi obrazy dvou slov „borcením“ pro dosažení podobností s druhým obrazem. Nevýhoda této metody spočívá v tom, že může pracovat jen s omezeným počtem slov, která jsou zahrnuta v jejím slovníku.



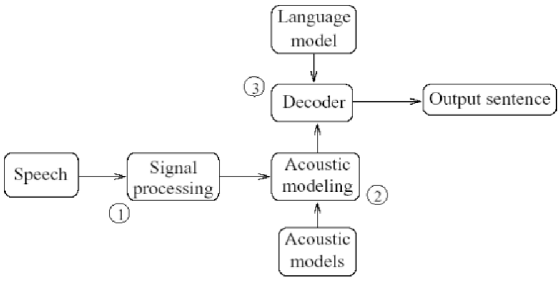
Obrázek 2.1: Schéma rozpoznávání izolovaných slov ze slovníku [13]

Metoda využívající skrytých Markovových modelů se považuje za mnohem efektivnější. Je založena na statistických modelech se skrytými stavy, které používají fonémy[1]. Řečový signál je rozdělen na malé časové stacionární úseky, tvořící krátký stacionární signál. Jednotlivé slovo je pak modelováno zřetězením modelů kratších jednotek — fonémů. Nebo mohou být modelovány i části fonému – stavy fonému. Přesně tak, jak člověk sestavuje slova ze zvuků. Modelování pomocí fonémů má převahu nad modelováním pomocí jednotlivých slov v tom, že všechny modely fonémů jsou okamžitě dostupné a jejich počet je konstantní. Skóre takového modelování je však většinou nižší než u modelování pomocí slov. Modelování pomocí fonémů se ale používá pro řešení složitějších problémů jako rozpoznávání plynulé řeči, kde je znění slov ovlivněno intonací věty nebo předchozím či následujícím slovem.



Obrázek 2.2: Schéma procesu rozpoznávání klíčových slov [13]

Rozpoznávání klíčových slov (Keyword spotting) Pro účel rozpoznávání řeči by řečové příznaky měly pokud možno zachycovat takové charakteristiky, které jsou důležité pro vnímání a pochopení významu projevu. Individuální charakteristiky mluvčího a šum by se měly naopak potlačovat. Zároveň se požadují znalosti o akustice, tvořící akustický model. Akustické modely použité např. v knihovně BSAPI jsou založené na neuronových sítích. Takové sítě se trénují na velkém množství akustických dat. Akustické modely neobsahují znalosti o jazyku a spoléhají se jenom na akustické charakteristiky signálu. Existují techniky diskriminačního trénování, které se považují za lepší, a také adaptace modelu na určitou databázi nebo řečníka. Výstupem zpracování řečových příznaků takovou sítí je pravděpodobnost, se kterou konkrétní část signálu odpovídá hledanému slovu v akustickém modelu. Taková pravděpodobnost se určí pro model úseku řeči obsahující klíčové slovo a pro model stejného úseku řeči klíčové slovo neobsahující. Výstupem je poměr těchto dvou pravděpodobnosti. Pokud tento poměr překročí stanovený práh, je odpovídající klíčové slovo detekováno.

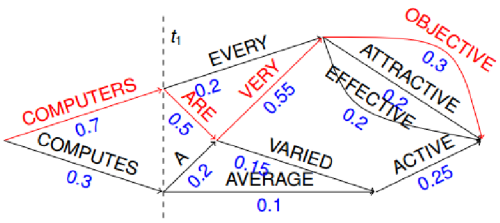


Obrázek 2.3: Schéma procesu přepisu řeči na text [13]

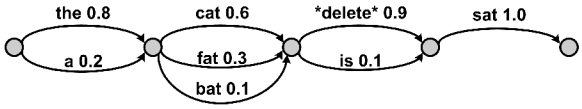
[1]Foném – minimální funkční (abstraktní) jednotka fonologického systému mající rozlišovací platnost, může rozlišit význam slova. (Edvard Lotko, Slovník lingvistických termínů pro filology)

Přepis řeči na text (Transcription) Rozpoznávání řeči s velkým slovníkem (LVCSR – Large Vocabulary Continuous Speech Recognition) je náročnější úkol. Tento proces vyžaduje navíc znalosti o struktuře jazyka (jazykový model), čímž se liší od rozpoznávání jednotlivých klíčových slov. Stejně jako u technologie keyword spotting je pro každý rámec signálu vytvořen vektor řečových příznaků, který je dále klasifikován oproti zadaným modelům. Knihovna BSAPI realizuje akustický model a počítá pravděpodobnosti pomocí směsi Gaussových rozdělení (GMM – Gaussian Mixture Models). Jazykové modely určují pravděpodobnost výskytu sekvencí slov. Jsou realizované pomocí tabulky s pravděpodobnostmi sekvencí trigramů (tří slov), bigramů (dvou slov) a unigramů (jednoho slova). Jsou trénované na velkých textových korpusech. Dekodér na základě rozpoznávací sítě, tvořené jazykovým modelem a akustickým modelem vyhledává optimální sekvence slov – hypotézy. Na základě konfigurace můžeme hovořit o několika typech výstupních dat [6]:

- nejlepší hypotéza (one-best)
- n prvních nejlepších hypotéz (n-best)
- dopředný acyklický graf hypotéz (lattice)
- confusion network



Obrázek 2.4: Příklad výstupního grafu hypotéz (lattice) [6]



Obrázek 2.5: Příklad confusion network

Výstupy rozpoznávání klíčových slov a přepisu řeči na text v knihovně BSAPI

Pro účel rozpoznávání klíčových slov a přepisu řeči na text vrátí knihovna BSAPI seznám objektů třídy `TransOneWord`. Každý objekt třídy `TransOneWord` má tyto atributy:

- `word` — rozpoznávané slovo
- `confidence` — pravděpodobnost se kterou slovo bylo detekováno
- `start` — čas začátku slova v nahrávce
- `end` — čas konce slova v nahrávce
- `segmentId` — id řečového segmentu ve kterém slovo bylo detekováno
- `channel` — kanál ve kterém slovo bylo detekováno

2.4 Jiné úlohy zpracování řeči

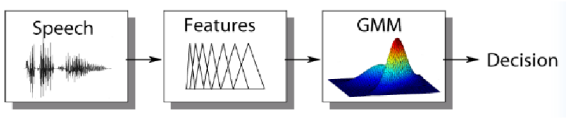
V předchozí části je popsán základní postup při rozpoznávání řeči. V této práci nebyl detailně rozebrán proces tvorby řeči, zachytávání, kódování a dekódování signálu, detekce řečové aktivity, neboť to přesahuje její rámec. Mnohem podstatnější jsou konkrétní aplikace řečových technologií pro řešení specifických úloh.

2.4.1 Rozpoznávání jazyka (Language identification)

Rozpoznávání jazyka je jednou ze základních úloh řešených technologiemi zpracování řeči. Úkolem je rozpoznat jazyk, kterým se mluví ve fragmentu řečového záznamu. Rozpoznávání jazyka může sloužit například k řešení problémů překladu na konferencích, může být využito v bezpečnostním sektoru pro řazení záznamů a zužování pásma hledání záznamů hovorů mluvčích ze specifických etnických skupin. V kontaktních centrech se LID může využívat pro přiřazení příchozího volání operátorovi, který mluví jazykem volajícího. Při volání na linku rychlé pomoci může být rychlost porozumění rozhodujícím faktorem. Tato technologie může mimo jiné sloužit i jako filtr určující nastavení dalších technologií, které budou pokračovat ve zpracování stejných záznamů. Například výstup LID může určovat jazykový model, se kterým bude následovně spuštěno vyhledávání klíčových slov v řeči.

Existují dva hlavní postupy realizace rozpoznávání jazyka:

- akustické rozpoznávání jazyka Tato metoda pro rozpoznávání jazyka využívá vlastnosti řečového signálu. Vlastnosti řeči jako například tón, tempo řeči, či doba artikulace se přímo promítají do vlastností řečových signálů a tím pádem mohou být využity pro následovné porovnávání s řečovými modely. Tento postup je nejvíce vhodný pro krátké řečové segmenty.

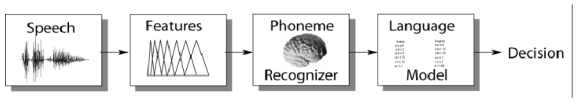


Obrázek 2.6: Základní schéma akustické metody rozpoznávání jazyka [10]

- fonotaktické rozpoznávání jazyka Tato metoda nejdříve zpracovává signál rozpoznávačem fonémů a prezentuje výsledek jako graf(lattices) fonémů (viz. Obr. 2.4). Potom se podle statistik modelů různých jazyků určuje frekvence výskytu dvojic a trojic fonémů. Tyto modely jsou přímo trénované na grafech takovými způsoby, aby obsahovaly nejpřesnější statistické údaje o výskytu skupin fonémů. Jako příklad mohou sloužit trojice fonémů, tvořící samostatná slova „und“ a „uno“. První slovo je německé, druhé italské. V modelu pro němčinu bude mít verze „und“ vyšší pravděpodobnost výskytu, tudíž pokud toto slovo zaznělo v záznamu, lze říci, že se jedná o němčinu. Existují také anti-modely jazyků, které ještě zvyšují šanci na správné rozlišení jazyků mezi sebou.

Výstupem rozpoznávání jazyka řečového záznamu knihovnou BSAPI je:

- seznam názvů jazyků — `getNames()`
- seznam skóre rozpoznání odpovídajících jazyků — `getScores()`
- délka zpracovaného signálu na kanálu — `getSpeechLength(int arg0)`



Obrázek 2.7: Základní schéma fonotaktické metody rozpoznávání jazyka [10]

2.4.2 Rozpoznávání pohlaví (Gender identification)

Další klasickou aplikací řečových technologií je rozpoznávání pohlaví mluvčího. Můžeme říci, že tato oblast aplikace řečových technologií poskytuje i na nízko-kvalitních řečových záznamech výsledky s téměř 100% přesností [7]. Tento druh rozpoznávání je založen na výběru takových řečových příznaků, jako jsou například výška hlasu nebo frekvence základního tónu[7]. Muži mají delší a silnější hlasivky, a tudíž i hlubší hlas. Průměrná frekvence základního tónu řečového signálu u mužů se pohybuje kolem 120 Hz, kdežto řečový signál ženského hlasu má tuto hodnotu přibližně na úrovni 200 Hz. Kromě toho je frekvence formantů[2] u mužů nižší než u žen, což souvisí s délkou hlasového traktu.

Výstupem rozpoznávání pohlaví mluvčího v řečovém záznamu knihovnou BSAPI je následující sada dat:

- počet kanálů v záznamu — `getNChannels()`
- skóre rozpoznávání mluvčího jako ženy pro záznam z příslušného kanálu nebo v celé nahrávce — `getFemaleScore(int arg0), getFemaleScore()`
- skóre rozpoznávání mluvčího jako muže pro záznam z příslušného kanálu nebo v celé nahrávce — `getMaleScore(int arg0), getMaleScore()`
- délka zpracovaného signálu na kanálu — `getSpeechLength(int arg0)`

2.4.3 Identifikace mluvčího

Rozlišuje se identifikace mluvčího („*kdo mluví?*“) a verifikace mluvčího („*mluví Jan Novák?*“). Existují systémy závislé na řečeném textu a textově-nezávislé systémy.

Princip fungování této technologie je podobný ostatním. Jsou vybrány specifické řečové příznaky, na kterých je trénován model, podle nějž se poté bude provádět porovnávání podobnosti (likelihood). Standardně jsou odfiltrovávány šumy, základní tón řeči apod. Řečové příznaky jsou rozpoznáváni natrénovaným modelem (GMM). Výsledkem je poměr věrohodnosti toho, že řečník náleží modelu a věrohodnosti toho, že řečník modelu nenáleží.

2.4.4 Časová analýza dialogu

Detekce řečové aktivity umožňuje analýzu řečových dat ve spojení s časem. Na základě znalosti časových charakteristik řečové aktivity a pozorování jejich dynamické změny lze statisticky analyzovat průběh dialogu. Dostaneme například odpověď na takové otázky: „*Kdy začal řečník mluvit?*“, „*Jak dlouho řečník mluvil?*“, „*Kolík času řečník mlčel?*“, „*Mluvil někdo jiný zároveň s řečníkem okamžik?*“, apod. Na základě takových statistik lze budovat komplexnější analýzu dialogu. Příkladem výsledků takové analýzy mohou být následující vlastnosti:

[2]Formant - (ve fonetice) některý z vyšších tónů tvořící akustickou podstatu hlásky a dodávající jí charakteristické zabarvení. (Edvard Lotko, Slovník lingvistických terminů pro filology) Formanty jsou rezonanční frekvence našeho hlasového traktu.[13]

- rychlost promluvy
- maximální a minimální délka promluvy
- maximální a minimální délka ticha
- počet změn řečníka
- detekce a počítaní skoků do řeči

Implementace knihovny BSAPI prezentuje seznam výsledků, který koresponduje s výše uvedeným:

- rychlost řeči — `getSpeechSpeed()`
- délka promluvy — `getSpeechLength()`
- maximální délka ticha (doba reakce) — `getMaxReaction(int arg0, int arg1)`
- čas výskytu maximální délky ticha — `getMaxReactionPosition(int arg0, int arg1, int arg3)`
- minimální délka ticha (doba reakce) — `getMinReaction(int arg0, int arg1)`
- čas výskytu minimální délky ticha — `getMinReactionPosition(int arg0, int arg1, int arg3)`
- průměrná délka ticha (doba reakce) — `getAvrgReaction(int arg0, int arg1)`
- počet změn řečníka — `getNSpeakerTurns(int arg0, int arg1)`
- detekce skoků do řeči — `getNCrossTalk(int arg0, int arg1)`
- čas skoku do řeči — `getCrossTalkPosition(int arg0, int arg1, int arg2)`
- počet kanálů — `getNChannels()`

2.4.5 Negativní faktory a problémy

Zároveň s vývojem metod a technologií rozpoznávání řeči, ve kterém byl od 50. let minulého století dosažen velký pokrok, se vyskytly problémy, které jsou stálé aktuální. Existuje velké množství externích a interních proměnlivých faktorů, které naši řeč ovlivňují:

- Variabilita mluvčího. Každý člověk má jedinečný hlas a řeč jednoho mluvčího se liší od ostatních. Je to dané fyziologií člověka – jiným hlasovým ústrojím, artikulací, tempem řeči, barvou hlasu atd. Toto výrazně ovlivňuje vstupní řečový signál, a tedy i následující zpracování. Pro zvýšení robustností a eliminování vlivů tohoto původu vznikají další řešení:
  - systémy závislé na řečníkovi (jsou trénovány a schopny rozpoznávat hlas jen konkrétního člověka)
  - systémy na řečníkovi nezávislé (adaptují se na hlas mluvčího).

- Variabilita barvy hlasu. Hlas konkrétního člověka se může měnit podle situace, ve které se člověk nachází: klidný hlas, křik, šepot, pláč, smích. Bolest nebo únava náš hlas rovněž ovlivňují. Doba trvání jednotlivých řečových signálů a také slov je velice proměnlivá. Člověk téměř pokaždé vyslovuje stejné slovo trochu jinak. Navíc znění slova může být ovlivněno jeho zvukovým okolím, tzn. slovy před rozpoznávaným slovem a po něm.

- Variabilita okolí. Běžně zachycené zvukové signály v sobě obsahují šum, který je způsoben vlastnostmi okolí, ve němž se nachází mluvčí. Existuje klasifikace šumů podle jejich vlivu na kvalitu řečového signálu a jejich fyzických vlastností. Šumy mohou být korelované s řečí jako například odraz řečového signálu od zdi, nebo takové, které korelované s řečí nejsou. Nekorelované šumy mohou mít stacionární a nestacionární charakter.

- Variabilita kanálu. Kvalita záznamového zařízení, změna zařízení a rušení kanálu ovlivňují ve značné míře kvalitu vstupního řečového signálu.

Tyto problémy se řeší již řadu let a určitý pokrok už byl dosažen různými vědeckými skupinami i komerčními firmami.

Kapitola 3

Scénáře použití řečových technologií

Mluvená řeč obsahuje obrovské množství metadat, které člověk produkuje neustále. Při použití ve správném kontextu se takové množství dat se může stát silnou zbraní pro marketingové agentury, vědce, pracovníky bezpečnostního sektoru a další. Výše již bylo popsáno několik příkladů použití jednotlivých technologií. V této kapitole popíšeme několik aktuálních scénářů použití řečových technologií v praxi.

