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

Bakalářská práce

Transkripční software pro trénování ASR systémů

Plzeň, 2012 Tomáš Kubový

Prohlášení

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

V Plzni dne ……. ……

Podpis

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu práce Ing. Kamilu Ekšteinovi, Ph.D. za praktické rady, výbornou spolupráci a výstižné poznámky při tvorbě.

Abstract

Transcription Software for ASR Systems Training

The main goal of this bachelor thesis is to implement the transcription software through which it would be easy to assign text to the selected part of an audio file in the WAV format. This process will not be automated but realised by people. Their work could be later used for training of automatic speech recognition systems. The application is written in Java programming language and uses Swing toolkit for its graphical user interface. The theoretical part of this text deals with competitive software accessible on the Internet, storage of sound in the computer and graphical visualization of sound. The second part of the thesis concerns the design and implementation of an application such as playing a sound in Java or saving data into the XML format.

Obsah

1 Úvod

Porozumění řeči mezi lidmi je běžný děj. Mluvíme s někým ve stejném jazyce a často ani nepřemýšlíme nad tím, jaké slovo má jaký význam. Jsou to pro nás věci naprosto automatické. Daleko zajímavějším případem je naučit počítače rozpoznávat a analyzovat slova řečená člověkem. Mohli bychom například místo mačkání kláves přímo diktovat naši požadovanou zprávu, zatímco by ji počítač vypisoval na obrazovku. Nakonec mu jen přikázat, aby připojil obrázek z dovolené a odeslal vše na vyslovenou adresu.

Úroveň informačních technologií jde dopředu nezastavitelným tempem a snaha o dokonalé ovládání přístrojů hlasem už není pouze představou. Právě k tomuto tématu přispívá i transkripční software, který neslouží k rozpoznávání zvuků, nýbrž k učení inteligentních systémů pro dokonalé porozumění lidem pomocí hlasu.

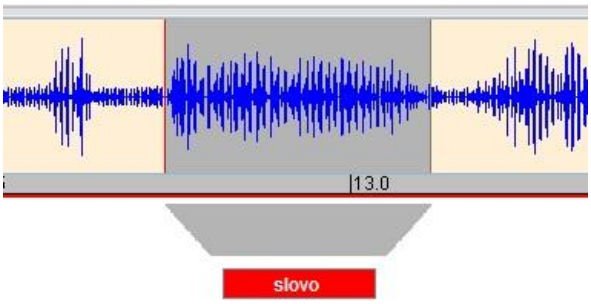
Cílem této bakalářské práce je tedy vytvořit software, pomocí kterého bude možné jednoduše přidat zvukovému úseku ve formátu WAV jeho textový přepis. Tento proces nebude nijak automatizovaný, transkripci budou provádět lidé. Pomocí jimi vytvořené práce proběhne trénování systémů pro rozpoznávání řeči.

Aplikace musí umět přehrávat zvuk a umožňovat obecné ovládání známé z běžných přehrávačů, jakým je například posouvání či pozastavování. Je zapotřebí vstupní zvuková data zobrazit na obrazovku (například vykreslením zvukové vlny), aby rozpoznání hranic jednotlivých slov bylo co nejpatrnější, a tudíž práce s programem jednoduchá a rychlá. Výstupem z aplikace bude textový soubor obsahující seznam přepsaných segmentů s časem začátku a konce ve zvukové stopě.

2 Teoretická část

2.1 Pojem transkripce

Jedná se o přepis zvukového signálu do textové podoby (viz Obrázek 1). V našem případě hovoříme o přiřazení textů k vybraným úsekům (nejčastěji slovům nebo větám) zvukové stopy a následné uložení do souboru.



Obrázek 1: Příklad transkripce

Abychom výsledek mohli využít, musí obsahovat i informaci o začátku a konci úseku, ke kterému byl text přiřazen. Z toho důvodu je nutné využít k přepisu speciálně navržený software, jelikož pouhé poslouchání zvukové stopy a následné vytváření textového výstupu spolu s ohraničujícími časovými údaji by bylo velmi nepřesné a zdlouhavé.

2.2 Veřejně dostupný software

Moje aplikace není novinkou v této oblasti. V současnosti existuje několik dostupných programů obsahujících základní funkce potřebné pro transkripci a export výsledných dat.

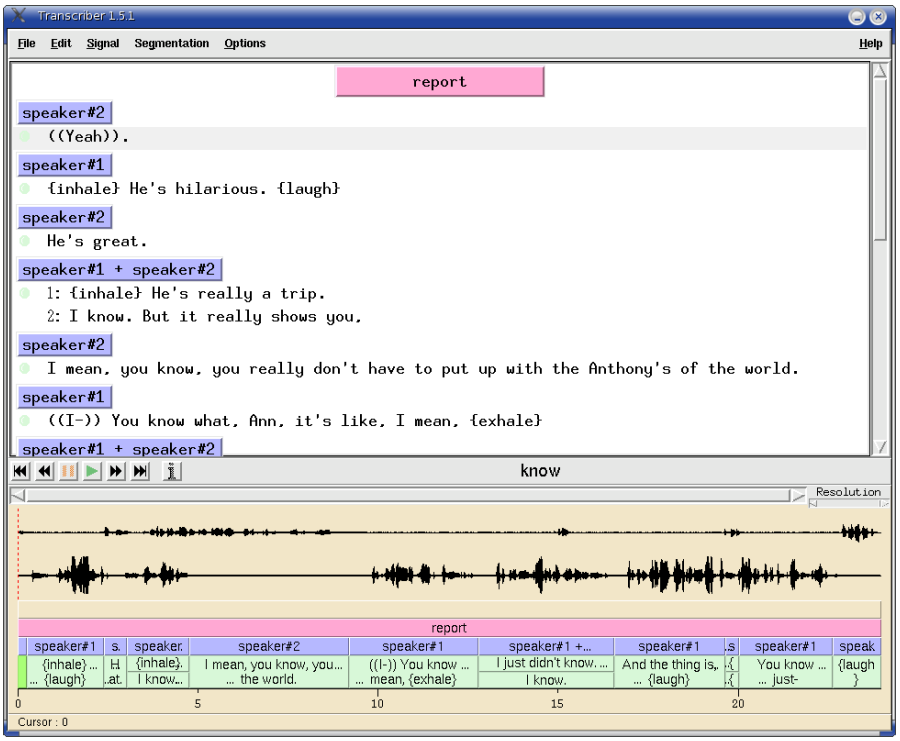
2.2.1 Transcriber 1.5.1

Jde o rozsáhlý open-source (otevřený software – software s volně dostupnými a využitelnými zdrojovými kódy) projekt (viz Obrázek 2) vytvářený a zdokonalovaný v letech 1998 až 2008 skupinou autorů. Původní distribuce mířily pouze na operační systém Linux, avšak přínosem nových verzí bylo i rozšíření na systémy Windows, Mac OS a další. Informace a software jsou dostupné na internetové stránce [1].

Konkrétně je Transcriber 1.5.1 určen pro anotace nahrávek zpravodajských vysílání a souborů pro vývoj automatických zpravodajských transkripčních systémů, avšak jeho možnosti mohou být využity i v jiných oblastech zabývajících se výzkumem řeči.

Aplikace umí pracovat s nejrůznějšími druhy zvukových formátů, mezi nejznámější a pravděpodobně nejpoužívanější patří podpora WAV, MP3 nebo OGG. Umožňuje jejich načtení a grafické zobrazení zvukové vlny. Zahrnuta by měla být i možnost

zpracovávat video formáty, avšak pouze pod operačním systémem Mac OS. Použitelnost nejsem schopen zhodnotit, jelikož nevlastním tento systém.



Obrázek 2: Transciber 1.5.1

Pro přepisování je potřeba si nejprve vytvořit tzv. speakery (osoby mluvící ve zvukové stopě), nastavit jejich dialekt, další specifické vlastnosti a následně jim přiřazovat segmenty s textem. Dále je možné vytvořit různé úseky rozdílných dialogů, například telefonát nebo cokoliv jiného, které budou obsahovat příslušné segmenty. Co ovšem tento program neumí, je rozdělení úseku na více menších. Tedy nelze nejdříve vytvořit souvětí, poté jej rozdělit na věty a ty následně na slova. Stručně řečeno, jednotlivé segmenty se nesmí překrývat, neboli stejný čas nesmí být obsažen ve dvou a více částech.

Po zpracování celé stopy můžete uložit svoji práci do souboru s příponou TRS. Jedná se o klasický XML formát (viz kapitola 2.6), ovšem s vlastní koncovkou, aby bylo možné soubor otevřít přímo tímto programem. Výstup obsahuje definované speakery a další speciální části vytvořené během transkripce spolu s jednotlivými přepsanými segmenty. Jelikož mezi úseky nemůže zůstat nepřiřazené místo, i když se v něm žádný potřebný zvukový signál nevyskytuje, obsahuje každý exportovaný segment pouze koncový čas. Počáteční je stejný jako koncový předchozího úseku. Dalšími možnostmi exportu je transformace přeloženého textu do formátu HTML nebo TXTS. Nemají

v sobě ovšem zapsánu informaci o čase výskytu a další nezbytnosti. Slouží tedy pouze jako textový výstup přepsaného zvuku.

Při vývoji mé aplikace jsem byl inspirován právě tímto programem. Pravděpodobně nebude tak kvalitní, protože Transcriber 1.5.1 byl zdokonalován a rozšiřován po několik let týmem programátorů. Věřím ale, že základní funkci bude splňovat podobným způsobem.

2.2.2 Další programy pro transkripci

Mezi další aplikace sloužící pro transkripci se dá zařadit například program *InqScribe,* který nabízí možnosti přiřazování textu k přehrávanému videu. Výhodou je i přeložení výstupu do standardního formátu titulků k filmu SRT. Je dostupný v několika verzích. Bezplatná verze má sice veškerou funkčnost, nelze ovšem ukládat. Tím se stává téměř nepoužitelnou a uživatele to nutí koupit si licenci na 4 – 6 týdnů nebo plnou verzi napořád.

Ještě bych mohl zmínit jednodušší verzi tohoto typu softwaru, jíž je *EasyTranscriber.* Nabízí pouze přehrání zvuku různou rychlostí a přiřazení textu.

2.3 Uložení zvuku v počítači

Jelikož jsou data v počítači uložena výhradně v digitální podobě (tedy posloupnosti jedniček a nul) a zvuk, který my můžeme slyšet, je mechanické vlnění s proměnlivou frekvencí, jež vyvolá sluchový vjem, je tedy nutné zavést metody, které převedou signál do posloupnosti číslic a naopak číslice do zvukového signálu.

2.3.1 PCM – Pulsně kódová modulace

Digitalizace je proces, který reprezentuje analogový zvukový signál jako proud jedniček a nul. Jak uvádí kniha [2], PCM metoda založená na principu vzorkování je nejběžnější používanou metodou pro digitalizaci. Po pravidelných intervalech převádí hodnotu amplitudy signálu na hodnotu číselnou. Analogový signál je poté reprezentován proudem čísel.

Nyquist-Shannonův teorém [7] říká, že vzorky (anglicky *samples*) musí být pořizovány s alespoň dvojnásobnou frekvencí, než je frekvence vzorkovaného analogového signálu. Naše uši slyší zvuk v rozmezí frekvencí od 20 Hz až po 20 000 Hz (20kHz), tudíž počet vzorků získaných v každé sekundě by měl být alespoň 40 000

pro zaznamenání zvuku s frekvencí 20 kHz. Zvuk na CD je zaznamenáván s frekvencí 44,1 kHz vzorků za sekundu, tedy o něco lepší než dvojnásobek frekvence, kterou můžeme slyšet. Ani to není naprosto ideální, je to pouze minimum.

Dalším parametrem PCM je velikost vzorku. Každý vzorek analogového signálu je charakterizován svojí amplitudou vyjadřující hodnotu elektrického napětí. Tato hodnota je v počítači uložena v podobě datového slova složeného z jedniček a nul. Čím delší slovo je, tím větší hodnoty může reprezentovat a díky tomu i větší dynamický rozsah zvuku zaznamenat. CD formát má velikost vzorku 16 bitů, která dostačuje k reprodukci zvuku s rozsahem asi 96 dB (decibelů). Mnoha lidem připadá zvuk z CD „studený“ s občasným pískáním ve vyšších frekvencích. Z toho důvodu byl vytvořen DVD-Audio formát, který podporuje frekvenci vzorkování 192 kHz s velikostí vzorku 24 bitů.

Je-li tedy zvuk na CD ve stereo formátu (2 kanály) a s velikostí vzorku 16 bitů, jedna sekunda záznamu zabere 176,4 kB (viz Rovnice 1).

$$44100 * 2 * 16 = 1411400\ b/s = 176\,400\ B/s = 176{,}4\ kB/s$$

Rovnice 1: Výpočet velikosti dat 1 sekundy záznamu na CD

2.4 Formát WAV

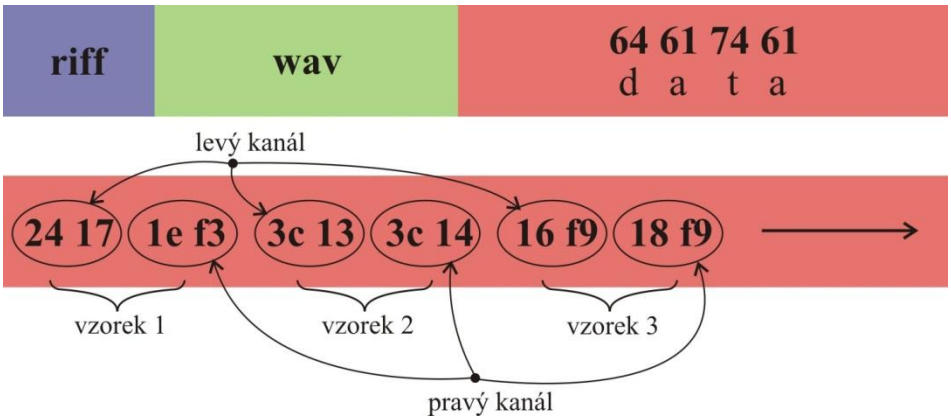
Formát WAV (zkratka pro *Waveform audio file format*) byl vytvořen úpravou obecnějšího formátu RIFF společností Microsoft a slouží pro ukládání multimediálních dat. Může obsahovat i komprimované formáty, například MP3, nejčastěji ale bezztrátový PCM. Každý jednotlivý soubor typu WAV obsahuje hlavičku (viz Obrázek 3) rozdělenou na dvě části (anglicky *chunk*). První obsahuje specifikace o souboru riff a druhá, větší, o formátu WAV. Informace jsou čerpány z internetové stránky [6].