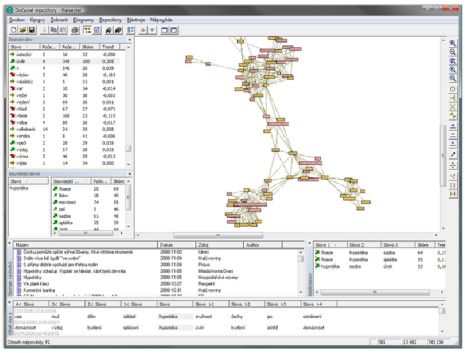
Monitoringové služby. V současném se rychle měnícím světě může být aktuální informace velmi důležitá. Makléři, novináři, politici, zpravodajové se chtějí při rozhodování opírat o co nejaktuálnější stav věcí. Proto služby a produkty zaměřující se na monitoring zpráv a novin v reálném čase získávají čím dál větší popularitu. Jedním z takových řešení je produkt Media Mining Indexer rakouské společností Saillabs. Tento produkt je určen pro sledování toku zpráv z velkého počtů zdrojů (televizní vysílaní, rádio, časopisy, populární internet komunity, Tweeter apod.). Toto řešení vzniklo z technologií rozpoznávání řeči a bylo postupně rozšířeno o další moduly.

Obrázek 3.1: Media Mining Client, Saillabs [1]

Marketingové kampaně a sledování trendů. Velké úsilí a prostředky firmy investují do výzkumu spotřebitelského trhu a poptávky. Už delší dobu vyhledávač Google podporuje

[1] Převzato z Media Mining System - Data Sheet

servis AdWords, který ukazuje reklamu na zboží založenou na tom, co uživatel nejčastěji vyhledává v poslední době. Stejně jako textové vyhledávací systémy můžeme používat například i volání do kontaktních center. Kontaktní centrum nahrává všechny hovory a tím vzniká široké pole pro analýzu řečových informací jako „Big Data“. Indexování získaných řečových informací umožní rychle analyzovat změny zájmů velkého počtu lidí. Toto může být zajímavé nejen z pohledu obchodování, ale i pro řešení kritických situací, havárií nebo výpadků. Například rapidní zvýšení počtu stížností na výpadek elektriky v jedné lokalitě může znamenat opravdu velký problém, kterému pak bude dána přednost na vyřízení. Detekce zajímavých témat se obecně nazývá *topic detection*. Produkt s takovou funkcionalitou nabízí například pražská firma Tovek s.r.o.



Obrázek 3.2: Tovek Analytical deSktop Kit, Tovek [2]

Banky. Vyřízení požadavku pomocí telefonního volání je pohodlným a rychlým způsobem komunikace s bankou. Pro úspěch ovšem každý zákazník musí potvrdit svoji identitu. Většinou se tak děje pomocí hesla anebo ověření čísla smlouvy. Heslo ovšem člověk může zapomenout a smlouvu nemusí mít po ruce. Hlas však má sebou vždy. Ověření identity člověka pomocí hlasového otisku může zjednodušit a urychlit tento proces. Navíc se tímto přidá další bezpečnostní úroveň pro transakci.

Kontaktní centra. Kontaktní centra disponují velkým množstvím audio záznamů, které lze analyzovat. V předchozích příkladech kontaktní centra hrála roli zprostředkovatele dat a výsledků zpracování i analýzy pro svoje externí zákazníky. Řečové technologie ovšem mohou sloužit i pro vnitřní potřeby kontaktního centra. Na základě analýzy konkrétního hovoru lze kontrolovat kvalitu práce operátora, který hovor provedl. V případě neúspěšného hovoru (nespokojený zákazník, nebylo prodáno nabízené zboží, vypukla konfliktní situace) můžeme určit chyby, které operátor udělal a následně ho doplňkově přeškolit. Není nutné vysvětlovat, že lepší práce operátorů a vysoká spokojenost zákazníků znamenají více klientů pro kontaktní centrum, a tak i větší zisk. Na oblast zvýšení kontroly kvality práce v kontaktních centrech se soustředím dále v této práci.

[2] Převzato z http://www.tovek.cz/reseni-analyza

Kapitola 4

Problematika vnitřních procesů kontaktních center

Hlavním zaměřením této práce je zjednodušení vnitřních procesů kontaktních center prostřednictvím vizualizace metadat získaných z řečových technologií, konkrétně v úseku kontroly kvality. Pro modelování problematiky v této oblasti lze popsat jednoduchý příklad, který také bude demonstrovat zjednodušenou vnitřní strukturu kontaktního centra .

4.1 Struktura kontaktního centra

Představme si malé kontaktní centrum s 15 operátory. Při běžném provozu každý operátor provede denně 40 volání. Pro potřeby monitorování je každé volání nahráváno. Tak vzniká denně v takovém malém kontaktním centru 600 záznamů hovorů. Kontrolu práce těchto 15 operátorů provádí jeden supervizor. Kontrola se běžně provádí manuálním odposlechem nahrávky a hodnocením daného hovoru podle určeného seznamu kritérií – hodnotícího formuláře. Tímto způsobem je supervizor schopen za osmihodinový pracovní den ohodnotit cca 30 hovorů, což představuje 5% celkového počtu záznamů. Při tom nebereme v potaz různé negativní faktory, které ovlivňují rychlost a hlavně kvalitu práce supervizorů.

Během komunikace se supervizory jednoho z největších kontaktních center v České republice jsem zjistil, že nejvyšší produktivita práce supervizora je během prvních 2-3 hodin, poté se stává odchytávání chyb operátora při poslechu záznamu hovoru mnohem náročnější a ne zcela efektivní. Skutečný počet kontrolovaných hovorů v praxi nepřesahuje 3 % celkového počtu nahraných hovorů.

Větší kontaktní centra (s počtem operátorů převyšujícím 100) fungují podobným způsobem. Větší počet operátorů je rozdělen do skupin, které kontroluje jeden ze supervizorů. Kromě toho může existovat skupina supervizorů kvality, která má na starosti kontrolu běžných supervizorů, jejich školení a kontrolu dodržení standardů práce a provozu kontaktních center. Takové standardy jsou většinou obecnější a týkají se řízení takzvaných poskytovatelů zákaznických služeb (CSP – Customer Service Provider). Klasickým případem takového poskytovatele zákaznických služeb jsou kontaktní centra.

4.2 Standardy pro CSP

Jedněmi z nejznámějších standardů provozu CSP je sada standardů od společnosti COPC Inc., Customer Operations Performance Cenrter (centrum zaměřené na výkonnost péče

o zákazníky). Společnost COPC Inc. byla založena skupinou úspěšných firem (Microsoft, Intel, Compaq a dalšími) v roce 1996 se záměrem vytvořit standardy řízení péče o zákazníky, které by umožnily odlišit špičkové poskytovatele služeb od průměrných.

Podobné standardy určují postupy řízení a měření výkonu kontaktních center. Existuje několik zásadních provozních ukazatelů, které vystihují úspěšnost řízení projektu. Z hlediska kontaktních center to jsou: průměrný čas zvládnutí transakce (čas na jeden hovor), produktivita operátora (kolik hovorů stihne operátor provést za pracovní den), úspěšnost prodeje (kolik hovorů skončí prodejem) a další [2]. Fáze monitorování transakce, která aktivně využívá tyto provozní ukazatele, je osnovou pro samotný proces řízení výkonu. K této fázi patří monitorování všech druhů transakcí provedených operátorem. Sem patří telefonní hovory, e-maily, zprávy v chatech a SMS. Pro každou z těchto skupin jsou stanovena kritická kritéria, která musejí být splněna. Jestli alespoň jedno z těchto kritérií nebylo splněno (kritické selhání v hovoru), hovor se považuje za „špatný“ a má negativní vliv na statistiku práce operátora. Co lze považovat za kritické selhání? Na tuto otázku pomáhá odpovědět následující klasifikace:

- kritické selhání vzhledem k právním předpisům — falešná informace, rozhlášení osobních údajů jiného zákazníka
- kritické selhání vzhledem k obchodním vztahům — vyjmenování konkurence a konkurenčních produktů, neznalost obchodních podmínek
- kritické selhání vzhledem k zákazníkovi — konfliktní chování, nekvalitní podpora, nesrozumitelnost řeči

První dva typy kritických selhání lze zobecnit jako selhání, které se týká smyslové náplně dialogu a může se detekovat jen při rozboru obsahu dialogu. Třetí typ kritického selhání lze odhalit i podle příznaků v řečovém signálu. Například konflikt se vždycky projevuje zvýšením hlasitosti hovoru, rapidním zvýšením rychlosti řeči, častou změnou mluvčího apod.

Kromě míst kritických selhání existují i další předpisy, které operátor musí dodržovat. Jako příklad můžeme uvést takzvaný callscript neboli plán hovoru, v němž je popsáno o čem, jak a pomocí kterých výrazů má operátor vést hovor. Existují také povinné fráze, které by měl operátor použít během hovoru, například upozornit zákazníka, že hovor může být monitorován. Navíc se také bere v úvahu způsob vedení dialogu, a to rychlost reakce operátora, skákání do řeči komunikačnímu partnerovi, rychlost řeči, případné řečové vady nebo řečové zlozvyky (nadužívání vycpávkových slov, opakování slov).

4.3 Požadavky kontaktních center

Abych mohl přesněji definovat požadavky na data a vizualizaci informací získaných z řečových záznamů, zúčastnil jsem se kalibrace supervizorů jednoho z největších kontaktních center v České republice. Během kalibrace jsem sledoval celkový proces hodnocení hovoru supervizory a také jsem se zapojil do diskuze o stanovení kritérií hodnocení hovorů. Mohl jsem vybrat nejdůležitější data, která supervizor nezbytně potřebuje pro hodnocení jednotlivého hovoru a také kritéria úspěšnosti, podle kterých lze porovnávat hovory mezi sebou. Na základě této komunikace a analýzy je možné vyčlenit hlavní data potřebná pro kontrolu hovorů.

Supervizor má na starosti celý tým operátoru a je za něj zodpovědný. Kontroluje jejich hovory, školí a sbírá statistiky. Jeden operátor provede za den cca. 40 hovorů. Supervizor tedy musí zkontrolovat velké množství nahrávek.

Existují dvě základní varianty zpracování tak velkého počtu nahrávek. První varianta, která se běžně používá v současně době, spočívá v tom, že si supervizor zcela náhodně vybere malý počet nahrávek od každého operátora a ohodnotí je. Sice to je přístup tolerantnější vůči operátorům, ale prokazuje dramaticky nízkou úroveň kontroly, jelikož je vysoká pravděpodobnost, že si supervizor vybere tři „dobré“ hovory z 200 hovorů za týden a „špatné“ mu uniknou. Řečové technologie se přitom využívají jenom pro zrychlení procesu hodnocení konkrétní nahrávky – ověření výskytu zakázaných slov, kontrola konfliktních situací apod.

Druha varianta využívá řečové technologie naplno, a to i při výběru nahrávek pro hodnocení. Supervizoři stanoví kritéria, podle kterých by se měly nahrávky porovnávat. Nahrávky nesplňující zadaná kritéria se analyzují detailně. Takovým postupem docílíme toho, že se při analýze budou odchytávat jenom vadné hovory, a následující kontrola práce se bude provádět efektivněji. Takový postup lze pojmenovat *kvantitativní analýzou.*

Pro účely kvantitativní analýzy jsou použita následující kritéria úspěšnosti vzestupně seřazená podle důležitosti:

- délka hovoru
- rychlost řeči
- počet střídání mluvčího
- rychlost reakce operátora
- počet skákání do řeči
- splnění kritérií týkající se obsahu hovorů (co operátor řekl z toho, co nesměl říct, nebo naopak co neřekl z toho, co říct měl).

Na základě porovnání hovorů podle splnění/nesplnění výše uvedených kritérií jsou potom jednotlivé problematické hovory vybírány pro detailnější zkoumání. V tomto kroku přecházíme ke *kvalitativní analýze.* Během detailního rozboru hovoru má supervizor za úkol zjistit, co skutečně chybného zaznělo v hovoru, čím byla chyba způsobena, a choval-li se operátor správným způsobem. Při takové analýze jednoho konkrétního záznamu potřebuje supervizor možnost poslechu jak celé nahrávky, tak i její části. Ve velkých kontaktních centrech kvůli velkému vytížení existuje pravděpodobnost, že nahrávka hovoru je přiřazená operátorovi A, ale ve skutečnosti patří operátorovi B. Proto se při odposlechu hovoru supervizor nejdřív musí ujistit, že hodnotí správný záznam. Ověření identity je možné díky tomu, že operátoři mají povinnost se představovat na začátku volání. Neznamená to ovšem, že tento proces je pro supervizora zcela jednoduchý. V procesu hodnocení hovoru supervizor vyplňuje standardní formulář, který shrnuje výsledek hodnocení. Velice často se k formuláři připisuje komentář, který může obsahovat citaci z komunikace, která v hovoru proběhla.

Shrneme klíčové informace, které supervizor potřebuje při hodnocení konkrétní nahrávky:

- identita operátora
- zazněla nebo chybějící klíčová slova, a případně čas, kdy byla řečena, s pravděpodobností detekování těchto slov
- výskyt konfliktních situací – křik v průběhu hovoru

- výskyt skoků do řeči v nahrávce
- dlouhé reakční doby operátora

Z uvedeného popisu procesů běžících v kontaktních centrech lze odvodit nejdůležitější principy, které má splňovat vizualizace řečových dat a které budou tvořit základ návrhu uživatelského rozhraní:

1. Analýza záznamů má dvě úrovně abstrakce: zkoumání velkého počtu hovorů a analýza konkrétní nahrávky
2. Kritické jsou výsledky rozpoznávání klíčových slov a časová analýza hovoru
3. Přehrávaní nahrávky s rozšířenou možností ovládání je prioritní funkcionalita, kterou uživatelské rozhraní musí mít.
4. Proces přehrávání musí být propojen a synchronizován s vizualizací konkrétních výstupů řečových technologií.

Kapitola 5

Navrch řešení

V této kapitole budeme postupně procházet seznam požadavků na vizualizace dat a uživatelské rozhraní, definovaných v předchozí kapitole. Pro každou položku z daného seznamu budeme navrhovat implementaci, a v závěru sestavíme celkový koncept vizualizace.

5.1 Kvantitativní analýza

Nejpřirozenějším způsobem porovnávání velkého počtu záznamů mezi sebou na základě rozsáhlého spektra kritérií je porovnávaní pomocí tabulek výsledných dat. Avšak musíme si být vědomi toho, že ve výsledku obdržíme ke každé nahrávce mnoho stejnorodých dat, většinou čísel, která při využití statické tabulky s velkým počtem záznamů budou tvořit nepřehlednou sadu dat. Proto výsledná tabulka musí umět dynamicky spravovat obsah, umožňovat třídění podle jednotlivých sloupců, skrývat nebo ukazovat jednotlivé sloupce, zvýrazňovat kritická data atd. Pomocí dynamické analýzy obsahu tabulky a zvýraznění buněk obsahujících nesplněná kritéria lze přehledně upozornit na problematické záznamy, které mohou být analyzovány podrobněji.

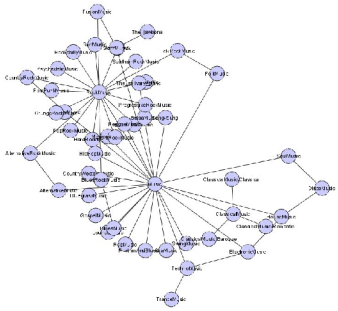
Tabulky nabízejí možnost vizualizace celého seznamu vlastností pro každou nahrávku najednou. Také můžeme zobrazit a analyzovat každou vlastnost nahrávky zvlášť. Tím získáme další statistiku, jak často se vyskytuje určitá vlastnost v celkovém počtu nahrávek. Pro vizualizaci změn takových vlastností hovoru jako délka nahrávky, délka řečové aktivity, rychlost řeči a dalších lze uvažovat o použití grafů. Pomocí grafů můžeme přehledně znázornit závislosti počtu hovorů na hodnotě určitého parametru. Kromě toho dokážeme určit průměrné hodnoty daných parametrů a analyzovat hovory, které tyto hodnoty mají podprůměrné nebo nadprůměrné.

Pokud mluvíme o dynamické analýze velkého počtu nahrávek, v úvahu přihází i další typ vizualizace – neorientované grafy relací neboli vztahových síti (viz. obr. 5.1). Například přehledná ukázka vztahu *n-n „operátor – nesplněná kritéria KWS“* může jasně znázornit seskupení operátorů kolem konkrétního problematického slova.

5.2 Kvalitativní analýza

Kvalitativní analýza by měla umožnit supervizorovi přístup ke všem informacím o nahrávce, které jsou k dispozici. Z částí se to tyká dat, se kterými se již supervizor mohl setkat při

[1] Převzato z http://www.mkbergman.com/



Obrázek 5.1: Vztahová síť[1]

kvantitativní analýze, jako je například počet skoků do řeči. Avšak oproti kvantitativní analýze by tyto informace měly být rozšířené, doplněné a zpřehledněné. Například v tabulce všech hovorů supervizor vidí pouze počty hovorů, kde operátor kritéria pro rozpoznávání slov splnil, nebo nesplnil. Při detailní analýze by měl supervizor mít možnost vidět i konkrétní slova s místem detekce.

5.2.1 Přehrávání audiozáznamů

Nejdůležitějším zdrojem pro analýzu nahrávky je nahrávka sama. Prvotní funkcionalita, kterou požadují supervizoři je přehrávání audia. Jednoduchý a standardní přehrávač audia ovšem pro detailní rozbor nahrávky uživateli stačit nebude. Uvedeme základní případy, kdy standardní funkce přehrávače nejsou dostačující:

- Telefonní hovory mohou trvat i desítky minut a hledat konkrétní slovo může být obtížné a časově náročné
- Hovory mohou obsahovat delší pauzy (uživatel něco hledá nebo čeká na připojení), které supervizor poslouchat nechce
- Při nesrozumitelné řeči může vzniknout potřeba poslechnout si jen jeden vybraný úsek nahrávky

Pro grafické zobrazení audio signálu se používají tzv. waveformy. Waveform představuje grafické znázornění toho, jak se audio signál mění s časem. Použití waveformu jako vizualizačního prostředku přímo vyhovuje naším cílům. Zobrazení grafické podoby signálu nahrávky poskytne supervizorovi přehledný prostředek pro hodnocení hlasitosti hovoru. Umožní přeskočit místa, kde je ticho, a naopak se zaměřit na poslech části hovoru, kde je hlasitost signálu nejvyšší.

Pro případ stereo záznamu je také užitečné rozdělení sdruženého grafického znázornění nahrávky na dvě samostatná znázornění podle jednotlivých kanálů operátora a zákazníka, avšak stereo záznamy se v prostředí kontaktních center používají řídce. Použití rozděleného waveformu nahrávky není tudíž zcela umožněno.

Přehrávání by mělo být synchronizováno s grafickým znázorněním signálu hovoru a aktuálně přehrávané místo by mělo být zvýrazněno. Pro lepší orientování v čase by měl waveform obsahovat i časovou osu.

Navíc by bylo vhodné zobrazit supervizorovi zvlášť jméno operátora a identifikační číslo nahrávky. Ve vnitřní komunikaci mezi supervizory jsou toto hlavní údaje pro odkazování na konkrétní problém.

Obrázek 5.2: Příklad waveformu

5.2.2 Kritická místa v hovoru

Kritická místa v hovoru jsou dány detekcí klíčových slov, skoky do řeči, dlouhou reakční dobou apod. Jsou to důležitá místa, na která se soustředí supervizorova pozornost. Musíme proto umožnit rychlý přístup k poslechu každého kritického místa. Je také vhodné zavést speciální značení pro každý druh kritického místa. Přitom se supervizor může zaměřovat na kontrolu pouze jednoho určitého typu kritického místa. Pro pohodlnou práci v takovém režimu by se neměla ostatní kritická místa zobrazovat. Kritická místa v hovoru by se měla zobrazovat vedle časové osy a ve vztahu k času jejch výskytu.

5.2.3 Přepis řeči

Přepis řeči do textu umožní supervizorovi rychlou analýzou obsahu nahrávky bez nutného poslechu záznamu, což může ušetřit čas. Kromě toho, textová podoba ulehčí uživateli psaní komentářů k hovoru. Supervizor vidí celý obsah hovoru před sebou a citování je možné provádět pomocí kopírování hotového textu.

5.2.4 Identifikace mluvčího

Kvůli externím faktorům musí supervizoři kontrolovat také identitu operátora v nahrávce. Řečové technologie řeší přímo tuto úlohu. Díky poměrně velkému množství audio dat od každého mluvčího (operátora) lze vytvořit kvalitní hlasový otisk mluvčího a na základě porovnání otisku s určeným vzorem můžeme říct, zda v této nahrávce opravdu mluví určitý operátor. Výsledek se zobrazí přímo v rámci přehrávače.

5.3 Shrnutí

Výše popsané požadavky lze shrnout následovně. Pro účely analýzy velkého počtu nahrávek musíme vytvořit vizualizační nástroje, které by dohromady tvořily jednoduchý a flexibilní workflow. Ten bude přehledně provádět uživatele z vyšší úrovně analýzy velkého počtu nahrávek do nižší úrovně analýzy konkrétního záznamu. Vizualizace dat pro porovnání většího počtu záznamů mezi sebou bude mít formu dynamické tabulky s obsahem řízeným uživateli. Vizualizace dat získaných z řečových technologií pro kvalitativní analýzu určité nahrávky bude mít podobu audio přehrávače, který umožní poslech nahrávek a jejich vybraných částí. Přehrávač má obsahovat grafické znázornění signálu spolu s časovou osou. Detekovaná kritická místa nahrávky se musí lišit podle typu a mít přehledné označení, které by vystihovalo jejich význam. Kritická místa by měla být zobrazena s ohledem na čas jejich výskytu a korespondovat s časovou osou nahrávky. Zvláštní sekce přehrávače by měla obsahovat prostor s vypsaným textovým přepisem řeči. U částí přepsaných textů reprezentujících jednotlivé skupiny řečových segmentů by se také měl uvádět čas jejich výskytu.

Kapitola 6

Řešení SPAS

V předchozí kapitole 4 uvedeno základní požadavky pracovníků kontaktních center na řečové informace a návrh možného řešení jejich vizualizace. V úvodní kapitole jsem zmínil, že výsledná implementace bude integrována do existujícího řešení, které se již používá v kontaktních centrech. Tímto řešením je Speech Analytics Server (SPAS), který vyvíjí brněnská společnost Phonexia. V této kapitole krátce popíšeme platformu SPAS a technologie, které SPAS používá. Také popíšeme existující model vizualizace řečových dat a konkrétní úpravy, které vyhovují požadavkům kontaktních center a zároveň zapadají do architektury SPAS.

6.1 Speech Analytics Server (SPAS)

Platforma SPAS je webovou aplikací, která je určena pro přímé použití pracovníky kontaktních center, zejména supervizory a supervizory kvality. SPAS je ve skutečnosti nadstavbou knihovny BSAPI pro práci s řečovými technologiemi, resp. pracuje s jejím rozhraním BSAPI Wrapper pro jazyk Java. Popíšeme základní princip fungování webové aplikace.

6.1.1 Webová aplikace

Webové aplikace si získaly v dnešní době velkou popularitu. Dostupnost internetu, mobilních zařízení a chytrých telefonů dělají informace a aplikace přístupné kdykoliv a kdekoliv. Takové aplikace umožňují poskytování služeb a přístup k informaci bez zásahu do prostředí uživatele – není vyžadována žádná instalace a aktualizace, což je velkou výhodou oproti klasickým desktopovým řešením. Právě proto se poskytovatelé služeb, firmy, banky, e-shopy a další snaží prezentovat svoje produkty i na webu.

> **Webová aplikace** – je aplikace poskytovaná uživatelům z webového serveru přes počítačovou síť internet, nebo její vnitropodnikovou obdobu (intranet)[1].

Webová aplikace je tedy aplikace, která má architekturu *klient-server*. Její prezentační část běží na klientovi, kterým je obyčejný internetový prohlížeč schopný reagovat na požadavky uživatele, posílat je, přijímat a zpracovávat odpovědi z webového serveru. Celá komunikace se uskutečňuje prostřednictvím HTTP protokolu. Webový server je buď hardware (počítač) nebo softwarová aplikace, která je schopná přijímat HTTP požadavky od vícera vzdálených klientů, zpracovávat je a posílat požadovaný webový obsah nebo případnou chybu jako odpověď. Nejpoužívanějším webovým serverem v současné době je Apache

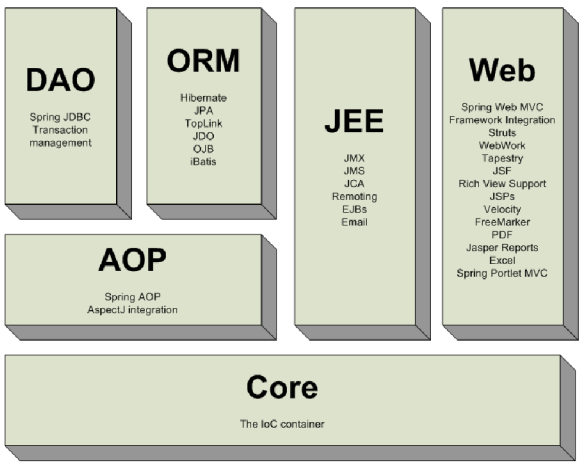
[1] http://cs.wikipedia.org/wiki/Webová_aplikace

HTTP Server. Webový server obsahuje statický HTML obsah, který může být poslán jako odpověď klientovi. Pro generování dynamické odpovědi používá webový server moduly, jež podporují spuštění kódu a skriptu, které mohou generovat dynamický webový obsah, zpracovávat obsah z HTML formulářů, uchovávat stavy objektů apod. V případě použití jazyku Java pro generování dynamického obsahu se daná Java aplikace jmenuje Java HTTP Servlet, nebo zkráceně servlet. Webový server, aby byl schopný spouštět a komunikovat s Java servlety musí mít Servlet kontejner – komponentu řídící životní cyklus servletů, mapující URL cesty požadavků ke správné aplikaci, řídící práva přístupu apod. Jeden z nejznámějších kontejnerů servletů je Apache Tomcat.

Pro řešení běžných úkolů (připojení k databázi, logování apod.) Java, podobně jako ostatní jazyky používané pro generování dynamického obsahu webových stránek, používá řádu externích knihoven. Podobné knihovny používané pro vývoj webových aplikací jsou specifičtější než knihovny používané pro běžný vývoj desktopových aplikací. Pro podporu komplexního vývoje webových aplikací v jednom programovacím jazyku s rozšířenými možnostmi vznikají tzv. webové aplikační rámce (frameworky). Jeden z takových frameworků představuje Spring Framework.

6.1.2 Spring

Základní část (Core) rámce umožňuje vývoj jakýchkoliv Java aplikací, ale na rozdíl od ostatních rámců Spring velice usnadňuje vývojářům programování všech vrstev J2EE aplikace, od nejnižší datové vrstvy až po nejvyšší prezentační[2]. Toto je realizováno pomocí celé řady rozšíření a modulů. Kromě toho Spring konzistentně podporuje velké množství softwarových řešení, jako například Hibernate, MyBatis, Struts či Wicket. Spring má modulární strukturu a programátor si může vybrat jen ty moduly a balíčky, které jeho aplikace skutečně potřebuje. Takovým způsobem se zachovává malá velikost balíčku.



Obrázek 6.1: Modulární struktura rámce Spring
[3]

Svoji popularitu Spring ovšem získal nejen kvůli možným rozšířením, ale také díky principům spravování objektů, řízení objektových závislostí (Dependency Injection, DI), spravování zabezpečení (Spring Security), řízení transakcí a deskriptivnímu způsobu konfigurace aplikace. Velké J2EE aplikace lze implementovat pouze s využitím jednoduchých

[2] http://www.tutorialspoint.com/spring/spring_overview.htm

Java objektů (POJO), a proto není potřeba mít EJB (*Enterprise Java Bean*) kontejner jako aplikační server, který je náročnější na spravování a vyžaduje více zdrojů jako paměť nebo výpočetní výkon. Při práci se Spring je vyžadován pouze obyčejný kontejner servletů jako např. Apache Tomcat.

Vkladání závislostí (Dependency injection)

Popíšeme jeden ze základních konceptů rámce - vkládání závislostí. DI je speciální realizací návrhového vzoru Inversion Of Control (IoC, Obrácení řízení).

Víme, že při napsání velkých a složitých Java aplikací musíme dodržovat jednoduchost a maximální možnou nezávislost jejich tříd. Tím zvýšíme možnost opakovaného použití tříd, škálovatelnost systému a produktivní testování.

IoC uvádí princip, podle nějž se programátor může spolehnout na rámec v kroku vytváření objektů před jejich použitím. Programátor jen sdělí rámci co chce použít a rámec se sám postará o vytváření, inicializaci a předaní potřebného objektu do místa použití. Uvedeme příklad takového scénáře. Představme si aplikaci, která vytváří auto. Každé auto má motor. Bez použití IoC bychom kód třídy Car napsali takto:

```
 public class Car {
   private Engine engine;
   public Car() {
      engine = new Engine();
   }
}
```

Tímto vytváříme takové propojení mezi Car a Engine, které závisí na implementaci konstruktoru `Engine()`. Budeme-li při dalším vývoji tento konstruktor měnit na `Engine(int power)`, pak budeme muset refaktorovat kód ve všech třídách, které mají tuto závislost. Pro aplikaci, ve které máme 100 různých druhů aut, objektů tříd rozšiřujících třídu Car, tato situace přináší architektonické problémy. Nebo rozhodneme-li se v budoucnu používat další implementaci konstruktoru třídy Engine `Engine(int power){return new DieselEngine(power)}`, potom by třída Car závisela už na správné inicializaci objektu DieselEngine. Vytváříme nepřehlednou hierarchii a složitě spravovatelné závislosti. Bylo by mnohem jednoduší vědět, že máme k dispozici objekt Engine a můžeme ho rovnou použít. Přepíšeme tento příklad s využitím IoC ve Spring.

```
public class Car {
   private Engine engine;
   ...
   // a setter method to inject the dependency.
   public void setEngine(Engine engine) {
      this.engine = engine;
   }
   // a getter method to return engine
   public Engine getEngine() {
      return engine;
   }
        ...
}
... \\ Spring XML configuration file
   <bean id="engine" class="com.tutorial.Engine"></bean>

   <!-- Definition for Car bean using inner bean -->
   <bean id="car" class="com.tutorial.Car">
      <property name="engine">
         <bean id="engine" class="com.tutorial.Engine"/>
       </property>
   </bean>
...
```

Teď se objekt Car nemusí starat o inicializaci Engine a o jeho implementaci. Engine se vytvoří zvlášť a bude předán objektu Car při jeho instanciaci. Popis závislostí je přenesen do XML souboru, který pak použije Spring. Takovým způsobem lze delegovat zodpovědnost rámci.

Toto je typický a jednoduchý příklad konfigurace a použití rámce Spring. Velké aplikace jsou ovšem komplikovanější jak z pohledu konfigurace, tak z pohledu architektury. Připojení dalších důležitých modulů jako například logování (Apache Log4J), plánování a časování akce (Quartz), testování (JUnit) či bezpečnosti (Spring Security) vyžaduje jistou míru zkušeností. Ve výsledku obdržíme rozsáhlou strukturu vzájemně propojených konfiguračních XML souborů, které úplně definují nejen inicializaci a spouštění aplikace, ale občas i její chování.

6.1.3 Apache Wicket

Apache Wicket je komponentně orientovaný rámec pro jazyk Java, což znamená, že aplikace je konstruována pomocí skládání funkčních komponent, které jsou zároveň tvořené menšími komponentami, do jednotlivých webových stránek. Má velkou výhodu díky znovupoužití jednotlivých komponent, komponenta může být dokonce přenesena i do jiné aplikace.

Jako naprostá většina současných frameworků Wicket realizuje návrhový vzor *Model-View-Controller (MVC)*. Ale na rozdíl od ostatních je Wicket postaven na principu použití komponent s úplně určeným stavem, a to na straně serveru. Podle autorů Wicketu kontrola stavů objektů na stráně serveru chybějící ve většině rámců byla jednou z motivací[4] pro tvorbu tohoto frameworku. Každá stránka je tvořena stromem komponent se stavy a modelem. Framework uchovává tyto stromy a stará se o HTTP interakce mezi webovým rozhraním a stavem Java objektů na serveru. Použití POJO usnadňuje další práci s perzistentní vrstvou a databází.

Pro tvorbu uživatelského rozhraní Apache Wicket používá šablony v jazyce XHTML. Funkčnost a vzhled jsou udržovány zvlášť. Propojení je realizováno pomocí speciálního atributu `wicket:id`, který je definován při inicializaci objektu v Javě. XHTML doplněný takovým způsobem lze upravovat a stylizovat pomocí CSS a všech existujících nástrojů pro tvorbu HTML stránek. To ovšem přináší i určité komplikace spojené s nutností kontrolovat odpovědnost struktury Java komponent a strukturu výsledného XHTML a zároveň i stejné hodnoty atributu `wicket:id` v HTML a Wicket komponent.

Kromě toho Wicket prezentuje širokou podporu pro práci se vzhledem a chováním aplikace na stráně uživatele (přímá podpora jQuery - od verze Wicketu 6.0) a rychlou AJAX komunikaci se serverem. Wicket nabízí validace uživatelských vstupu, různé druhy tabulek, stránkování, přímé řízení změn stylu z Java přes AJAX a mnoho dalších výhod se kterými se ještě setkáme během implementace navržené vizualizace.

Výše uvedená problematická místa se týkají spíše úsilí a pohodlí programátora, existují však i čistě architektonické záležitosti, které ovlivňují vývoj webu v tomto frameworku.

- Wicket nespravuje svoje objekty

  Wicket neřídí životní cyklus svých komponent, což znamená, že jakákoliv stránka nebo komponenta může být potenciálně vytvořená kdekoliv jen pomocí operátoru *new*. Toto dělá injekci závislostí obtížnou jak už bylo zmíněno v předchozí podsekci [5].