Obrázek 3: Hlavička WAV souboru

Po prvních 36 bitech následuje libovolně dlouhá datová oblast. Po počáteční značce označující datovou část jsou zapsána zvuková data. Je-li záznam rozdělen do dvou a více kanálů, vzorky jednotlivých kanálů nejsou zapsány jeden za druhým, ale střídají se (viz Obrázek 4). To umožňuje proudové čtení a přehrávání všech kanálů najednou.

Například u stereo záznamu se na prvním místě nachází první vzorek levého kanálu, poté první pravého, následuje druhý vzorek opět levého kanálu a tak dále až do konce souboru.

Jelikož výsledkem pulsně kódové modulace je formát bezztrátový, je jeho zpracování velmi snadné a nenáročné na výpočet. Často se používá jako výchozí formát před další konverzí. Pro svoji jednoduchost je vhodný k přenosu zvuku mezi různými systémy.



Obrázek 4: Struktura wav souboru

2.5 Vizualizace zvuku

Pro snadné ovládání programu je velmi vhodné zvuk nějakým způsobem vizualizovat na obrazovku a vnímat jej očima, jimiž jsme schopni zaznamenat daleko více informací než sluchem.

Nejlepší způsob, jak zvuk zobrazit, je pomocí grafu se zvukovou vlnou (viz Obrázek 5). Jedná se o XY graf, kde horizontální osa označuje čas a vertikální amplitudu signálu v čase. Velikost amplitudy v sobě nese každý vzorek. Kdybychom je postupně všechny do grafu vykreslili, byla by výsledná vlna velmi podobná tvaru původního analogového signálu. A stejně tak by byla příliš roztažená a nepřehledná. My ovšem nepotřebujeme znát amplitudu v každém čase, kdy byl signál vzorkován, ale v nějakém rozumném podintervalu. Nezbývá než počet zobrazovaných bodů výrazně zredukovat. To můžeme vyřešit například tím, že vypočteme průměr intervalu a ten následně vykreslíme.

Pomocí takového grafu je orientace a nalezení začátku a konce slova o mnoho jednodušší a rychlejší.

Obrázek 5: Vizualizace zvukové vlny

2.6 XML jazyk

Aplikace musí umožňovat uložení již zpracovaných dat na disk a na druhou stranu opětovné načtení jednou vytvořených dat. Jeden z formátů, se kterým se tímto způsobem velmi dobře pracuje, je právě XML.

Jak se píše na internetové stránce [3], XML (neboli *eXtended Markup Language*) je takzvaný značkovací jazyk vhodný pro uchovávání dat v textové podobě. Na návrhu XML se začalo pracovat v druhé polovině devadesátých let s cílem vytvořit standard, který by lépe vyhovoval dnešním nárokům na zpracování informací než stávající obdobné formáty.

Při návrhu XML se autoři řídili následujícími požadavky:

- Musí být použitelný v rámci internetu,
- formát XML by měl podporovat širokou škálu aplikací,
- musí být kompatibilní s formátem SGML (jazyk, ze kterého XML vychází),
- musí být snadné vytvářet programy, které manipulují s dokumenty v XML,
- množství variant XML by mělo být minimální – nejlépe žádné a
- XML dokumenty by měly být čitelné a pochopitelné i pro člověka.

XML dokument (viz Obrázek 6) je tvořen z otevíracích a uzavíracích značek (tzv. elementů) a atributů obsahujících dodatečné informace (např. *<element atributId=“5“> text elementu </elem-ent>*). Každý otevírací element musí mít k sobě právě jeden uzavírací a mezi nimi mohou být zapsány informace nebo další dvojice elementů, které se nesmí

```
<?xml version="1.0" encoding="utf8" standalone="yes"?>
<transcription>
    <soundFile>speech.wav</soundFile>
    <speakers>
        <speaker id="0">
            <name>(default)</name>
            <info></info>
            <color b="120" g="80" r="255"/>
        </speaker>
    </speakers>
    <segments>
        &lt;segment idSpeaker="0" idSegm="0">
            <speakerName>(default)</speakerName>
            <time endTime="2.304" startTime="0.888"/>
            <text>yes</text>
        &lt;/segment&gt;
    </segments>
</transcription>
```

Obrázek 6: Ukázka XML souboru

navzájem překřižovat. Pomocí takové soustavy vzájemně zanořených elementů a atributů se snaží XML jazyk zachytit co nejpřesnější strukturu a provázanost uložených dat.

2.6.1 XML Schema

Jde o speciální typ jazyka, který specifikuje vlastnosti elementů a atributů XML dokumentu. XML schémata v jazyce XML Schema jsou opět XML dokumenty s pevně danou strukturou. Tento jazyk přesně určuje, jak má konkrétní XML soubor vypadat. Definuje, jak se mají jednotlivé elementy jmenovat, kde přesně mají být umístěny, jaké mají atributy s daným datovým typem, jak jsou do sebe zanořeny s jinými elementy a další specifikace, které musí dokument splňovat, aby byl validní podle tohoto schématu. Tyto informace a mnoho dalších pochází z knihy [4].

Soubory obsahující XML schéma (viz Obrázek 8) mají příponu `xsd`. Odkaz na ně může být obsažen v hlavičce XML souboru a využit pro validaci dokumentu (viz Obrázek 7).

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="FastButtons"
type="buttons"/>

<xs:complexType name="buttonStyle">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="id" type="xs:int"
        use="required"></xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="buttons">
        <xs:sequence>
   <xs:element name="Button"
type="buttonStyle"
     maxOccurs="unbounded" minOccurs="0"/>
  </xs:sequence>
   </xs:complexType>
</xs:schema>
```

Obrázek 8: Ukázka XSL schéma

```
<?xml version="1.0"?
                encoding="UTF-8"?>
<FastButtons
xsi:noNamespaceSchemaLocation=
                "fastButtons.xsd">

  <Button id="1">SaveBtn</Button>
  <Button id="2">DeleteBtn</Button>
  <Button id="5">ExitBtn</Button>