[4] http://wicket.apache.org/meet/introduction.html

[5] Podrobný popis, techniku a příklad popisuje také Martin Fowler http://martinfowler.com/articles/injection.html

- modely a objekty ve Wicketu jsou serializovatelné

  Wicket uchovává stromy komponent se stavy v objektu sezení. Pro přenos a výměnu dat v tomto prostředí musejí data být serializovaná. Toto přináší problém z pohledu závislostí, které nejsou serializovatelné, resp. jejich serializace je velmi nežádoucí. I když úspěšně provedeme kaskádovou serializaci celého stromu závislostí, při deserializaci daného objektu už nebude součástí kontejneru, nýbrž jeho samostatným klonem. Kromě toho správná serializace objektu je důležitá pro práci s perzistentní vrstvou.

Tyto problémy můžeme elegantně řešit pomocí Spring Framework. Aplikace ve Wicketu obsahují globální aplikační objekt, který je potomkem Wicket třídy Application. Takový objekt se vytváří pouze jednou, neobsahuje žádnou uživatelskou informaci a zůstává pořad stejný pro všechny ostatní komponenty. Proto se takový objekt nikdy neserializuje a je dobrým místem pro implementaci návrhového vzoru Service Locator. Tím docílíme zachování principu IoC a nízkého provázání objektů. V aplikačním kontextu pro Spring můžeme nadefinovat potřebné servisní třídy, externí knihovny, cesty ke konfiguračním souborům apod. Kromě toho Wicket přímo nabízí použití vlastní továrny `SpringWebApplicationFactory` (návrhový vzor Továrna), která si dokáže převzít celý aplikační kontext vytvořený Spring.

Nadefinujeme jednoduchý servlet a uvedeme jeho parametry inicializace. V daném případě ukážeme, že pro inicializaci musí být použita třída `SpringWebApplicationFactory`.

```
<servlet>
    <servlet-name>wicket</servlet-name>
    <servlet-class>org.apache.wicket.protocol.http.WicketServlet</servlet-class>
    <init-param>
        <param-name>applicationFactoryClassName</param-name>
        <param-value>org.apache.wicket.spring.SpringWebApplicationFactory</param-
            value>
    </init-param>
    <init-param>
        <param-name>applicationBean</param-name>
        <param-value>wicketApplication</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>

</servlet>
```

Proto aby Spring byl schopen načíst a inicializovat aplikační kontext musíme určit konfigurační soubory kontextu a servis zodpovědný za načtení jejich obsahu.

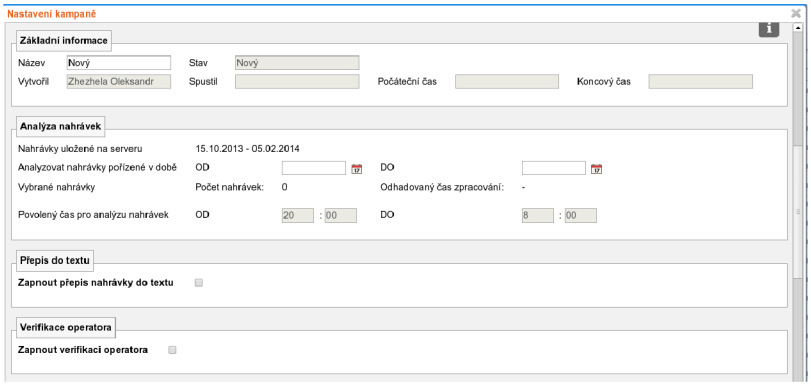
```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener
        -class>
</listener>
```

Další konfigurování záleží na počtu modulů a balíčků, které aplikace potřebuje. V konfiguračních souborech aplikačního kontextu definujeme Spring beany a propojujeme je podle architektury.

6.1.4 Základní funkcionalita a možnosti SPAS

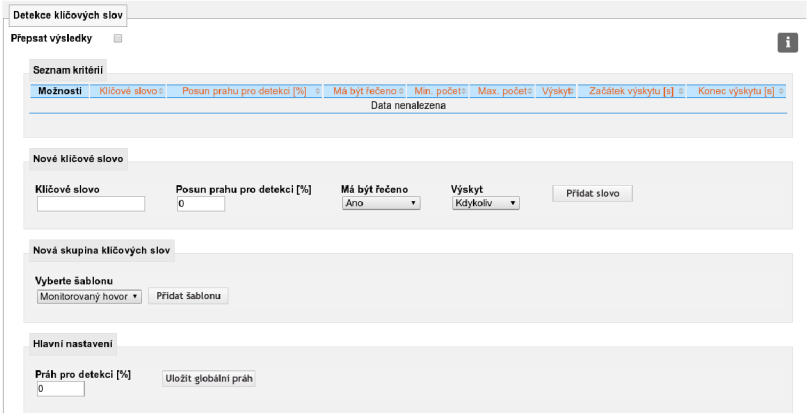
Spojení Spring a Wicket vytváří zázemí pro tvorbu rozsáhlé a stabilní webové aplikace. Integrovaní do architektury knihovny BSAPI-wrapper pro jazyk Java umožňuje zpracování nahrávek řečovými technologiemi. Spring podporuje použití OpenSymphony Quartz

Scheduler, který umožňuje plánování spuštění úloh. Pro správné zpracování platformou SPAS musejí být nahrávky umístěny do určité složky zadané v konkrétní konfiguraci a být roztřízeny do definované struktury složek. Pro SPAS je charakteristické, že jakmile budou nahrávky umístěny v definované složce, budou téměř okamžitě zpracovány. Provede se případný převod do formátu WAV (je vyžadován pro práci s řečovými technologiemi) a časová analýza nahrávky. To poskytne uživateli základní informace o nahrávce v prostředí platformy SPAS. Ostatní řečové technologie jsou buď náročné na provoz, nebo nejsou vyžadovány pro analýzu všech nahrávek. Jejich použití lze ovšem jednoduše nastavit. Z pohledu uživatele je pouze potřeba určit, které nahrávky chce uživatel zpracovat, a zadat nezbytné parametry. Na obrázku 6.2 je zobrazeno okno s možnostmi zadání nahrávek pro zpracování a konfigurace použití technologií pro přepis řeči a verifikaci mluvčího.



Obrázek 6.2: Zadaní parametrů pro zpracování audio záznamů

Použití technologie rozpoznávání klíčových slov vyžaduje složitější nastavení. Musíme určit vyhledávaná slova, nastavit prahy detekování, určit, zda detekované slovo mělo zaznít, nebo ne a jestli slovo má (popřípadě nesmí) říct operátor nebo zákazník. Ve SPAS lze také definovat časový interval v nahrávce, ve kterém bude dané slovo vyhledáváno. Pro často vyhledávaná slova a seznamy slov lze definovat tzv. šablonu kriterií, kterou lze používat víckrát na různých zdrojích.



Obrázek 6.3: Zadání parametrů pro zpracování KWS

O zpracování nahrávek, ukládání výsledků do databáze, sběr statistik a o další akce SPAS zajistí bez jakékoliv účasti uživatele. Výsledky analýzy může uživatel zkoumat ve

více podobách. Na svých stránkách SPAS nabízí jak možnost interaktivního zkoumání obdržených dat, tak generování celé řady reportů ve formátech PDF a XML.

Počet hovorů	Prodejních hovorů	Detekce KS	Hodnotící formuláře
▲ 708 (+23%)	➤ 10% (0%)	▼ 35% (-63%)	▼ 73% (-6%)
▼ 779 (-4%)	▼ 8% (-4%)	▼ 53% (-40%)	▲ 91% (+48%)
1069	1%	37%	50%
▲ 1338 (+15%)	➤ 11% (0%)	▼ 46% (-47%)	➤ 91% (0%)
▲ 813 (+25%)	▲ 6% (+1%)	▼ 33% (-64%)	▼ 63% (-10%)
▼ 122 (-57%)	▼ 2% (-3%)	▼ 34% (-64%)	nezjištěno
▲ 41 (+90%)	➤ 0% (0%)	▼ 25% (-75%)	nezjištěno
▼ 276 (-24%)	▼ 1% (-4%)	▼ 42% (-53%)	nezjištěno
▲ 1522 (+24%)	▼ 8% (-2%)	▼ 43% (-53%)	▼ 76% (-17%)
▲ 267 (+48%)	▲ 6% (+2%)	▼ 34% (-64%)	▼ 73% (-9%)

Obrázek 6.4: Statistické údaje

6.2 Vizualizace řečových informací ve SPAS

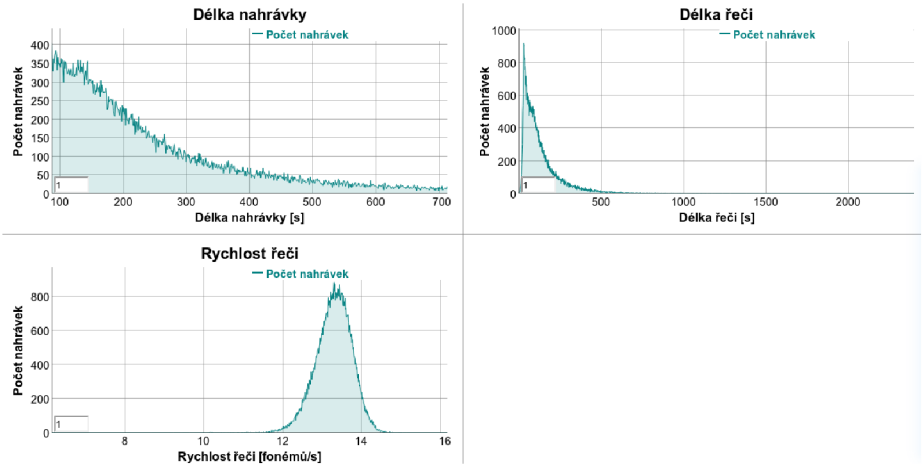
Výsledkem výše popsaného nastavení a zpracování je velký obsah dat, který musíme uživateli zpřístupnit v přehledné podobě. Platforma SPAS již v sobě zahrnuje dost širokou sadu vizualizačních prostředků, některé však vyžadují vylepšení a rozšíření.

Kvantitativní analýza

Pro účely porovnávaní velkého počtu záznamů SPAS používá tabulky a grafy. Vizualizace pomocí grafů je založena na principech popsaných v předchozí kapitole. Pro znázornění jsou vybrána taková kritéria jako délka nahrávky, délka a rychlost řeči. Použité grafy jsou dynamické a lze je ovládat: posouvat osy, zvýrazňovat intervaly, zvětšovat a zmenšovat měřítka. Vyplněná plocha grafu představuje pole hovorů. Označením částí grafu uživatel vybírá všechny hovory, jejichž parametry zapadají do vybrané části grafu. Vybranou sadu hovorů lze přímo přidávat do seznamu pro další analýzu. Například chceme-li prozkoumat nejdelší hovory zároveň se zmáčknutým tlačítkem *Control*, vybereme myší interval na grafu. Hovory patřící k této oblasti grafu budou automaticky uloženy do databáze pro další analýzu s rozšířenými parametry. Pro nezkušeného supervizora, který nezná obvyklé hodnoty parametrů, grafy přehledně představují průměrné hodnoty každého parametru a ukazují extrémy. Implementace grafů je založená na použití JavaScript knihovny Dygraph. Obrázek 6.5 slouží jako příklad takových grafů.

Podle zpětné vazby supervizorů, běžně používajících SPAS jsou grafy užitečným vizualizačním prostředkem přitom se nepoužívají příliš často pro detailnější zkoumání hovorů. Mnohem častěji jsou grafy využívané v prostředí supervizorů kvality pouze pro prezentaci obecné dynamiky změn parametrů a jako vedlejší statistiky.

Následná analýza se provádí pomocí tabulek. Apache Wicket nabízí celou sadu komponent, které implementují prezentaci tabulek. Liší se podle funkcionality, kterou komponenta zvládá. Všechny typy jsou ovšem založeny na stejném principu použití *opakovačů (Repeaters)*. Daný typ komponent umí zobrazit seznam objektů tak, že každý objekt ze seznamu představuje řádek tabulky a jednotlivé pole obsahuje hodnotu určitého atributu objektu nebo jinou komponentu Wicket. Vzhled, vlastnosti a chování každého sloupce musíme definovat zvlášť. Proto definujeme seznam jednotlivých komponent, představující seznam sloupců. Při iterování nad seznamem zobrazovaných objektů je značení jednotlivých sloupců



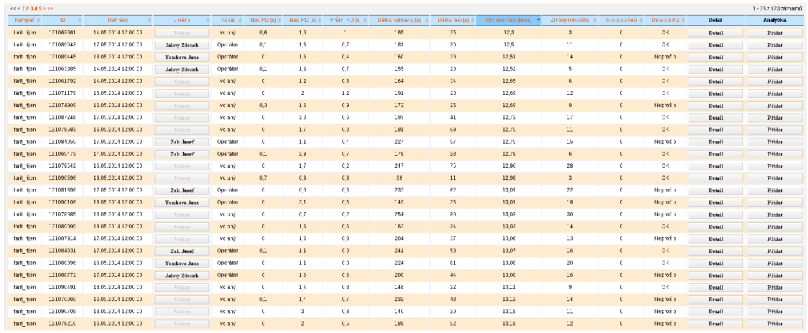
Obrázek 6.5: Vizualizace výstupů řečových technologií pomocí grafů

doplněno odpovídajícím obsahem. Wicket nabízí poměrně širokou řadu komponent plnících účely znázornění různých tabulkových druhů. Jsou k dispozicí následující implementace:

- `RepeatingView` – nejjednodušší implementace, odpovídající výše popsanému příkladu
- `RefreshingView` – každé pole je schopno dynamicky obnovovat svůj obsah při změně stavu odpovídajícího Java objektu
- `AbstractPageableView` – předchozí varianta s možností stránkování
- `GridView` – statická jednoduchá tabulka s daty stejného druhu v každém poli, definuje se jen vzhled pole a počet sloupců a řádků; také je zde možnost stránkování
- `DataView` – injekce dat se provádí nikoliv iterováním, ale pomocí objektu implementujícího rozhraní IDataProvíder, což výrazně zjednodušuje práci s daty z databáze
- `DataGridView` – je kombinací DataView a GridView; injektování dat se provádí pomocí provideru a vzhled sloupců definuje objekt ICellPopulator
- `DataTable` – má nejvíc rozšířenou funkcionalitu: seřazení, stránkování, zpráva o celkovém počtů záznamů a aktuální stránce, stylizování sudých a lichých záznamů, filtrování, dynamické záhlaví sloupců atd.; rozšířením je AjaxFallbackDefaultDataTable s možností detailního a hlubokého konfigurování

I když standardní komponenty Wicket zvládají širokou řadu funkcí, je nutné přidat několik dalších. Na obrázku výše lze vidět, že tabulka obsahuje poměrně velký počet sloupců, což při malé velikosti obrazovky počítače způsobuje problémy. Některé sloupce by mohli být odstraněny, avšak každý supervizor pracuje s odlišnou sadou dat. Proto musíme implementovat vlastní realizaci tabulky, která umožní uživateli konfigurovat viditelnost samostatných sloupců podle jeho potřeb.

Kromě ostatních údajů tabulka zobrazuje i výsledek detekce klíčových slov (třetí sloupec zprava) a to binární způsobem – buď jednoznačné „OK“ anebo rezolutní „Wrong“. To by vyhovovalo v případě, že bychom hledali jenom jedno slovo v nahrávce. SPAS ovšem umožňuje uživateli zadat libovolný počet slov k detekování. Musíme tudíž zobrazit složitější výsledek závislý na celkovém počtu kriterií a na počtu splněných kriterií KWS. Detekce



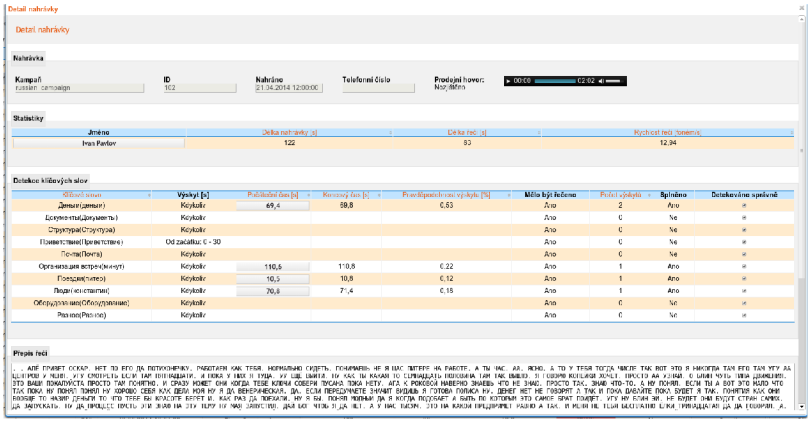
Obrázek 6.6: Vizualizace výstupů řečových technologií pomocí tabulky (AjaxFallbackDefaultDataTable)

klíčových slov je nezbytnou součástí hodnocení hovoru a její výsledek by měl být znázorněn přehledně. Můžeme doplnit informaci o výsledku detekce grafickým způsobem, a to tak, že na základě úspěšnosti splnění kriterií bude mít pole tabulky s výsledkem KWS pozadí různého odstínu červené barvy.

Ve druhém sloupci zprava je umístěno tlačítko „Detail“, které vede uživatele na stránku s detaily nahrávky, kde supervizor může zkoumat nahrávku podrobněji.

Kvalitativní analýza

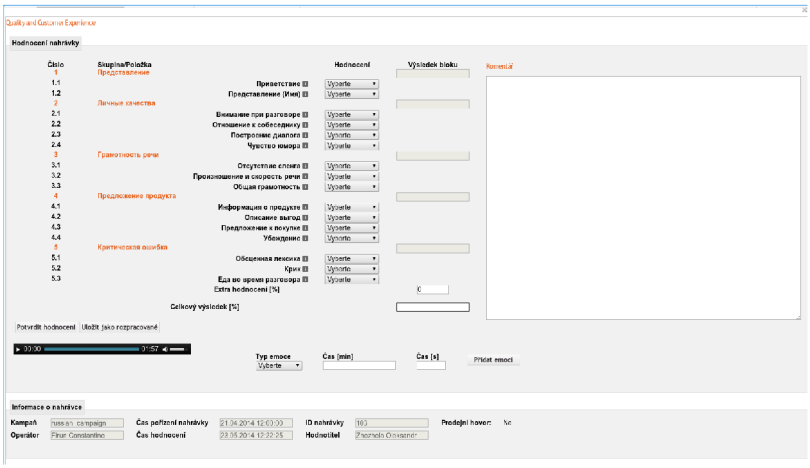
Pro účely kvalitativní analýzy řešení používá SPAS stránky „Detail nahrávky“ a „Hodnocení nahrávky“. Obě dvě stránky obsahují jednoduchý audio přehrávač a základní informace o nahrávce (název, jméno operátora, datum pořízení apod.)



Obrázek 6.7: Stránka Detail nahrávky

Stránka „Detail nahrávky“ v horní části obsahuje údaje o řečových parametrech každého kanálu nahrávky. Ve střední části se nachází tabulka s detailními výsledky detekce klíčových slov. Ve třetím sloupci tabulky můžeme vidět tlačítka s údaji o času, kdy bylo hledané slovo řečeno. Toto tlačítko je synchronizováno s přehrávačem a při zmáčknutí se nahrávka začne přehrávat od uvedeného času. Výsledek splnění kritéria je uveden v předposledním sloupci tabulky. Stejně jako u tabulky na obr. 6.6 by se jednotlivá pole mohla graficky zvýrazňovat podle skutečného výsledku.

V dolní části stránky nalezneme sekci s textovým přepisem nahrávky. Jedná se o statické textové pole. Znázorněná nahrávka má délku kolem dvou minut. U nahrávky s délkou deseti minut bude pole přepisu obsahovat velký úsek nepřehledného textu a orientace v něm bude značně obtížná. Vzhled a funkcionalitu této sekce by bylo potřeba kompletně předělat, anebo nahradit alternativním řešením.



Obrázek 6.8: Stránka Hodnocení nahrávky

Na stránce „Hodnocení nahrávky“ má supervizor možnost poslechnout si nahrávku a vyplnit předdefinovaný formulář. Pro náslech je zde základní audio přehrávač, který nemá žádné funkce navíc. Supervizorovi zde nejsou k dispozici žádné pomocné informace o nahrávce a musí ji poslouchat „naslepo“. Přehled o výsledcích detekce KWS, skoky do řeči a jiné vizualizační prostředky popsané v kapitole 4 by mohly supervizorovi ulehčit proces hodnocení.

Kapitola 7

Implementace

V této kapitole popíšeme proces implementace navržených vylepšení a rozšíření vizualizace platformy SPAS. V první části kapitoly rozebereme implementaci rozšíření standardních komponent frameworku Wicket. V druhé části popíšeme implementaci takových externích prostředků jako JavaScript knihovna a jejich následnou integraci do systému SPAS.

7.1 Dynamické tabulky

Popis základních typů a vlastností tabulek byl uveden v předchozí kapitole. Ze všech druhů implementace tabulek, které má k dispozici Wicket, řešení SPAS nejvíce používá realizaci pomocí tříd `DataTable` a `AjaxFallbackDefaultDataTable`. Dané implementace obsahují:

- navigační panel se stránkováním, informaci o počtu záznamů a aktuální stránce
- seřazení záznamů podle jednotlivých sloupců
- stylizování sudých a lichých řádků
- nastavení CSS stylů pro jednotlivé sloupce
- filtrování obsahu

Uvedeme jako příklad použití standardní tabulky pro znázornění seznamu kontaktů pomocí objektu třídy `AjaxFallbackDefaultDataTable`. Nejdřív musíme nadefinovat seznam sloupců, které bude obsahovat tabulka. Pro každý sloupec určíme komponentu, která bude definovat vzhled, chování a obsah jednotlivého pole.

```
List<IColumn<Contact>> columns = new ArrayList<IColumn<Contact>>();

columns.add(new AbstractColumn<Contact>(new Model<String>("Actions")){
        public void populateItem(Item<ICellPopulator<Contact>>  cellItem, String
            componentId,IModel<Contact> model){
                cellItem.add(new ActionPanel(componentId, model));
        }
});

columns.add(new PropertyColumn<Contact>(new Model<String>("ID"), "id"));
columns.add(new PropertyColumn<Contact>(new Model<String>("First␣Name"), "firstName"
    , "firstName"));
\\ ...
\\ ostatní sloupce
\\ ...
add(new AjaxFallbackDefaultDataTable<Contact>("table", columns,
            new SortableContactDataProvider(), 8));
}
```

V příkladu uvedeném výše jsou použity dva typy sloupců `AbstractColumn` a `PropertyColumn`. První varianta nemá ani žádný předem definovaný vzhled ani funkcionalitu. Musíme specifikovat jak má vypadat, co má obsahovat a jak má data získat. V tomto místě musíme se zmínit také o další důležité části rámce Wicket – o modelech.

Při vytvoření jakékoliv komponenty musíme určit její zdroj dat. Pro takové účely jsou ve Wicket používány objekty implementující rozhraní `IModel`. Pomocí modelů se zajišťuje návaznost komponent na konkrétní modelované objekty. Při vykreslování komponenty jsou konkrétní hodnoty extrahovány z modelu a při odesílání uživatelského vstupu (např. z formuláře) se hodnoty do modelu načítají. Jak již bylo uvedeno, Wicket realizuje návrhový vzor MVC (*Model-View-Controller*). V rámci vzoru MVC jsou komponenty Wicket zodpovědné za vrstvy View (pohled) a Controller(řadič). Komponenty se umí vykreslit a postarat o zpracování uživatelských akcí. Poslední vrstvu MVC zajišťují `IModel` objekty. Použití správné implementace modelu se promítá do spotřeby paměti, stručnosti kódu a výsledné podoby zobrazovaného objektu. V uvedeném příkladu tabulky se modely použijí pro promítání správných dat do jednotlivých polí.

Konstruktory sloupců jsou generické metody (`new AbstractColumn<Contact> (...)`), jimž musíme určit, který objekt budou modelovat. Jako parametr konstruktoru přidáme model řetězce, který bude uveden jako hlavička sloupce – *„Actions“*. Je-li požadováno řazení záznamů tabulky podle daného sloupce, pak musíme přidat další parametr řetězec s názvem atributu objektu Contact, podle kterého se pole sloupců budou seřazovat. Jelikož používáme realizaci sloupce, která nemá předdefinovaný vzhled a obsah, musíme přetížit metodu `populateItem()`, zodpovědnou za vykreslování pole. Daná metoda obdrží jako parametr objekt `Item` odpovídající poli sloupce, řetězec `componentId`, který se následně automaticky promítne do definice vzhledu v HTML, a model konkrétního objektu Contact. Zapouzdřením do objektu `cellItem` přidáme do každého pole daného sloupce jednoduchou komponentu `ActionPanel`, která umožní vybrat konkrétní objekt Contact přímo z tabulky. Realizace komponenty `ActionPanel` je uvedená v příloze B. Získáme tak kompletní inicializaci sloupce tabulky, který umožní vybírat konkrétní záznam stisknutím speciální zapouzdřené komponenty.

```
new AbstractColumn<Contact>(new Model<String>("Actions")){
        public void populateItem(Item<ICellPopulator<Contact>>  cellItem, String
            componentId,IModel<Contact> model){
                cellItem.add(new ActionPanel(componentId, model));
        }
};
```

Pro jednodušší účely se používají realizace sloupců, umožňující standardní znázornění obsahu jako jednoduchého textu. Například při použití `PropertyColumn` stačí pouze zadat jako parametr jméno atributu objektu Contact, který musí být znázorněn. Standardní implementace `PropertyColumn` vytvoří textovou komponentu `Label`, do které uloží hodnotu vybraného atributu. Můžeme ovšem také rozšiřovat a upravovat standardní chování `PropertyColumn` přetížením metody `populateItem()` jako v příkladu výše.

```
columns.add(new PropertyColumn<Contact>(new Model<String>("First␣Name"), "firstName"
    , "firstName"));
```

Nadefinovaný seznam sloupců následně předáme jako parametr při vytvoření instance třídy `AjaxFallbackDefaultDataTable`. Kromě toho v konstruktoru určíme také id komponenty (*„table“*), objekt realizující rozhraní `IDataProvider` a zajišťující naplnění tabulky daty a uvedeme rovněž počet záznamů na stránku. Inicializovanou instanci tabulky zapouzdříme do kontejneru funkcí `add()`

```
add(new AjaxFallbackDefaultDataTable<Contact>("table", columns, new
    SortableContactDataProvider(), 8));
}
```

Showing 1 to 8 of 48 << < 1 2 3 4 5 6 > >>

Actions	ID	First Name	Last Name	Home Phone	Cell Phone
select	5	Abby	Brown	445-555-8801	616-555-5125
select	22	Abby	Gomez	462-555-7007	588-555-8676
select	35	Abby	Davis	624-555-6252	700-555-4000
select	38	Abner	Davis	201-555-2646	324-555-5708
select	17	Brianna	Davis	405-555-1262	300-555-1064
select	28	Brianna	Graham	520-555-2281	264-555-3672
select	39	Christopher	Davis	202-555-8304	358-555-1726
select	4	Debra	Cruz	521-555-1784	350-555-7868

Obrázek 7.1: Příklad AjaxFallbackDefaultDataTable

Výsledná tabulka bude mít následující vzhled. Na obrázku je obvyklý vzhled standardních komponent, které nabízí Wicket. Rozšíříme standardní vzhled a chování komponent Wicket tak, abychom splnili stanovené požadavky na vizualizaci.

7.1.1 Zvýraznění kritického obsahu

Tento úkol je jednodušší, neboť zasahuje pouze do obsahu a vzhledu jednotlivého pole. V popsaném příkladu byl uveden jednoduchý způsob ovládání obsahu pole tabulky.

Tabulka používaná pro porovnávání nahrávek má následující strukturu. Každý řádek tabulky obsahuje údaje o jednotlivém kanálu nahrávky. V případě používání mono nahrávek bude každé nahrávce odpovídat jeden řádek tabulky, v případě použití stereo nahrávek – dva řádky (jeden bude obsahovat údaje zpracování kanálu operátora, druhý se bude týkat zákazníka). Pro každý kanál jsou uvedeny základní informace o kanálu a nahrávce, výsledky časové analýzy dialogu a rozpoznávání klíčových slov. Zaměříme se na sloupec obsahující výsledek KWS. Sloupec má typ `PropertyColumn` a přetěžuje metodu `populateItem()`. V závislosti na výsledku KWS dostává zapouzdřené textové pole buď model řetězce „Ok“, nebo model řetězce „Wrong“. Wicket nabízí lokalizaci pomocí lokalizačních souborů, které obsahují jména lokalizovatelných proměnných a skutečnou lokalizaci. V kódu pak zbývá jen inicializovat správný model se jménem lokalizovatelné proměnné jako parametrem.

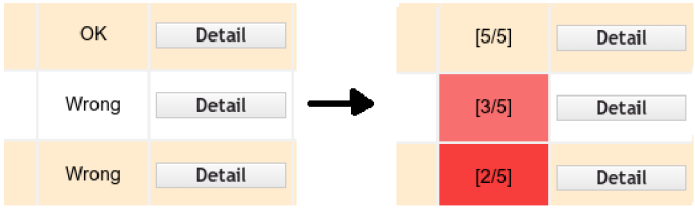
Změníme jednoduché podání informací o KWS ve formátu „Ok/Wrong“ na formát „[n/k]“, kde n je počet splněných kriterií a k je celkový počet kriterií. Vezmeme objekt třídy Channel, který je Java modelem kanálu a obsahuje kompletní informace o kanálu. Tento objekt obdržíme jako objekt modelu konkrétního řádku. Přepíšeme přetíženou metodu `populateItem()` a zapouzdříme textovou komponentu `Label` s požadovaným textem.

```
...
Channel channel = rowModel.getObject();
List<KwsResult> kwsResults = channel.getKwsResult();
if (!kwsResults.isEmpty()) {    //check if there are some KWS results

 cellItem.add(new Label(componentId, "[" + channel.getCorrectKwsCriteriaCount().
     intValue() + "/" + channel.getTotalKwsCriteriaCount().intValue() + "]"));

}
...
```

Zvýrazníme pole podle počtu splněných kriterií. Vypočteme poměr splnění kriterií a v případě 100% neúspěšnosti bude mít pole červené pozadí a naopak při 100% úspěšnosti se pozadí měnit nebude. Určíme také méně výrazné odstíny červené barvy pro případy

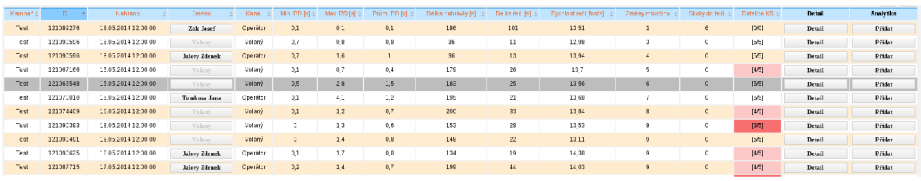
25%, 50% a 75% neúspěšnosti. Změnu stylu elementu provedeme pomocí instancí třídy `AttributeModifier`. Přidání takového objektu do komponenty umožní dynamicky změnit atribut HTML tagu odpovídající dané komponentě. Jako parametry uvedeme jméno atributu (v našem případě to je styl) a novou hodnotu. Úplný kód definice sloupce je v příloze B.

```
BigDecimal ratio = channelTmp.getKwsResultsRatio();
if (ratio != null) {

if (ratio.compareTo(BigDecimal.valueOf(50)) == 1) {
        if (ratio.compareTo(BigDecimal.valueOf(75)) == -1) {
                cellItem.add(new AttributeModifier(AttributeName.STYLE,
                    AttributeValue.BACKGROUND_COLOR + Color.RED_50));
        } else {
                cellItem.add(new AttributeModifier(AttributeName.STYLE,
                    AttributeValue.BACKGROUND_COLOR + Color.RED_25));
        }
} else {
        if (ratio.compareTo(BigDecimal.valueOf(25)) == -1) {
                cellItem.add(new AttributeModifier(AttributeName.STYLE,
                    AttributeValue.BACKGROUND_COLOR + Color.RED));
        } else {
                cellItem.add(new AttributeModifier(AttributeName.STYLE,
                    AttributeValue.BACKGROUND_COLOR + Color.RED_75));
        }
}}
```



Obrázek 7.2: Výsledek provedené úpravy

V předposledním sloupci se nachází tlačítko „Detail“ které otevírá modální okno s detailním popisem nahrávky. Po uzavření tohoto okna se uživatel vrací k výchozí tabulce. Jako další ulehčení pro uživatele změníme navigace a to tak, že zvýrazníme poslední navštívenou položku. Docílíme toho stejným způsobem pomocí změny stylu a uchovávání aktuálního řádku.



Obrázek 7.3: Výsledek provedené úpravy

7.1.2 Dynamické zobrazování sloupců

Jak již bylo zmíněno při rozboru požadavků supervizorů a v návrhu vizualizace musíme umožnit uživateli vybírat, které sloupce tabulky skutečně potřebuje vidět. V současné době má hlavička tabulky následující vzhled.

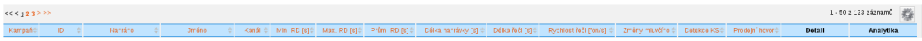
Obrázek 7.4: Standardní hlavička tabulky Wicket

Hlavní myšlenkou implementace požadované funkcionality je vytvoření vlastní rozšířené implementace třídy `DataTable`, která rozšíří svoji hlavičku, respektive její navigační panel o tlačítko. Daný ovládací prvek bude schopen otevírat modální okno obsahující nastavení viditelnosti jednotlivých sloupců tabulky. Dané tlačítko nesmí překážet uživateli při běžném ovládání tabulky a umístění v hlavičce tomuto požadavku vyhovuje. Nastavení uživatele se pak uloží do databáze a bude platné pro každé jeho další přihlášení do aplikace.