</FastButtons>
```

Obrázek 7: Ukázka validního XML dokumentu podle schématu

2.6.2 Parsování XML

Pro práci s XML dokumentem existuje několik technik pracujících rozdílným způsobem. Abychom nemuseli sami v programu procházet znak po znaku a rozlišovat, zdali se jedná o kořenový element, atribut či jestli je hodnota číselného typu nebo řetězce, je vhodné dokument zpracovat pomocí tzv. parseru.

Obecné vlastnosti parseru:

- Provádí nízko-úrovňovou analýzu XML,
- provádí kontrolu správné strukturovanosti,
- extrahuje názvy elementů a atributů a jejich hodnoty,
- zpracovává všechny další pomocné informace z XML souboru.

Jak uvádí kniha [4], parserů existuje mnoho. Mezi ty nejznámější a obecně použitelné ve většině programovacích jazyků patří SAX (*Simple API for XML*) a DOM (*Document Object Model*).

Parser SAX

Parser SAX pracuje na principu proudového čtení. Postupně čte XML dokument a pro každou ucelenou část (element, atribut atd.) vyvolá událost. Právě proto je také tento princip nazýván „událostmi řízené zpracování“. Jeho výhodou je tedy velká rychlost a nízká paměťová náročnost, na druhou stranu se ovšem nemůže při průchodu vracet zpět.

Parser DOM

Zcela odlišným způsobem pracuje parser DOM. Ten načte celý dokument do stromové struktury v paměti, kterou lze následně upravovat podle potřeby. Je tedy možné provést jakékoliv změny v datech a uložit je zpět do dokumentu, cenou ovšem je velmi velká spotřeba paměti a menší rychlost zpracování.

Parser JAXB

Java ovšem nabízí další parser, jímž je JAXB (*Java Architecture for XML Binding*). Jde o automatické mapování mezi XML dokumentem a odpovídajícími Java třídami přes XSD. Není tedy potřeba vytvářet podobné třídy v Javě, jelikož ty mohou vzniknout automatickým vygenerováním. Pokud tedy známe dopředu schéma XML dokumentu, je vhodné jednorázově před použitím vygenerovat třídy pomocí nástroje XJC (*XML to Java Compiler*) a ty poté použít v aplikaci. Výrazně ulehčuje naplnění tříd daty a následně práci s opětovným zapsáním dat zpět do XML dokumentu. Program XJC spolu s třídami vytvoří i jejich programátorskou dokumentaci vhodnou pro práci s nimi.

3 Realizace

Aplikace je napsána v programovacím jazyce Java a byla vytvářena ve vývojovém prostředí Eclipse. Jde o velmi mocný nástroj sloužící programátorům nejen v Javě, pro kterou je primárně určen, ale umožňuje doinstalování zásuvných modulů pro vývoj aplikací například v jazyce Python nebo C/C++, atd. Aplikace v Javě nepracuje přímo pod operačním systémem, ale na virtuálním stroji (přesněji *Java Virtual Machine*), který je nezbytnou součástí pro její spuštění a běh. Verze jazyka, ve kterém byl program psán, je *JavaSE-1.6*. Na nižší nebude pracovat, jelikož obsahuje části, které starší verze nepodporuje.

3.1 Struktura aplikace

Jazyk Java je takzvaný objektově orientovaný programovací jazyk. Jeho části jsou umístěny do tříd, které v sobě nesou potřebné atributy k dané části a metody s nimi pracující. Jednotlivé třídy spolu mohou komunikovat a využívat možností jiných, čímž vytváří přehlednou a pochopitelnou strukturu.

Projekt se skládá z celkem 30 veřejných tříd rozdělených do čtyř balíků (v Javě *package*).

- **general**

  Hlavní balík aplikace, obsahuje všechny třídy vytvářející její jádro. Většina z nich slouží jako komponenty zobrazované na obrazovku. Zpracovává události vytvořené uživatelem (například vybrání úseku k přehrání nebo stisknutí tlačítka, atd.) a využívá další balíky.

- **player**

  Obsahuje třídu pro komunikaci se zvukovou kartou a vytváří vlákno, které odesílá zvuková data k přehrání.

- **exportImport**

  Třídy v tomto balíku slouží pro ukládání vytvořené transkripce do xml souboru nebo naopak importování rozpracované části do aplikace. Další třídou je spustitelné vlákno, které na pozadí automaticky ukládá aktuálně přepsanou transkripci a poté zobrazuje uživateli zprávu o uložení stavu do souboru.

- **startConfig**

  Poslední balík uchovává informace o nastavení aplikace, ať už barev okna nebo automatického ukládání a třídu pro zobrazení dialogového okna s nastavením.

3.2 Hlavní části programu

Jelikož se nejedná o aplikaci postavenou na nějakém složitém matematickém jádře, není zde tedy tolik zajímavých algoritmů a konstrukcí, které by bylo třeba popisovat.

3.2.1 Načtení zvukové stopy

O načtení zvukového souboru a celkově o správu zvukových dat se stará třída `AudioData`. V ní jsou obsaženy tři důležité metody sloužící ke zjištění informací z hlavičky souboru, spočítání hodnot grafu zvukové vlny a předání požadované části grafu na vykreslení.