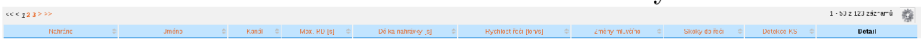
Typická komponenta Wicket `DataTable` je kontejnerem, který se skládá z menších částí. `DataTable` obsahuje standardní horní hlavičku, navigační panel a panel zaštiťující korektní zobrazování tabulky, která neobsahuje žádná data. Wicket umožňuje programátorovi rozhodovat o umístění jednotlivého panelu, např. hlavička může být ve spodní části tabulky apod. Implementujeme vlastní nástrojovou lištu a následně ji zapouzdříme do implementace tabulky. Do lišty zapouzdříme i navigační panel tabulky. Ušetříme tím prostor a sjednotíme vzhled a umístění panelů.

Vlastní implementaci lišty `CustomToolbar` bude rozšiřovat standardní rozhraní lišty Wicket `AbstractToolbar`. Pro vytvoření a přidání navigačního panelu nejdříve přetížíme a následně použijeme v konstruktoru lišty dvě standardní metody `newPagingNavigator()` a `newNavigatorLabel()`. Vytvoříme komponentu tlačítko pro modální okno s vlastnostmi sloupců a také ho přidáme do konstruktoru nové lišty. Pro dané tlačítko musíme nadefinovat funkci, která bude vykreslovat modální okno při stisknutí. Uvedeme jako privátní atribut instance třídy `ModalWindow`, který bude podřízený tlačítku. Celkový kód implementace se nachází v příloze B.

Při navrhování vlastních komponent musíme přemýšlet o jejich znovupoužití. V dané situaci musíme navrhnout rozhraní komponenty tak, abychom ji mohli použít v libovolném místě aplikace. Pro příklad této lišty to znamená, že musíme být schopni ji použít pro libovolnou tabulku. Každá tabulka má odlišnou sadu sloupců a mohou existovat i kritické sloupce, které nesmíme skrývat. Kromě toho má modální okno takové parametry jako šířka, výška, akce při zavíraní a další, které záleží na stránce, kde bude dané modální okno použito. Za takových podmínek nemůžeme implementovat samotný objekt modálního okna a musíme delegovat danou zodpovědnost řetězcem abstraktních metod tak, aby při inicializací tabulky daného typu musel uživatel komponenty abstraktní metodu vytvoření okna implementovat.



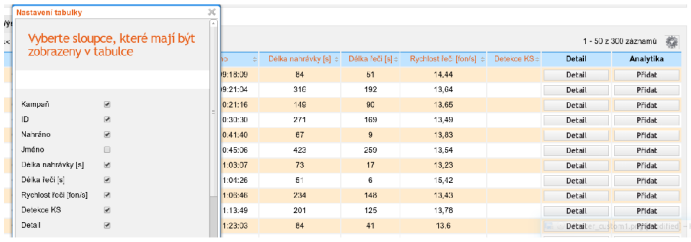
Obrázek 7.5: Rozšířená hlavička tabulky Wicket



Obrázek 7.6: Hlavička tabulky se skrytými sloupci

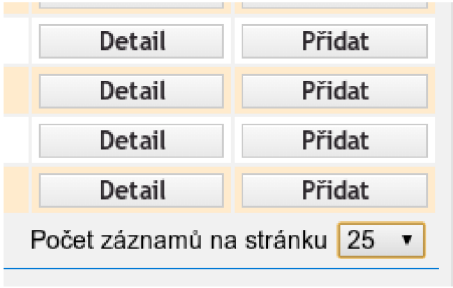
Pro hlavní tabulku znázorňující zpracované nahrávky bylo implementováno jednoduchou realizaci modálního okna, které obsahuje seznam sloupců spojených s komponentou-přepínačem `CheckBox`.

Další statický atribut tabulky představuje počet záznamů na stránku. Při práci s velkým počtem záznamů se uživatel bude muset často pohybovat mezi stránkami tabulky. Implementací možnosti výběru počtu záznamů na stránku můžeme uživateli tento proces



Obrázek 7.7: Implementace modálního okna s vlastnostmi viditelnosti sloupců

zjednodušit. Rozšíříme vlastní implementaci tabulky o další lištu, která bude obsahovat navigační komponentu jako instanci vypadajícího menu `DropDownBox` se seznamem variant počtu záznamů. Danou lištu umístíme ve spodní části tabulky. Při výběru nové položky ve vypadávajícím menu bude vzhled tabulky obnoven prostřednictvím AJAX.



Obrázek 7.8: Spodní lišta tabulky pro výběr počtu záznamů na stránce

7.2 Audio přehrávač

Ačkoliv jsou tabulky přehledným zdrojem vizualizace a rozšíření jejich funkcionality má pozitivní vliv na rychlost práce a pohodlí supervizora, hlavním vizualizačním prostředkem implementovaným v rámci této práce je audio přehrávač s rozšířenými možnostmi ovládání procesu náslechu.

7.2.1 Použité technologie

Součásti webových aplikací mohou běžet buď na straně klienta, nebo na straně serveru. V klientské části jsou spouštěny moduly, které nepotřebují vykonávat žádnou zákeřnou byznys logiku nebo např. obracet se pro data z databáze. Skripty jsou spouštěny a vykonávány přímo v prostředí klienta-prohlížeče. Po načtení obsahu stránky ze serveru mohou skripty z klientské části dynamicky měnit prezentace stránky a ovládat základní uživatelské akce. Funkcionalita audio přehrávače zapadá kompletně do daného paradigmatu. Pro vývoj modulů aplikace, které se spouštějí na straně klienta, se nejčastěji používá jazyk JavaScript. Daný jazyk dokáže dynamicky ovládat objektový model DOM, zpracovávat uživatelský vstup, pracovat s CSS styly apod.

Při programování webových aplikací se klade velký důraz na korektní fungování aplikace v různých prohlížečích. Tento problém je známý a někdy je velmi obtížné zajistit správné chování aplikace ve všech existujících web-browserech. Platforma SPAS doporučuje k použití dvě poslední verze jednoho z prohlížečů Google Chrome, Mozilla Firefox a Microsoft Internet

Explorer. V těchto prohlížečích je zajištěno správné chování aplikace. Při realizaci audio přehrávače musí být toto pravidlo dodrženo.

HTML5

HTML5 je pátá revize značkovacího jazyku HTML (*HyperText Markup Language*). Jedním z hlavních problémů předchozích verzí jazyků HTML a XHTML, které se v současné době používají na webu, bylo to, že tyto jazyky nejsou ničím jiným než souhrnem metod a rys definovaných v různých specifikacích. Důsledkem je spousta syntaktických chyb ve webových stránkách. HTML5 je snahou sjednotit rysy jazyků a syntaxe. Další velkou motivací vzniku této verze byla snaha o posun jazyku HTML dále ve směru podpory multimediálního obsahu. V HTML5 přibyla spousta nových možností prezentace a práce s multimediálním obsahem:

- nové elementy <audio> a <video> umožňující přehrávání a práci s audio a video záznamy přímo v prostředí webového prohlížeče
- element <canvas> pro přímé kreslení v prohlížeči, integrovaná podpora škálovatelné vektorové grafiky (SVG) a práce s matematickými formulemi
- nové elementy pro sémantickou prezentaci dokumentů
- drag-and-drop funkcionalita, offlinové webové aplikace, správa historie a další

W3C sice plánuje převést HTML5 do stavu „doporučeno k použití“ až v posledním čtvrtletí roku 2014, ale mnoho nových vlastností jazyku je již podporováno většinou webových prohlížečů a už se běžně používá při vývoji webových aplikací.

CSS

V současné době je málokterá webová stránka psána bez pomoci kaskádových stylů (*Cascading Style Sheets, CSS*). CSS je jazyk popisující způsob zobrazení elementů a stránek napsaných v jazycích HTML, XHTML a XML. CSS umožňuje oddělit vzhled dokumentu od jeho struktury a obsahu, zpřehledňuje text a orientování v dokumentu. Přiřazení elementu HTML definovaného stylu se provádí pomocí selektorů. Selektory mohou vybírat elementy určitého typu (např. pouze elementy s tagem H2), nebo prvky s určitým zapouzdřením, či elementy s definovanými hodnotami atributu(*id*, *class* a další).

Výběr typů selektorů ovlivní následní integrací přehrávače do platformy SPAS. Jelikož SPAS obsahuje vlastní pravidla pro styly elementů musíme navrhnout způsob stylizování implementované komponenty tak, aby její vzhled nepodléhal vnějším popisovacím pravidlům. Harry Roberts v publikaci *„Smashing book 4: New perspectives on Web design“* [3] od jednoho z nejpopulárnějších internetových zdrojů o webovém designu Smashing Magazine.[1] uvádí rozsáhlý přehled současných trendů a budoucnosti CSS. Daný přehled se věnuje zejména principu objektově orientovaného přístupu k architektuře CSS pro velké a škálovatelné aplikace, který spočívá v rozdělení monolitních bloků CSS na menší pravidla, jejich zapouzdření a znovupoužití. Pro potřeby implementace modulu přehrávače je důležitá část článku, která popisuje selektory. Z této části vyplývá, že pro účely definování specifických stylů pro takové portovatelné elementy jako audio přehrávač je nejvhodnějším řešením použití selektoru klasů. Selektory klasů jsou přímo navázány na konkrétní elementy a zachovávají přehlednou architekturu.

[1] http://www.smashingmagazine.com/

JavaScript a JQuery

JavaScript je programovacím jazykem Webu.[5] JavaScript je nenáročný, interpretovaný skriptovací jazyk s dynamickým přiřazováním typu, který má vlastnosti jak objektově orientovaných jazyků, tak rysy jazyků funkcionálních. Standardem pro JavaScript je ECMAScript, který tvoří jádro jazyka. Aktuálně podporována verze ECMAScriptu je 5.1 a šestá verze je ve fázi vývoje. Kromě ECMAScript JavaScript obsahuje sadu Web rozhraní. Jedním z takových rozhraní je *Document Objeect Model (DOM)*. DOM je jazykově nezávislá konvence pro prezentaci a interakci s HTML, XHTML a XML prvky. Realizace DOM v JavaScript umožňuje interaktivní ovládaní HTML stránky. JavaScript nesmí být spojován s jazykem Java.

HTML uchovává obsah a definuje formátování dokumentu. CSS popisuje jeho vzhled. JavaScript se používá pro podporu dynamického ovládání obsahu dokumentu a tvorbu prezentačních grafických efektů. V dnešní době existuje velké množství JavaScript knihoven pro řešení různých úloh webového programování. Významnou knihovnou pro pohodlné a rychlé ovládaní HTML obsahu a použití dynamických efektů je JQuery. Daná knihovna spolu s knihovnou hotových JavaScript komponent JQuery UI se již používají ve SPAS. Kvůli vnitřní politice vývojového týmu SPAS použití externích knihoven ve SPAS je nežádoucí.

7.2.2 Architektura

Kvůli vlastnostem chování lze audio přehrávač charakterizovat jako samostatný modul na webové stránce. Modul je integritní a samostatná část robustní aplikace. Realizace funkcionality pomocí modulů zachovává dobré rozdělení aplikace, její přehlednost a architekturu. Přidávání a použití modulu musí být co nejjednodušší. Modul musí být inicializován s potřebnými parametry a poté pracovat samostatně a nezávisle na webové aplikaci. Splněním těchto podmínek docílíme zachování principů znovupoužití a zjednodušíme si zapouzdření modulu do aplikace přes rámec Wicket.

Pro implementaci přehrávače použijeme návrhový vzor Modul. Při použití v JavaScript daný navrhovací vzor emuluje koncepce klasů z objektově orientovaného programování tak, aby realizovaný objekt obsahoval jak privátní metody a atributy, tak veřejné rozhraní. [9] Realizace uschování implementace je umožněno díky vlastnostem funkcí JavaScript. Funkce, která je definována vevnitř jiné funkce, je skrytá, pokud ji uživatel nezařadí do veřejných metod.

Implementace vzoru Modul z části kopíruje volání IIFE (*Immediately-Invoked Function Expression*)[2] funkcí, které využívají speciality syntaxe JavaScript tak, aby definovaná funkce byla spuštěna okamžitě. Vzor Model se liší tím, že vrací objekt místo funkcí. Uvedeme příklad jednoduché realizace vzoru Modul s jednou privátní proměnnou a dvěma veřejnými metodami.

```
var testModule = (function () {
  var counter = 0;
  return {
   incrementCounter: function () {
      return counter++;
    },
    resetCounter: function () {
      console.log( "counter value prior to reset: " + counter );
      counter = 0;
    }
  };
})();
```

[2] http://benalman.com/news/2010/11/immediately-invoked-function-expression/

Modul přehrávače bude založen na stejném principu chování a řízení přístupu. Pojmenujeme ho *SpasPlayer*. Na rozdíl od uvedeného příkladu nebude SpasPlayer vytvářet nový objekt. Wicket má přesně definovanou a řízenou hierarchii elementu stránky, a proto nemůžeme libovolně dosadit nový element na jakékoliv místo. Vytvoříme místo toho speciální element, který SpasPlayer nahradí po své inicializaci. Podrobněji rozebereme tento proces při integraci JavaScript knihovny do platformy SPAS.

Pří implementaci budeme potřebovat definovat takové pomocné objekty jako například výchozí parametry přehrávače a objekt s pomocnými funkcemi. Definování dalších objektů do globálního kontextu je velmi nežádoucí technikou. Takový postup může potenciálně vést ke kolizi s jinými knihovnami a mít další negativní důsledky. Podobné problémy se řeší pomocí prostorů jmen (*namespaces*). Pro modul SpasPlayer použijeme prostor jmen `spspl`. Do tohoto prostoru zapouzdříme všechny pomocné objekty.

```
\\define namespace
var spspl = spspl || {};
\\předáme jQuery funkcionálitu přímo do spspl
if (typeof jQuery != 'undefined') {
        spspl.$ = jQuery;
}
\\ IIFE style function call
(function ($) {

        \\default player settings.
        spspl.SpasPlayerDefaultOptions = {
                ...
        };

        \\utility methods for clear architecture
        spspl.SpasPlayerUtils = {
                ...
        };
        \\player object
        spspl.SpasPlayer = function(node, options) {

                var audioElement = node;
                ... \\implementace přehrávače
                return audioElement;
        };
})(spspl.$);

\\window.spspl = spspl;
window.SpasPlayer = spspl.SpasPlayer; \\pass SpasPlayer to outer namespace
```

Při takové implementaci modulu je inicializace objektu zcela jednoduchá. Konstruktor přehrávače SpasPlayer vyžaduje dva parametry. První parametr je prvek HTML, do kterého bude přehrávač injektován. Druhý parametr je objekt JavaScript obsahující skutečné parametry pro přehrávání, ukázku kritických míst, znázornění waveformu a dalších dat.

```
... //definice parametrů sourcesArray, operatorObject, eventsArray
var player = new SpasPlayer($('#audio'),
        {width:1200,
        height:250,
        sources: sourcesArray,
        waveformUrl:"img/waveform.png",
        operator:operatorObject,
        segments:segmentsArray,
        events:eventsArray,
        imageUrl:''}
);
```

7.2.3 Ovládaní procesů přehrávaní

V popisu HTML5 jsme se zmínili o nativní podpoře multimediálního obsahu pomocí elementů <audio> a <video>. Použití elementu <audio> umožňuje přehrávání audio záznamů přímo v prostředí webového prohlížeče. Pro správnou inicializaci je potřeba uvést cesty k audio záznamům, definovat viditelnost standardních ovládacích prvků a další volitelné parametry. Daný element lze vytvořit přímo v JavaScript pomocí konstruktoru Audio().

```
var audio = new Audio();

if (audio.canPlayType("audio/mpeg")) {
        audio.src = "test.mp3";
} else if (audio.canPlayType("audio/wav")) {
        audio.src = "test.wav";
}
audio.play();
```

Jelikož některé prohlížeče nepodporují přehrávání určitých formátů audio záznamů, musíme nejdříve ověřit možnost přehrávání daného zdroje. Například Internet Explorer nepodporuje přehrávání formátu WAV[3].

Browser	Operating system	Formats supported by different web browsers						
		Ogg Vorbis	WAV PCM	MP3	AAC	WebM Vorbis	Ogg Opus	WebM Opus
Google Chrome	All supported	9	Yes	Yes	Yes	Yes	25 (since v31 in Windows)	Yes
Internet Explorer	Windows	No	No	9	9	No	No	No
Mozilla Firefox	All supported	3.5	3.5	Windows (21.0), Linux (24.0, needs a gstreamer codec), OS X (26.0)	Windows (21.0) and Linux (24.0, needs a gstreamer codec) only	4.0	15.0	28.0[3]

Obrázek 7.9: Podpora přehrávaní různých formátu audio v prohlížečích

Obvyklým řešením daného problému je definovat několik zdrojů v různých formátech, což vyhovuje řešení SPAS, který pro svoje vnitřní účely převádí nahrávky do formátů WAV a MP3. Během posledních několika měsíců byla v knihovně BSAPI implementována podpora zpracování nahrávek ve formátu OPUS. Tento formát je vhodný jak pro zpracování nahrávek, tak pro jejich uchovávání a přehrávání.

Po úspěšné inicializaci audio objektu ho můžeme ovládat pomocí definovaných funkcí `play() a pause()`, změna místa přehrávání se provádí nastavením atributu `currentTime` na požadovaný čas v sekundách. Audio objekt také generuje důležité události, které můžeme odchytávat. Například událost `loadedmetadata` je generována po načtení audio zdroje. Poté, co byla událost generována, můžeme využívat takové vlastnosti audio elementu jako například délku záznamu `duration`. Právě po inicializaci audio elementu přehrávač SpasPlayer inicializuje svoje hlavní ovládací prvky – tlačítka „Přehrát“(*Play*) a „Zastavit“ (*Stop*). Po spouštění přehrávaní Tlačítko „Přehrát“se bude dynamicky měnit na tlačítko „Pozastavit“ (*Pause*) a naopak. Daný rys je implementován pomocí dynamické výměny CSS stylů a funkcí definujících chování elementu při kliknutí. Sjednocení této funkcionality v jednom elementu je logické, neboť se tyto funkce přehrávače vzájemně vylučují a také se tímto krokem ušetří prostor.

[3]WAV (*Waveform Audio File Format*) je standardem pro ukládání audio dat do PC, vyvinutým Microsoftem ve spolupráci s IBM. Do dnešní doby však Internet Exporter nepodporuje tento formát.

Obrázek 7.10: Ovládací prvky

Obrázek 7.11: Rozšíření informace o nahrávce

Na obrázku 7.11 lze vidět rozšíření panelu přehrávače a nové elementy: název nahrávky, aktuální čas přehrávání a celková délka nahrávky. Ve volném prostoru pod danými elementy umístíme grafickou reprezentaci signálu nahrávky – waveform. Všechny komponenty popsané zde a dále jsou vytvářeny dynamicky a následně přidávány do HTML elementu obdrženého jako parametr při inicializaci přehrávače.

7.2.4 Zobrazování waveformu

Waveform je grafické znázornění signálu. Abychom mohli signál znázornit, musíme být schopni signál zpracovat. JavaScript a HTML5 prezentují rozhraní Web Audio API[4], které umožňuje práci se signálem na mnoha úrovních. Zpracování, syntéza a zachycování signálu, přehrávání jednotlivých kanálů, filtry, komprese signálu a další možnosti tvoří seznam rysů Web Audio API. Při použití prvku <canvas> pro dynamické vykreslování grafiky lze poměrně jednoduše vykreslit znázornění signálu přímo v prohlížeči. Bohužel je zatím dané API podporováno pouze v browseru Google Chrome, a proto ho nemůžeme použít v řešení SPAS.

Podobné úlohy jsou řešeny dvěma způsoby. První způsob je založen na použití externí JavaScript knihovny, která realizuje požadovanou funkčnost. Podobné knihovny ovšem většinou buď jenom předávají nahrávky na zpracování na vzdáleném serveru (např. *Sound Cloud*), nebo je zpracovávají samostatně. Připojení dalších externích knihoven do platformy SPAS je nežádoucí, kvůli vnitřní politice vývojového týmu. Navíc zpracování nahrávky knihovnou trvá určitý čas a zdržuje proces inicializace přehrávače na relativně dlouhou dobu. Předávání nahrávky na externí server není zcela možné kvůli ochraně citlivých dat zákazníků.

Druhý způsob řešení daného úkolu spočívá ve vytvoření waveformu pomocí některé serverové služby. Existují různé realizace podobných služeb a většina z nich je implementovaná v jazyku C. Daný postup také nevyhovuje architektuře aplikace a vytváří další softwarovou závislost, což pro velký komerční projekt není zcela vyhovující.

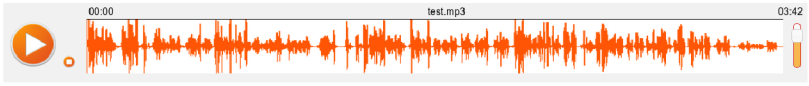
I když použití externího řešení není umožněno, postup tvoření waveformu na straně serveru může být správným. Během komunikace s vývojovým týmem knihovny BSAPI-wrapper bylo zjištěno, že daná funkcionalita je implementována v jazyce C v základní knihovně BSAPI. Knihovna BSAPI-wrapper pro jazyk Java byla rozšířena o danou funkcionalitu. Pro použití této funkce BSAPI-wrapper má být implementována speciální třída callback pro vykreslování základních geometrických forem, vytvoření a ukládání obrázků. Zdrojový kód implementace dané třídy je uveden v příloze B.

[4] http://www.w3.org/TR/webaudio/

Rozšíříme aplikační kontext SPAS o příslušné definice pro korektní použití nových instancí tříd knihovny BSAPI-wrapper v rámci vzoru IoC.

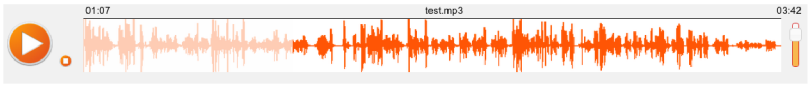
```
<!-- objekt řídící proces vykreslování pomocí nativních metod BSAPI --!>
<bean id="waveFormPanel" class="com.phonexia.bsapiwrapper.WaveformPanel" destroy-
    method="release"/>

<!-- objekt zodpovědný za načtení nahrávky a předání do waveFormPanel --!>
<bean id="waveFormData" class="com.phonexia.bsapiwrapper.WaveformData" destroy-
    method="release"/>

<!-- callback zodpovědný za vykreslování grafického obrázku pomocí metod Java --!>
<bean id="waveFormEditorCallbackImpl" class="com.phonexia.speechanalyticsserver.
    bsapi.WaveFormEditorCallbackImpl" destroy-method="release">
    <constructor-arg name="width"   value="900"></constructor-arg>
    <constructor-arg name="height"  value="100"></constructor-arg>
</bean>
```

Tato funkcionalita BSAPI-wrapper ovšem přináší jedno omezení, které ovlivní způsob jejího použití. BSAPI-wrapper pracuje jenom s nahrávkami ve formátu WAV. Pokud byla nahrávka přeformátována do jiného formátu například kvůli ekonomii místa na disku, použít BSAPI-wrapper pro vykreslování waveform nebude možné. Při popisu základní funkcionality SPAS bylo uvedeno, že každá nahrávka se zpracovává technologií TAE. Tato technologie také zpracovává jen WAV soubory, a proto vytvoření waveformu můžeme provést v této fázi. Dojde k vytvoření dost velkého počtu obrázků, většina z nich ale nebude použita, což se může zdát zbytečným krokem, avšak toto řešení splňuje všechny specifikace. Nároky na uchovávání obrázků jsou akceptovatelné: pro 11,7 GB audio dat potřebuje kompletní sada obrázků waveformu cca 40 MB prostoru na disku. Náročnost technologie vytvoření waveformu je zanedbatelná ve srovnání s technologií zpracování řeči. Kromě toho budeme ihned mít obrázek připravený k použití v libovolný okamžik. Zapouzdříme obrázek waveformu do přehrávače.



Obrázek 7.12: Realizace waveformu

Rozšíříme také funkcionalitu ovládání přehrávání o uživatelské akce v sekci waveformu. Kliknutím na určité místo obrázku se spustí přehrávání nahrávky od odpovídajícího času. Vhodnou vizualizační technikou je znázornění průběhu přehrávání na obrázku waveformu. Přidáním speciálního prvku-kontejneru se specifickým stylem a šířkou měnící se v závislosti na aktuálním čase přehrávaní docílíme požadovaného efektu.



Obrázek 7.13: Realizace waveformu s překreslením

Ovládání doplníme o funkci změny hlasitosti. Jako ovládací prvek použijeme komponentu knihovny jQuery UI, která se již používá ve SPAS.

7.2.5 Zobrazování kritických míst

Z pohledu supervizora jsou kritická místa v hovoru úseky nahrávky, kde bylo detekováno jedno z následujících:

- kritéria KWS[5]

  špatné - nesmělo být řečeno, ale bylo detekováno

  správné - mělo být řečeno a zaznělo v hovoru

- skoky do řeči
- dlouhá reakční doba operátora.

Nezávisle na tom, že daná kritická místa jsou různého typu, všechny tři případy mají společné vlastnosti. Jsou to úseky nahrávky a mají počáteční a koncový čas. V tomto kroku hraje čas důležitou roli. Zobrazování pouze aktuálního času přehrávání a délky nahrávky zřejmě nebude stačit. Podle požadavků a návrhu uživatelského rozhraní by měl přehrávač obsahovat také časovou osu. BSAPI-wrapper umožňuje vykreslování rovněž tohoto vizualizačního prvku, ale v tomto případě existuje i jednodušší řešení, které implementujeme.

Obrázek 7.14: Vizualizace časové osy pomocí BSAPI-wrapper

Již několikrát byl zmíněn prvek HTML5 <canvas> určený pro práci s grafikou a kresleni. Časová osa není ničím jiným než posloupností čárek s textovými značkami času. Vykreslování podobného obsahu je klasický případ použití <canvas>. Mírnou obtíží tohoto úkolu může být výpočet správného měřítka. Interval 10 vteřin pro textové označení času bude vhodným pro nahrávku, která má délku dvě minuty. Pro nahrávku o délce patnácti minut bude takové značení nepoužitelné. Vhodný interval vypočteme v závislosti na délce nahrávky a šířce volného prostoru určeného pro zobrazení časové osy.

```
var drawTimeline = function(){
if (ratio == null && audio.duration !== NaN) {
        ratio = parseInt(window.getComputedStyle(waveForm, null).width) / audio.
            duration;
}
canvas.width=parseInt(window.getComputedStyle(canvasContainer, null).width)*0.97;
canvas.height=parseInt(window.getComputedStyle(canvasContainer, null).height);
if (canvas.getContext){
  var interval;
  var ctx = canvas.getContext('2d');
  interval = countInterval(ratio);
  ctx.beginPath();
  for (var i = 0; i < audio.duration; i++) {
    if(i % interval === 0){
          ctx.moveTo(i*ratio,0);
          ctx.lineTo(i*ratio,10);
        }else{
          ctx.moveTo(i*ratio,5);
          ctx.lineTo(i*ratio,10);
        }
        if(i % interval === 0 && i!= 0){
                ctx.font = "bold 10px sans-serif";
                ctx.fillText(spspl.SpasPlayerUtils.formatTime(i), i*ratio-14, 20);
        }

  };
  ctx.stroke();
}
}
```

Pro nahrávku o délce 160 vteřin bude mít generována časová osa následující vzhled

[5]Zde uvádíme jenom případy, které lze grafický znázornit. Případ, kdy hledané slovo mělo být řečeno, ale nezaznělo, v nahrávce graficky znázornit nelze.

Obrázek 7.15: Vizualizace časové osy pomocí Canvas

Implementovaná časová osa má stejnou šířku jako obrázek waveform a to znamená, že proces přehrávání a časová osa jsou zcela synchronizovány. Stejně synchronizovaně se musejí umístit i znázornění kritických míst. Nejdřív však musíme vybrat vhodné značky pro každý druh kritického místa.

- kriteriá KWS

  špatné -

  správné -

- skoky do řečí -
- dlouhá reakční doba operátora -

Zde platí stejná pravidla jako v případě dynamického zobrazování sloupců tabulky. Každý supervizor může pracovat a zaměřovat se na odlišnou sadu dat. Proto musíme umožnit uživateli libovolně zobrazovat nebo skrývat elementy určitého typu. Ovládací tlačítka umístíme ve zvláštní sekci vedle sekce s ovládáním přehrávání.

Další nový rys HTML5 výrazně rozšiřující možnosti ovládání prvky HTML jsou vlastní atributy, které může definovat programátor. I když podobné rozšíření zcela nezapadá do koncepce HTML5 pro sjednocení různorodých specifikací HTML, přináší to určité výhody. Podobné vlastní uživatelské atributy musejí mít prefix `data-`. Takové atributy jen uchovávají informaci a nemají žádný vliv na vzhled prvku. Každému elementu zobrazující kritické místo přidáme podobné atributy s počátečním a koncovým časem. Definujeme pak funkci, která bude zpracovávat událost kliknutí na obrázek jednotlivého kritického místa, a to tak, že přečte definované vlastní atributy a zvýrazní příslušnou oblast waveformu. Navíc se daná oblast automaticky přehraje. Při navedení kurzory myši na jednotlivou značku kritického místa označující kritéria KWS se v oblasti nad kurzorem objeví panel s detekovaným slovem nebo frází a s přesností detekce.

Obrázek 7.16: Vizualizace kritických míst

7.2.6 Zobrazování přepisu nahrávky

Další podstatnou součástí vizualizace je přepis řeči do textové podoby. V prostředí webového vývoje pro znázornění přepisu řeči do textu synchronizovaně s přehráváním multimediálního obsahu se používá pojem *titulky*. Existují celé JavaScript knihovny, které nabízejí implementaci titulků (*Popcorn.js*). V rámci této diplomové práce implementujeme vlastní realizaci tohoto vizualizačního prostředku.

Stejně jako kritická místa mají segmenty přepisu obdržené při inicializaci přehrávače počáteční a koncový čas. Hlavní myšlenkou implementace je zapouzdření segmentu do tabulky se dvěma sloupci: počáteční čas segmentu řeči a text segmentu. Každý řádek tabulky

tak bude představovat jednotlivý segment. Jako `id` atribut každého řádku uvedeme počáteční čas odpovídajícího segmentu. Toto dává možnost rychle vyhledávat aktuální řádek-segment podle času pomocí triviálních selektorů a filtrování. Segment můžeme považovat za aktuální, pokud je jeho počáteční čas menší než hodnota aktuálního času přehrávání nahrávky a zároveň tato hodnota je menší než počáteční čas následujícího segmentu (pokud nějaký následuje). Aktuální řádek-segment bude uchováván ve speciální proměnné. Při přehrávání nahrávky v určitých časových intervalech se provede kontrola, zda je uchovaný segment stále aktuální. Pokud se najde nový segment, který splňuje podmínky aktuálnosti, uchovávaný segment bude obnoven. Při této akci pomocí úpravy stylu tabulka změní svoji polohu vertikálním posunem tak, aby aktuální segment byl umístěn jako první viditelný řádek. Skrytí přetékání tabulky za mez bloku kontejneru se vyřeší pomocí CSS. Prohledávání skryté části tabulky lze umožnit scrolováním. Aktuální řádek bude také zvýrazněn zvláštním stylem.

Takovým způsobem je implementována jednosměrná synchronizace přehrávání-aktualizace segmentu. Musíme také zajistit synchronizaci opačnou. Při výběru jakéhokoliv segmentu kliknutím přehrávač začne přehrávat nahrávku od počátečního času vybraného segmentu.

Obrázek 7.17: Vizualizace titulků

7.2.7 Verifikace operátora

Posledním rozšířením audio přehrávače o výstupy z řečových technologií implementovaným v rámci této diplomové práce bude panel verifikace operátora. Platforma SPAS má funkcionalitu umožňující verifikovat mluvící osobu na základě dříve získaných otisků hlasu. V návrhu vizualizace již bylo zmíněno, že znázornění této informace ulehčí práci supervizorovi při poslechu. Implementujeme jednoduchý panel, kde zobrazíme jméno operátora a obrázek vystihující výsledek verifikace. Tento panel se umístí do volného prostoru nad ovládacími elementy.

Obrázek 7.18: Vizualizace verifikace operátora

7.2.8 Zapouzdření do Wicket kontejneru

Implementovaný modul SpasPlayer lze nyní vložit do webové stránky aplikace SPAS. Rámec Wicket spravuje hierarchii stránek aplikace takovým způsobem, že nelze vložit libovolný prvek do struktury bez nahlášení této akce rámci. Pro zapouzdření externích modulů, knihoven a komponent musíme nejdříve vytvořit komponentu Wicket, která bude vložená

do stromové hierarchie Wicket komponent ve fázi kompilace a do které se následovně injektuje požadované rozšíření. Připojení potřebných knihoven a souborů stylu lze implementovat přetížením metody `renderHead()`, kterou dědí všechny komponenty ve Wicket. V této metodě nadefinujeme proměnné obsahující cesty k souborům knihoven, jež pak budou automaticky přidány do hlavičky webové stránky. V tomto místě také přidáme inicializační script pro vytvoření a zapouzdření, který vytvoří a vrátí metoda `getInicializeJs()`, která je uvedená v příloze B.

```
@Override
public void renderHead(IHeaderResponse response) {

  response.render(CssHeaderItem.forUrl(CssConstants.SPAS_PLAYER));
  ResourceReference jQueryResourceReference = getApplication().
      getJavaScriptLibrarySettings().getJQueryReference();
  response.render(JavaScriptHeaderItem.forReference(jQueryResourceReference));
  response.render(JavaScriptHeaderItem.forUrl(JsConstants.SPAS_PLAYER));
  response.render(JavaScriptHeaderItem.forScript(getInitializeJs(), "
      SpasPlayerInitJS"));
  response.render(JavaScriptHeaderItem.forUrl(JsConstants.JQUERY_UI));
  response.render(JavaScriptHeaderItem.forScript(getInitializeJs(), "JQueryUI"));
}
```

Po provedených úpravách stránka „Detail nahrávky“ má následující vzhled.