- **`loadAudio()`-** je volána vždy při otevírání nového souboru. Pomocí systémové třídy `AudioSystem`, která slouží jako vstupní bod ke zvukové kartě počítače, a pomocí zadaného souboru vytvoří vstupní proud dat. Kromě zvukových dat, jež můžeme postupně číst, má v sobě uloženy i informace získané z hlavičky wav souboru nezbytné pro vytvoření zvukové vlny nebo přehrávače. Jedná se například o počet kanálů, velikost jednoho vzorku, délku stopy a hlavně počet vzorků v jedné sekundě. Tento zvukový proud není ovšem používán pro přehrávání, samotný přehrávač si vytvoří stejným způsobem vlastní, ze kterého data čte.
- **`countSamples()`-** pokud předchozí metoda skončila bez problémů, přichází na řadu spočítání vzorků pro vykreslení. O to se stará právě tato funkce. Ze zjištěných vlastností o zvukové stopě postupně čte vstupní proud dat a spočítané vzorky ukládá do pole, které je později vykreslováno jako graf zvukové vlny. Podrobnosti v kapitole 3.2.2.
- **`getSamplesToDraw()`-** poslední důležitá metoda související s načítáním zvuku. Podle zadaných parametrů vytvoří pole o délce jako je šířka zobrazovaného grafu a naplní jej hodnotami, které budou vykresleny. Spolu s šířkou má metoda i parametry času začátku vlny a délky zobrazovaného času, podle nichž určí počáteční index a jeho inkrement v poli všech vzorků napočítaném v metodě `countSamples()`. Jde o to, že minimální zobrazovaná délka vlny na obrazovce je 2 sekundy, kde je pro každý bod spočítána hodnota. Počet pixelů v horizontální

linii je dán šířkou grafu, standardně 1000 bodů. Pokud je zobrazovaná délka vlny například 4 sekundy, každá druhá hodnota se přeskakuje, aby výsledná šířka vzorků k vykreslení byla opět 1000. Tedy inkrement indexu není 1, ale 2. Podobně se spočítá pro další délky vlny (4, 8, 16 a 32 sekund), které jsou záměrně mocninou dvou, aby vycházel inkrement indexu celočíselný.

3.2.2 Vykreslení zvukové vlny

Jak již bylo řečeno výše, vykreslení zvukové vlny zcela jistě patří mezi hlavní prvek této aplikace, který spojuje dohromady ostatní části. Na celý graf není však kladen pouze požadavek vykreslení. Zobrazit na 1000 bodech zvukovou stopu dlouhou například pět minut a nemít možnost jakékoliv další interakce by znamenalo, že v každém pixelu (vertikální úsečce), bude zakódováno skoro tři a půl sekundy. Během této chvíle může být vysloveno i deset slov, tudíž samotná jednoduchá vizualizace by neměla žádný význam.

Další nezbytností pro práci je možnost horizontálního posouvání v grafu, tedy posouvání po ose x. Zobrazována je pouze různě dlouhá výseč podle potřeby. Díky tomu je možné po přepsání jednoho úseku přeskočit na další. V jednu chvíli není potřeba vidět na obrazovce celou vlnu, ale pouze část, ve které budou dobře patrné přechody mezi jednotlivými slovy interpreta. Pro přesnější práci při transkripci je implementována i možnost zvětšování a zmenšování délky zobrazované části.

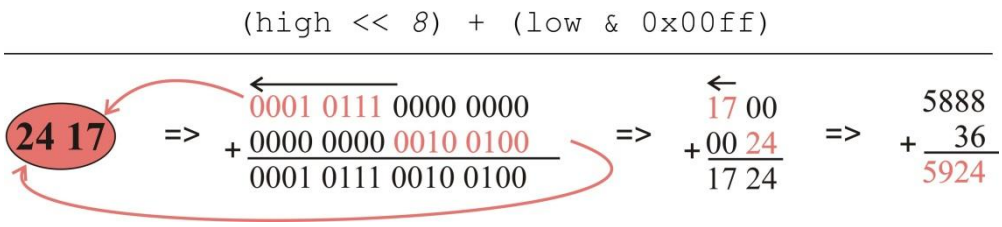
Ještě než začne jakékoliv vykreslení, je potřeba spočítat hodnoty vzorků, které budeme chtít zobrazit. O to se nám postará metoda `countSamples()` zmíněná výše. Aby bylo pozdější posouvání grafem co nejrychlejší, zvolil jsem řešení, kdy jsou všechny spočtené hodnoty celé zvukové stopy uloženy do pole, ze kterého se později získají pouhým překopírováním. Tím se při posunutí vlny ušetří nemalé množství operací, samozřejmě na úkor spotřebované paměti.

Inspirace na princip výpočtu byly získány z internetové stránky [5].

Získání vzorků

Jak bylo zmíněno v kapitole 2.4 ve druhém odstavci, zvuková data jsou uložena v souboru ve vzorcích. Z datového proudu, který jsme získali od třídy `AudioSystem`, načteme do pomocného pole data, která následně zpracujeme. Nemůžeme ovšem najednou načíst všechna zvuková data, aby nedošlo k přeplnění paměti a zastavení běhu aplikace. Na druhou stranu načítání a zpracování bajtů jednoho po druhém by bylo z důvodu příliš častého volání metody velmi pomalé. Aby k těmto problémům nedocházelo, má pole velikost 50 milionů bajtů.

Má-li vzorek velikost 16 bitů, skupinu dvou po sobě jdoucích bajtů ještě nemůžeme považovat za jeden vzorek levého nebo pravého kanálu. Bajty jsou totiž uloženy v pořadí *little-endian* (znamená, že nejméně významný bajt je uložen jako první). Abychom získali velikost amplitudy z dvojice bajtů, musíme jejich pozici prohodit. Na následujícím obrázku (viz Obrázek 9) je výpočet všech vzorků naznačen. Vyšší bajt, tedy 17_h, musíme bitově posunout o osm bitů doleva a přičíst ho k bajtu nižšímu, tedy 24_h.



Obrázek 9: Získání hodnoty ze vzorku

Tento postup musíme aplikovat na každou dvojici bajtů zvukových dat příslušících prvnímu kanálu. Jelikož v aplikaci používám vždy jen první zvukový kanál, který nám pro vykreslení vlny postačuje, jsou vzorky dalších kanálů ignorovány. Pokud tedy uvažujeme zvukovou stopu s dvěma kanály a 16 bitovou velikostí vzorku, jsou vždy dva bajty přečteny a zpracovány a další dva pouze přečteny a zahozeny. Tímto způsobem zpracujeme všechny prvky v načteném poli a znovu jej naplníme novými zvukovými daty. Celý tento cyklus se opakuje až do spočtení všech vstupních dat. Nové hodnoty jsou ukládány do pole `samplesByte[]`.