Obrázek 7.19: Vizualizace verifikace operátora

Kapitola 8

Diskuze s koncovými uživateli

Po implementaci vizualizace a rozšíření funkcionality standardních Wicket komponent byly výsledky prezentovány koncovým uživatelům – supervizorům jednoho z největších kontaktních center České republiky. Daná skupina uživatelů již byla seznámená s řešením SPAS a většina měla i praktické zkušenosti s použitím platformy.

8.1 Zpětná vazba

Navržené a implementované řešení získalo kladnou odezvu uživatelů. Změna funkčnosti a vzhledu tabulek ulehčila a zpřehlednila práci supervizoru při kvantitativní analýze nahrávek. Změna prezentace výsledků KWS přivedla k tomu, že supervizoři začali častěji využívat KWS pro analýzu záznamů.

Ještě větší zájem získal audio přehrávač SpasPlayer. Ve zpětné vazbě bylo uvedeno, že tento nástroj uspokojuje hlavní potřeby supervizorů při zkoumání a hodnocení hovorů. Kromě jiného byly rovnou navrženy další funkce přehrávače, které základní implementace neobsahovala. Bylo požádáno o umožnění změny rychlosti přehrávání audiozáznamů. Základní implementace byla následně rozšířená o požadované funkce.

8.2 Implementace dodatečné funkcionality

Audio objekt má atribut `playbackRate`, který je zodpovědný za rychlost přehrávání nahrávky. Přidáním ovládacích prvků, které dynamicky řídí změnu daného atributu, lze docílit požadovaného efektu.

Obrázek 8.1: Výsledná podoba SpasPlayer

Kapitola 9

Závěr

Hlavním cílem mé diplomové práce byla implementace uživatelského rozhraní a vizualizaci dat získaných pomocí řečových technologií pro zefektivnění vnitřních procesů kontaktních center.

Byly prozkoumány metody a technologie, jejichž pomocí lze data z řeči extrahovat. Za použití knihovny BSAPI a jejího rozhraní pro jazyk Java byl stanoven konkrétní rozsah dat, který můžeme získat. Výstupy řečových technologií tvoří velké množství dat, které lze používat jak pro vizualizace, tak pro další analýzu. Na základě analýzy použití řečových technologií v produktech různých firem byly definovány základní scénáře praktického použití daných technologií a jejich výsledků. Dalším zaměřením této diplomové práce bylo použití řečových technologií v kontaktních centrech zejména v úseku kontroly kvality práce operátorů.

V rámci práce probíhala komunikace s pracovníky kontaktních center. Účast v kalibraci supervizorů velkého kontaktního centra zjednodušila zkoumání jeho vnitřních procesů. Popis daných procesů je uveden v kapitole 4. Ve stejné kapitole jsou zároveň definovány požadavky pracovníků úseku kontroly kvality kontaktního centra na vizualizaci dat získaných pomocí řečových technologií. Byly stanoveny úrovně vizualizace dat v závislosti na úrovni zkoumání řečových záznamů. První úroveň analýzy hovorů zahrnuje porovnávání velkého množství záznamů mezi sebou a následující nižší úroveň obsahuje analýzu jednotlivého hovoru. Pro tyto úrovně analýzy byly navrženy, formulovány a odůvodněny požadavky na uživatelské rozhraní pro vizualizaci dat získaných pomocí zpracování záznamů hovorů řečovými technologiemi.

Požadované uživatelské rozhraní, implementované v rámci této diplomové práce je integrované do řešení Speech Analytics Server (SPAS), které je zaměřené na analýzu dat získaných z řeči a které se již používá v kontaktních centrech. Architektura a základní funkcionalita SPAS je uvedena v kapitole 6. Řešení SPAS již obsahovalo širokou sadu vizualizačních prostředků, které byly v rámci práce vylepšeny a rozšířeny. Kromě toho byl implementován samostatný modul audio přehrávače SpasPlayer s rozšířenými možnostmi přehrávání audio záznamů a vizualizace výstupu řečových technologií.

Výsledky daného rozšíření a zlepšení byly prezentovány pracovníkům kontaktního centra a byla obdržena kladná zpětná vazba uvedená v kapitole 8. Zároveň byla supervizory navržena další funkcionalita pro audio přehrávač, jíž základní implementace neobsahovala. Tato funkcionalita byla úspěšně dodána a v současné době se kompletní řešení úspěšně používá v praxi.

Literatura

[1] Call centre [online]. http://en.wikipedia.org/wiki/Call_centre.

[2] COPC CSP Standard The Performance Management System for Customer Contact Operations. 2012-11.

[3] *The Smashing Book 4 - New Perspectives on Web Design.* Smashing magazine, 2013.

[4] Brno Speech Application Interface Documentation [online]. http://www.phonexia.cz/docs/bsapi/index.html, 2013-05-17.

[5] Flanagan, D.: *JavaScript: The Definitive Guide, 6th Edition.* O'Reilly Media, 2011, iSBN 978-1-4493-0212-2.

[6] Hannemann, M.: Weight Finite State Transducers in Automatic Speech recognition. http://www.fit.vutbr.cz/study/courses/ZRE/public/pred/10_wfst_lvcsr/zre_lecture_asr_wfst.pdf, 2013-10-04.

[7] Hubeika, V.; Burget, L.; Černocký, J.; aj.: Maximum Likelihood and Maximum Mutual Information Training in Gender and Age Recognition System. http://www.fit.vutbr.cz/research/groups/speech/publi/2007/hubeika_tsd_2007.pdf, 2007.

[8] ITU: Measuring the Information society [online]. http://www.itu.int/en/ITU-D/Statistics/Documents/publications/mis2013/MIS2013_without_Annex_4.pdf, 2013.

[9] Osmani, A.: *Learning JavaScript Design Patterns.* O'Reilly Media, 2013.

[10] Plchot, O.; Matějka, P.: Language identification [online]. http://www.fit.vutbr.cz/study/courses/ZRE/public/pred/11_sid_lid/lid.ppt, 2012.

[11] Psutka, J.; Matoušek, J.; Muller, L.; aj.: *Mluvíme s počítačem česky.* Vyd. 1. Praha: Academia„ 2006, iSBN 8020013091.

[12] Volf, T.: Krize nášemu byznysu spíš pomáhá, říká ředitel největšího call centra v Česku [online]. http://byznys.ihned.cz/?p=020000_d&article[id]=60049510, 2013-06-17.

[13] Černocký, J.: Zpracování řečových signálů - studijní opora [online]. http://www.fit.vutbr.cz/study/courses/ZRE/public/ opora/zre_opora.pdf, 2006-12-06.

Příloha A

Obsah CD

1. Text práce ve formátu PDF spolu se zdrojovými soubory pro LaTeX
2. Zdrojové kódy implementovaných Wicket komponent
3. Zdrojový kód knihovny SpasPlayer
4. Webová stránka s ukázkou použití SpasPlayer
5. Testové nahrávky pro přehrávání pomocí SpasPlayer

Příloha B

Zdrojové kódy

B.1 Implementace komponenty ActionPanel uvedené v příkladu kódu na stráně 35

```
private class ActionPanel extends Panel{
public ActionPanel(String id, IModel<Contact> model)
{
  super(id, model);
  add(new Link("select"){
    @Override
    public void onClick(){
        setSelected((Contact)ActionPanel.this.getDefaultModelObject());
    }});
  SubmitLink removeLink = new SubmitLink("remove", form){
    @Override
    public void onSubmit(){
      Contact contact = (Contact)ActionPanel.this.getDefaultModelObject();
      info("Removed␣contact␣" + contact);
      DatabaseLocator.getDatabase().delete(contact);
      }
  };
  removeLink.setDefaultFormProcessing(false);
  add(removeLink);
}}
```

B.2 Implementace sloupce obsahující výsledek KWS s dynamickým přiřazováním stylu.

```
new PropertyColumn<Channel, String>(new ResourceModel("kwsResult"), "kwsResult", "
    kwsResult." + getLocaleColumn()) {

@Override
public void populateItem(Item<ICellPopulator<ChannelTmp>> cellItem, String
    componentId, IModel<ChannelTmp> rowModel) {
        Channel channel = rowModel.getObject();

        List<KwsResult> kwsResults = channel.getKwsResult();
        if (!kwsResults.isEmpty()) {
        BigDecimal ratio = channel.getKwsResultsRatio();

        if (ratio != null) {
        if (ratio.compareTo(BigDecimal.valueOf(50)) == 1) {
          if (ratio.compareTo(BigDecimal.valueOf(75)) == -1) {
                cellItem.add(new AttributeModifier(AttributeName.STYLE,
                    AttributeValue.BACKGROUND_COLOR + Color.RED_50));
          } else {
```

```
                cellItem.add(new AttributeModifier(AttributeName.STYLE,
                    AttributeValue.BACKGROUND_COLOR + Color.RED_25));
          }
        } else {
          if (ratio.compareTo(BigDecimal.valueOf(25)) == -1) {
                cellItem.add(new AttributeModifier(AttributeName.STYLE,
                    AttributeValue.BACKGROUND_COLOR + Color.RED));
          } else {
                cellItem.add(new AttributeModifier(AttributeName.STYLE,
                    AttributeValue.BACKGROUND_COLOR + Color.RED_75));
          }
        }}
        cellItem.add(new Label(componentId, "[" + channel.getCorrectKwsCriteriaCount
            ().intValue() + "/" + channel.getTotalKwsCriteriaCount().intValue() + "]
            "));
        } else {
                cellItem.add(new Label(componentId, ""));
}}}
```

B.3 Implementace třídy CustomDataTableWithColumnOptions rozšiřující základní funkcionalitu Wicket tabulek s použitím vlastních lišt.

```
public abstract class CustomDataTableWithColumnOptions<T, S> extends DataTable<T, S>
     {

private CustomToolbar customToolbar;

public CustomDataTableWithColumnOptions(String id, List<? extends IColumn<T, S>>
    columns, ISortableDataProvider<T, S> dataProvider, int rowsPerPage) {

        super(id, columns, dataProvider, rowsPerPage);
        customToolbar = new CustomToolbar(this) {
                @Override
                public ModalWindow createOptionsModalWindow() {
                        return createOptionModalwindow();
                }
                @Override
                public AjaxButton createCopyBtn(String id) {
                        return implementCopyBtn();
                }
        };
        addTopToolbar(customToolbar);
        addTopToolbar(new HeadersToolbar<S>(this, dataProvider));
        addBottomToolbar(new NoRecordsToolbar(this));
        addBottomToolbar(new CustomBottomToolbar(this));

}


public abstract ModalWindow createOptionModalwindow();

public abstract AjaxButton implementCopyBtn();

@Override
protected Item<T> newRowItem(final String id, final int index, final IModel<T> model
    ) {
        return new OddEvenItem<T>(id, index, model);
}

public void showCopyModalWindow(AjaxRequestTarget target, String text) {
        customToolbar.showCopyModalWindow(target, text);
}
}
```

B.4 Implementace vlastní komponenty lišty pro použití v tabulce s dynamickým zobrazováním sloupce.

```
public abstract class CustomToolbar extends AbstractToolbar {

        private static final long serialVersionUID = 1L;
        private ModalWindow optionsModalWindow;
        private String text;
        private DataTable<?, ?> table;

        public CustomToolbar(final DataTable<?, ?> table) {
                super(table);
                this.table = table;

                WebMarkupContainer span = new WebMarkupContainer("span");
                add(span);
                optionsModalWindow = createOptionsModalWindow();
                add(optionsModalWindow);
                span.add(AttributeModifier.replace("colspan", new
                    AbstractReadOnlyModel<String>() {
                        @Override
                        public String getObject() {
                                return String.valueOf(table.getColumns().size());
                        }
                }));

                span.add(newPagingNavigator("navigator", table));
                span.add(newNavigatorLabel("navigatorLabel", table));
                span.add(newOptions("options"));
        }
        private AjaxButton newOptions(String id) {
                AjaxButton button = new AjaxButton(id) {

                        @Override
                        protected void onSubmit(AjaxRequestTarget target, Form<?>
                            form) {
                                optionsModalWindow.show(target);
                        }
                };
                return button;
        }

        public abstract ModalWindow createOptionsModalWindow();

        protected PagingNavigator newPagingNavigator(final String navigatorId, final
             DataTable<?, ?> table) {
                return new PagingNavigator(navigatorId, table);
        }

        protected WebComponent newNavigatorLabel(final String navigatorId, final
            DataTable<?, ?> table) {
                return new NavigatorLabel(navigatorId, table);
        }
```

B.5 Implementace třídy-callbacku pro vykreslování obrázku waveformu pomocí knihovny BSAPI wrapper.

```
public class WaveFormEditorCallbackImpl implements WaveformEditorCallback,
    InitializingBean {

        private Graphics2D g2;
        private BufferedImage image;
        private int width;
        private int height;
```

```
@Override
public void afterPropertiesSet() throws Exception {
        image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB
            );
        g2 = image.createGraphics();
        g2.setColor(new Color(255, 85, 5));
}
public WaveFormEditorCallbackImpl() {

}
public WaveFormEditorCallbackImpl(int width, int height) {
        this.setWidth(width);
        this.setHeight(height);

        image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB
            );
        g2 = image.createGraphics();
        g2.setColor(new Color(255, 85, 5));
}
@Override
public void drawBox(int arg0, int arg1, int arg2, int arg3, WEditorColorType
     arg4) {
        g2.drawRect(arg0, arg1, arg2 - 1, arg3 - 1);
}
@Override
public void drawLine(int arg0, int arg1, int arg2, int arg3,
    WEditorColorType arg4) {
        double x1 = arg0;
        double y1 = arg1;
        double x2 = arg2;
        double y2 = arg3;
        g2.draw(new Line2D.Double(x1, y1, x2, y2));
}
@Override
public void drawPixel(int arg0, int arg1, WEditorColorType arg2) {
        double x1 = arg0;
        double y1 = arg1;
        g2.draw(new Line2D.Double(x1, y1, x1, y1));

}
public BufferedImage getImage() {
        return image;
}
public void release() {
        image = new BufferedImage(getWidth(), getHeight(), BufferedImage.
            TYPE_INT_ARGB);
        g2 = image.createGraphics();
        g2.setColor(new Color(255, 85, 5, 150));
}
@Override
public void drawTextLabel(int arg0, int arg1, String arg2, WEditorColorType
    arg3) {
        g2.drawString(arg2, arg0, arg1);

}

public BufferedImage getNoDataImage() {
        g2.setColor(new Color(255, 85, 5));
        g2.drawString("No␣Waveform␣Data␣available", 50, 50);
        return image;
}
public int getWidth() {
        return width;
}
public void setWidth(int width) {
        this.width = width;
}
```

```
        public int getHeight() {
                return height;
        }
        public void setHeight(int height) {
                this.height = height;
        }
}
```

B.6 Metoda, která formuje JavaScript funkci inicializaci modulu SpasPlayer

```
private String getInitializeJs() {

        WebClientInfo clientInfo = WebSession.get().getClientInfo();
        ClientProperties clientProperties = clientInfo.getProperties();

        List<MediaSource> sources = getSources();
        String sourcesVar = "[";
        if (sources != null && sources.size() > 0) {
                for (MediaSource source : sources) {
                        sourcesVar += "{'url':'" + source.getSrc() + "', 'mimeType
                            ':'" + source.getType() + "'},";
                }
        }
        if (sourcesVar.endsWith(",")) {
                sourcesVar = sourcesVar.substring(0, sourcesVar.length() - 1);
        }
        sourcesVar += "]";
        if (getSegments() == null || getSegments().equals("")) {
                setSegments("[]");
        }
        if (this.getEvents() == null || this.getEvents().equals("")) {
                this.setEvents("[]");
        }

        String result = "var player;\n";
        result += "$(document).ready(function () {\n";
        result += "player = new SpasPlayer($('audio'),{";
        result += "width:" + width + ",";
        result += "height:" + height + ",";
        result += "sources:" + sourcesVar + ",";
        result += "operator:" + getOperator() + ",";
        result += "events:" + getEvents() + ",";
        result += "segments:" + getSegments() + ",";
        result += "waveformUrl:'" + waveformUrl + "',";
        result += "imageUrl:'" + imageUrl + "'});\n";
        result += "});\n";
    return result;
}
```