```
while(bytes > 0){
  low = (int) byteBuffer[index];
  index++;
  high = (int) byteBuffer[index];
  index++;
  sampleAverage += (high << BITS_IN_BYTE) + (low & 0x00ff);
  sampleIndex++;
  if(sampleIndex % samplsPerPixel == 0){
    float max = 0.02f;
    float value = (float)(sampleAverage / (samplsPerPixel * 65536.0f));
    samplesByte[drawingIndex] = (graphHeight - ((graphHeight/FOUR) * value / max))/2;
        drawingIndex++;
        sampleIndex = 0;
        sampleAverage = 0;
    }

  index += step;

  if(index >= bytes){
    index = 0;
    bytes = audioInputStream.read(byteBuffer);
  }
}
```

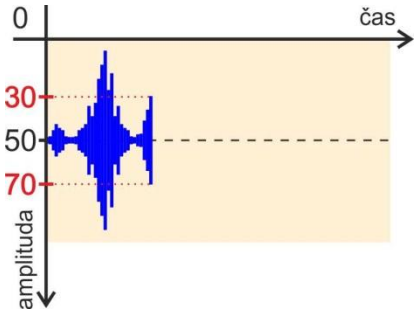
Obrázek 10: Ukázka kódu provádějícího výpočet hodnot amplitudy

Na obrázku 10 je naznačen úsek kódu, který získává ze vstupního proudu zvuková data a převádí je na hodnoty vzorků pro vykreslení. V každém kroku cyklu se z dvou hodnot vzorku získá jedna a ta se přičte do průměru. Když je v průměru dostatek vzorků, převede se na výslednou hodnotu a ta se zapíše do pole. Dokud jsou data ve vstupním proudu, cyklus se opakuje.

Vykreslení vlny

O samotné vykreslení vlny se stará třída `GraphWave`. Je zděděna od komponenty `JPanel` z balíku Swing a umístěna do zobrazovacího okna. Abychom mohli na komponentu kreslit, je obecně nutné překrýt její metodu, která ji vykresluje, tedy metodu `repaint()`.

Při každém posunutí nebo změně měřítka grafu si musí třída obstarat nová data pro zobrazení. Tak učiní pomocí metody `getSamplesToDraw()` zmíněné v kapitole 3.2.1 ve čtvrtém odstavci, která vrátí pole s novými hodnotami pro každou zobrazovanou svislou linii. Na úsečku jsou potřeba dva body, pro každý konec jeden, ale v poli je uložená vždy jen jedna číselná hodnota. Graf je totiž vytvářen symetricky podle osy x (časové osy), tudíž druhý bod se určí podle osové souměrnosti. Máme-li osu posunutou do poloviny celého grafu (bod s hodnotou 50) a má-li mít úsečka hodnotu například 20, bude ležet mezi hodnotami (50 – 20) a (50 + 20), tedy v bodech [x, 30] a [x, 70]. Tímto způsobem je v cyklu vkládána jedna úsečka po druhé do grafu (viz Obrázek 11).



Obrázek 11: Ukázka vykreslování grafu

Zvuková vlna není jediný prvek, o jehož vykreslení se musí metoda `repaint()` starat. Pokud je spuštěný přehrávač, je nutné ukazovat aktuální pozici ve stopě. Ta je v grafu zobrazena pomocí svislé červené úsečky posunující se po časové ose zleva doprava. Po dosažení konce právě zobrazované části vlny si třída vyžádá nová data pro vykreslení a ukazatel pozice začíná opět od začátku aktuálního grafu. V okamžiku, kdy se přehrávač zastaví, překreslování nepokračuje, jelikož neprobíhá žádná změna. Posledními částmi pro vykreslení jsou linky naznačující začátek a konec již vytvořených úseků a obdélník zobrazující výběr časového úseku v grafu, kterému bude přiřazen text transkripce.

3.2.3 Přehrávání zvuku

Abychom mohli v programu přehrávat zvuk, musíme zajistit komunikaci aplikace se zvukovou kartou počítače. K tomu máme v Javě hned několik možností. Pravděpodobně tou nejjednodušší je použít externí balík JMF (Java Media Framework) dostupný přímo od výrobce.

Nevhodný přehrávač z JMF:

Na začátku své práce jsem zvolil právě tuto cestu. Vytvoření přehrávače s těmito knihovnami je opravdu velmi jednoduché. Stačí pomocí statické metody `createPlayer()` třídy `Manager` vytvořit instanci přehrávače se zadanou cestou k souboru a můžeme začít přehrávat. Stejně jednoduše můžeme nastavit začátek a konec požadovaného úseku zadáním hraničních časů, zahájit start a těšit se z jednoduchosti a robustnosti implementace, kterou někdo vyřešil za nás. Pro aplikaci sloužící jako přehrávač všech možných podporovaných formátů je takováto realizace opravdu výhodná. Nejjednodušší cesta není ale vždy nejlepší. Při delším přehrávaném intervalu problém nepoznáme, ale pokud se snažíme přehrát úsek menší jak půl sekundy, zjistíme, že délka zvuku je velmi nejistá. Opakované spuštění vede k zastavení

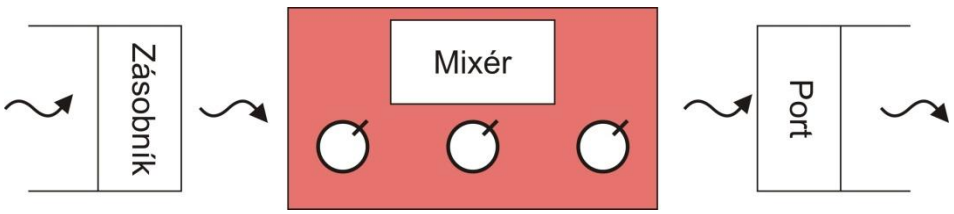
v různých časech, mnohdy ještě delších než je půl sekundy, i když vybraný úsek byl kratší. V programu tohoto typu, kdy potřebujeme mezi sebou odlišit slova, je nežádoucí přehrávat zvuk, který do našeho výběru nepatří. Z toho důvodu musel být tento postup nahrazen.

Přehrávač použitý v aplikaci:

Druhý způsob už je trochu obecnější a náročnější, avšak pracuje spolehlivě a hlavně přesně. Jediné, o co se musíme postarat, je otevření zdrojového datového zásobníku, v aplikaci se o to stará metoda `loadAudio()` příkazy:

```
DataLine.Info dataLineInfo = new DataLine.Info(SourceDataLine.class, playFormat);
SourceDataLine sdl = (SourceDataLine)AudioSystem.getLine(dataLineInfo);
```

Parametr `playFormat` je získán z hlavičky zvukového souboru, jde o vlastnosti vzorků, jejich počet, atd. Po zajištění přístupu k zásobníku už jen stačí do něho začít nahrávat požadovaná data získaná ze souboru, ze kterého čteme pomocí vstupního proudu získaného stejně jako při výpočtu hodnot grafu v kapitole 3.2.1 ve druhém odstavci, tedy pomocí systémové třídy `AudioSystem`. Zásobník už nemusíme nijak řídit. Ve chvíli, kdy bude mít dostatek dat k přehrání, odešle je dál směrem ke zvukové kartě (viz Obrázek 12). To už ovšem není naší starostí. Operační systém přehraje naše data hned, jak to bude možné.



Obrázek 12: Ukázka průchodu zvuku ke zvukové kartě

Ještě než náš zvuk zazní z reproduktorů, projde přes mixér. Ten do sebe může sloučit další data z jiných zásobníků, která mají být přehrána ve stejném čase. Jednou z jeho možností je například změna hlasitosti jednotlivých signálů. Z něho už putuje zvuk přes port napojený na zvukovou kartu až do reproduktorů.

Aby bylo možné během přehrávání stopy dělat i něco jiného, než jen poslouchat, například reagovat na výběr úseku nebo na tlačítko pro zastavení přehrávání, je nutné oddělit kód pro načítání dat do zásobníku od dalšího výkonného kódu aplikace. K tomu

je využito nové vlákno, přesněji instance třídy `PlayerThread`, která implementuje rozhraní `Runnable`, díky němuž může být její kód vykonáván nezávisle na aplikaci. Toto vlákno je vytvořeno vždy, když spustíme přehrávání, spolu s parametry odkazujícími na datový proud a počet bajtů, které mají být přehrány. Po jeho spuštění začne číst data a zapisovat je do zvukového zásobníku pro okamžité přehrání, dokud nepřepošle všechny bajty nebo nedosáhne konce stopy. Přitom stále kontroluje, jestli nebylo přehrávání pozastaveno uživatelem. Pokud ano, odstraní zapsaná, ale ještě nepřehraná data ze zásobníku a zanikne. Znovu bude vytvořeno při dalším spuštění přehrávače.

Použitím tohoto způsobu přehrávání je doba požadovaného úseku vždy stejná, jelikož do zvukové karty putují pouze námi vybraná data a žádná jiná. Nemůže se stát, že by při opakování stejné části zněl zvuk různě dlouho, jako tomu bylo u předchozího jednoduššího způsobu.

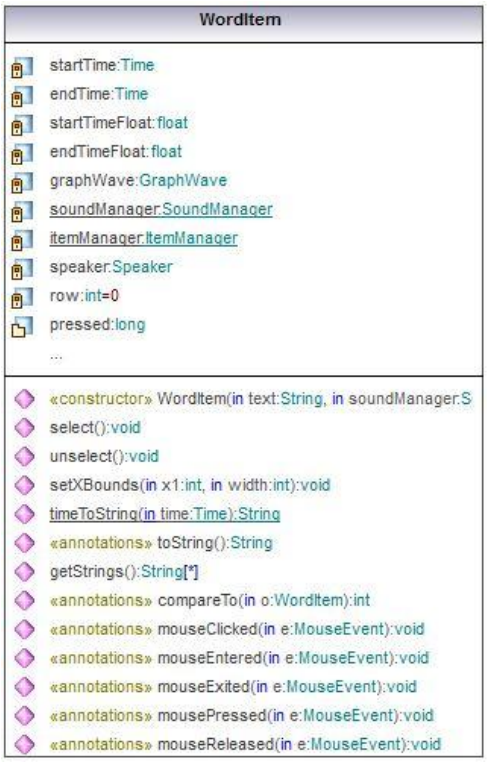
3.2.4 Změna rychlosti přehrávání

Spolu s vytvořením tohoto přehrávače se naskytlo jednoduché řešení pro změnu rychlosti přehrávání. Jak již bylo řečeno, každá hlavička wav souboru obsahuje informaci o počtu vzorků obsažených v jedné sekundě. Díky tomu zvuková karta ví, jak rychle má zvuk pouštět na výstup. Pokud tedy necháme data v původním stavu a změníme pouze informaci o množství vzorků v sekundě (upravíme hodnotu v instanci systémové třídy `AudioFormat` získané ze zvukového vstupního proudu), dosáhneme jiné rychlosti, než byla původní. Pro realizaci stačí uzavřít zvukový zásobník, změnit údaj v hlavičce a s tímto novým formátem zásobník znovu otevřít. Probíhalo-li během změny rychlosti přehrávání, je zapotřebí jej ještě předtím pozastavit a poté opět spustit. Měl-li například původní formát rychlost 44 kHz/s, bude nový zvuk s 88 kHz/s reprodukován dvakrát rychleji.

3.2.5 Uložení transkripce v paměti

Dalším požadavkem na aplikaci je určitě přehledné zobrazení a ukládání již hotových přepsaných částí. Právě k tomu slouží datová třída `WordItem`. Pro každý vybraný úsek je vytvořena vlastní nová instance zděděná od systémové třídy `JButton` a následně zobrazená jako tlačítko pod zvukovou vlnou.

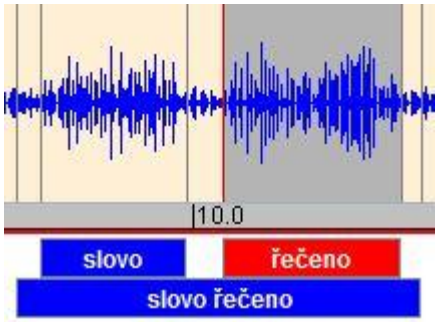
Obsahuje všechny potřebné informace jak pro korektní zobrazení v aplikaci, tak k uložení do souboru na disk (viz Obrázek 13). Jde především o čas začátku a konce úseku, příslušný text zvuku, odkaz na řečníka, který úsek vyslovil, nebo číslo řádku, kde se právě tlačítko nachází. Spolu s atributy jsou s instancí spjaty i funkce na označení prvku myší nebo smazání pomocí klávesy Delete.



Obrázek 13: UML diagram třídy WordItem

Všechny takto vytvořené prvky jsou umísťovány do panelů (pod grafem) patřících řečníkům (v kódu třída `SpeakersItemPanel`), kterým slovo náleží. Standardně je vytvořen pouze jeden řečník, tedy pouze jeden panel s bílou barvou. Tu je samozřejmě možné změnit. Pokud je řečníků v projektu více, zpřehlední barva práci s programem.

Jak bylo zmíněno v kapitole 2.2.1 ve čtvrtém odstavci o výhodě oproti konkurenčnímu softwaru, nabízí moje realizace možnost segmentovat delší úsek na několik kratších (viz Obrázek 14). O takovéto zarovnání, kdy nedochází k překrytí dvou a více segmentů, se stará každý panel řečníka. Vždy se pokusí vložit nový blok do prvního řádku a pomocí metody `checkIntersection()` implementované v panelům nadřazené třídě `ItemManager` zjistí, jestli na této pozici existuje průsečík s jiným blokem. Pokud ano, posune položku na další řádek a rekurzivně znovu kontroluje, dokud se s žádnou jinou nepřekrývá.



Obrázek 14: Ukázka transkripce

Velký důraz je kladen na jednoduchost práce s programem, což je vidět i na zarovnání tlačítek. Aby bylo v každé části zobrazovaného grafu jasné, ve kterých místech se již přeložené položky nachází, pohybují se přesně s vlnou. Pokud dojde k jejímu posunutí nebo změně měřítka, okamžitě se přepočítá velikost a umístění všech hotových bloků, aby jejich časové ohraničení vždy odpovídalo času

v grafu. Právě k takovému řešení se nabízí využít návrhový vzor Observer - Observable (neboli přeloženo jako pozorovatel - pozorovatelný), kterou nám jazyk Java nabízí. Jde o jednoduchý postup, kdy vytvoříme třídu (v našem případě `GraphObservable`), kterou zdědíme ze systémové třídy `Observable`. Ta v sobě nese informaci o začátku a konci času zobrazované vlny v grafu. V okamžiku, kdy dojde k jejich změně, jsou všichni zaregistrovaní pozorovatelé, kterým je i naše třída `ItemManager` implementující rozhraní `Observer`, obeznámeni o posunutí. Podle těchto časů dojde k přepočítání velikostí a přemístění všech přeložených úseků.

3.2.6 Uložení práce do souboru

Na disk v počítači nejsou ukládána pouze data získaná transkripcí, ale různá nastavení aplikace. V každém případě se vždy jedná o soubory typu XML, které byly vytvořeny pomocí JAXB parseru realizovaného ve třídě `Exporter`.

Celkem bylo potřeba připravit tři XML schéma jako šablony pro výsledné soubory. Jedno slouží pro konfigurační nastavení aplikace, tedy různé barvy grafu a jiných prvků, počáteční nastavení velikosti okna nebo časový interval automatického ukládání. Další schéma představuje vzor pro uložení transkripce, která musí obsahovat cestu ke zvukovému souboru, seznam všech vyskytujících se řečníků a k nim seznam přeložených slov spolu s počátečními a koncovými časy. A nakonec poslední slouží k uložení pozice a hodnot rychlých tlačítek s definovanými zkratkami, které si můžeme podle sebe upravovat a uspořádávat do kategorií.

Po takovém vytvoření schémat musíme provést jejich převod do datových Java tříd. Toho docílíme pomocí předem nainstalovaného programu XJC a zadáním následujícího příkazu do příkazového řádku:

```
xjc *.xsd –d (cilový adresář) –p (název výstupního jar balíku)
```

Parser XJC projde všechny elementy výsledného XML souboru a převede je do tříd. Ty je výhodné ještě přeložit (zkompilovat) a všechny dohromady zabalit do JAR balíku (jedná se o archivní soubor programovacího jazyka Java založený na ZIP kompresi). Já jsem pro tuto práci použil již dříve vytvořený skript, který tento postup provede a navíc do souboru přibalí i programovou dokumentaci vytvořených tříd.

Posledním krokem je připojení vygenerovaného balíku k našemu projektu. O tuto spolupráci se postará vývojové prostředí Eclipse. Poté už můžeme data z balíku bez problémů využívat. Chceme-li načíst například XML soubor s transkripcí do programu, JAXB parser jej celý přečte a rozdělí jeho data do tříd, kde s nimi můžeme pracovat jako se všemi jinými. Tedy podle nich načíst zvukový soubor, vytvořit instance řečníků a jednotlivých úseků a naplnit je načtenými údaji. Při zápisu postupujeme přesně opačným způsobem. Nejprve našimi vytvořenými daty naplníme pomocné třídy a ty následně parser zapíše jednorázově zpět do souboru.

Automatické ukládání

Abychom neztratili vytvořenou práci ignorováním ukládání do souboru a například následným výpadkem elektřiny, může se o zálohu postarat aplikace za nás. Po doběhnutí časovače program vytvoří nové vlákno (v kódu třída `SavingThread`), které nám na pozadí naši práci automaticky archivuje. Po dokončení své práce zabliká v levé dolní části pracovního okna upozorňující zpráva a opět je spuštěn časovač. Aby náhodou nedocházelo ke zbytečnému zahlcení disku automaticky uloženými daty, je v jejich složce udržováno pouze pět posledních souborů, ostatní jsou mazány.

4 Dosažené výsledky

Asi jediným časovým údajem, který má smysl v aplikaci hodnotit, je doba výpočtu hodnot pro vykreslení. Druhým měřeným faktorem může být spotřeba paměti. Všechny výsledky jsou získávány na počítači s čtyř-jádrovým procesorem Intel Core2 Quad Q9000 o taktu 2 GHz a pamětí 4 GB pod operačním systémem Windows 7.

4.1 Spotřeba paměti

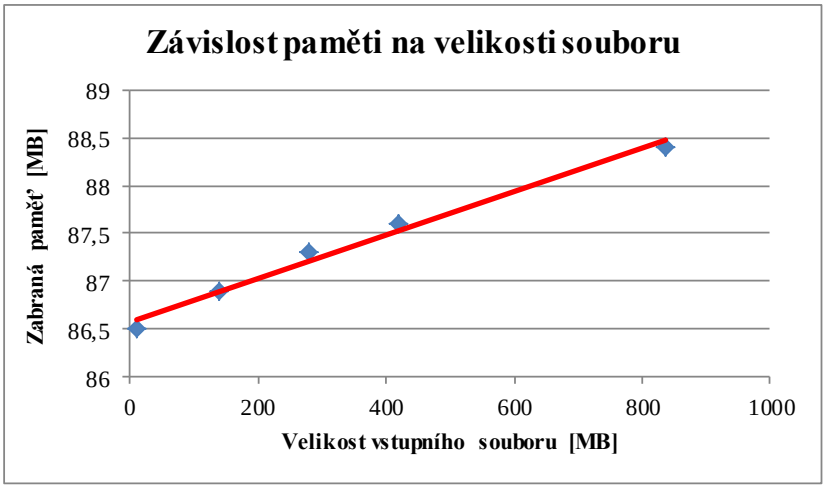
Spustíme-li naprosto triviální konzolový program, který pouze vypíše řádku textu, je mu přiděleno asi 4 MB paměti. Pokud k němu přidáme jediné dialogové okno a zobrazíme jej, paměť okamžitě vzroste na 12,5 MB. Ovšem po roztáhnutí zobrazeného okna se dostaneme až na hodnotu 18 MB. Jak je vidět na tomto jednoduchém příkladu, jazyk Java s pamětí opravdu moc nešetří. Na vyšší spotřebě se jistě podílí i nutnost spuštěného běhového prostředí JVM při běhu jakékoliv Java aplikace.

Nyní se už zaměříme na naši aplikaci, tedy Transcriber. Spustíme ji a přímo necháme načíst zvukovou stopu. Aniž bychom něco dalšího prováděli, zjistíme stav použité paměti. Tento postup budeme opakovat pro různé velikosti vstupních souborů, vždy se ovšem bude jednat o stejný zvukový formát (datový tok s frekvencí 48 kHz, 16 bitovou velikost vzorku, stereo).

velikost souboru [MB]	11,6	139,4	278,9	418,0	836,5
paměť [MB]	86,5	86,9	87,3	87,6	88,4

Tabulka 1: Závislost spotřebované paměti na velikosti vstupního souboru

Jak je z tabulky patrné, rozdílné množství spotřebované paměti při různých velikostech vstupního souboru je zanedbatelné. V grafu (viz Graf 1) si můžeme navíc všimnout, že je paměť na velikosti souboru lineárně závislá. Je to logické, jelikož pro jeden megabajt zvukových dat napočítáváme stále stejný počet hodnot k vykreslení. Pokud bychom tedy chtěli pracovat se souborem větším než 836 MB, který byl v testu zahrnut, nemusíme mít strach, že vlivem jeho velikosti bychom neměli dostatek paměti.



Graf 1: Závislost spotřebované paměti na velikosti vstupního souboru

Množství zabrané paměti je změřeno pouze pomocí správce úloh operačního systému. Hodnota zahrnuje i volnou paměť přidělenou procesu, která je připravena pro alokaci. Z toho důvodu se zdá být spotřeba paměti větší.

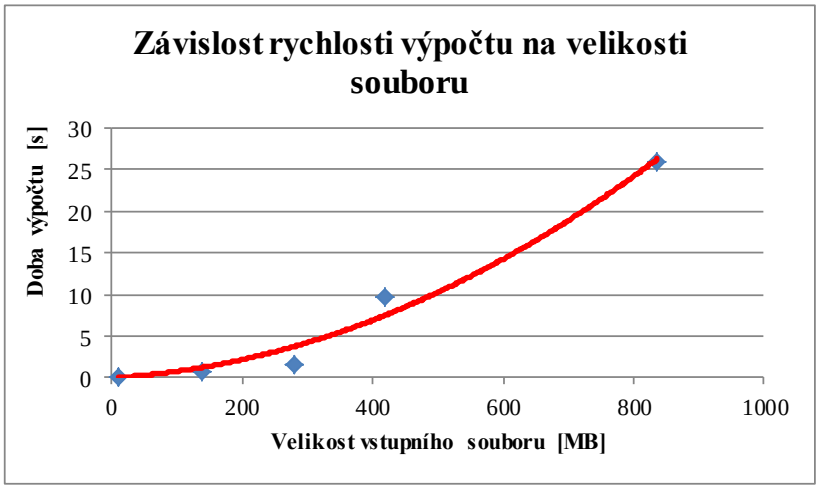
4.2 Doba výpočtu

Druhý měřeným faktorem byla doba potřebná na výpočet hodnot pro vizualizaci zvuku. Tento proces probíhá při načítání zvukového souboru do aplikace a je počítán pro celý soubor. S větším souborem tedy bude růst i doba počátečního výpočtu. Ještě dodám, že délka výpočtu je určená podle skutečného uplynulého času, nejedná se o dobu, kdy byl výkonnému vláknu přidělen procesor.

velikost souboru [MB]	11,6	139,4	278,9	418,0	836,5
doba načítání [ms]	50	726	1536	9698	26008

Tabulka 2: Závislost doby výpočtu na velikosti vstupního souboru

I zde si můžeme povšimnout, že první tři časy stoupají lineárně, jak by také teoreticky správně měly. Proč ovšem při vyšší velikosti nastane takový nárůst doby, nevím. Jednou z možností je, že procesor není úloze přidělován rovnoměrně po celou dobu trvání výpočtu, což může vést k nelinearitě.



Graf 2: Závislost rychlosti výpočtu hodnot grafu na velikost vstupního souboru

4.3 Ostatní výsledky

Stejně jako předešlé hodnoty bychom mohli změřit rychlost generování výstupního XML souboru nebo jeho velikost. Budeme-li ovšem hovořit o reálné situaci, nemá tento test podstatný význam. Pokud uložíme transkripci o 1000 položkách, její velikost na disku je asi 200 KB a rychlost uložení je téměř okamžitá (přesněji asi 50 ms).

Pravděpodobně větší rozsáhlejší soubory ani nebudeme vytvářet, tudíž není nutnost je do testu zahrnovat.

4.4 Souhrn výsledků

Pokud budeme muset pracovat s velmi rozsáhlými soubory, teoreticky není aplikace omezena nedostatkem paměti. Jak jsme zjistili, žádný podstatný vliv na spotřebu paměti velikost zvukového souboru nemá. Pouze bude nutné déle čekat při úvodním výpočtu dat. Na druhou stranu, další práce s aplikací bude po načtení stejně rychlá jako by byla s malým souborem. Na rychlost běhu programu nemají rozsáhlá vstupní data vliv. Snad jediný problém by mohl nastat u přetečení rozsahu nějakých proměnných důvodem extrémní velikosti dat. Aby ovšem mohl nastat, museli bychom vložit soubor o velikosti asi 900 GB, tudíž ani toto omezení není příliš reálné, jelikož jenom na načtení bychom čekali několik hodin.

5 Závěr

Cílem této práce bylo vytvoření funkčního programu sloužícího pro transkripci zvukových dat do textových. Mezi hlavními požadavky byly jednoduchá obsluha aplikace a umožnění zpracování rozsáhlých PCM dat. Těch bylo docíleno během několika hodin strávených při tvorbě programu.

Aplikace načítá 8 a 16 bitové wav soubory jakékoliv velikosti. Po spočítání hodnot pro graf celé zvukové vlny provede vizualizaci a umožní jeho volné posouvání a zvětšování za účelem přesnějšího a přehlednějšího zobrazení. V grafu jsme schopni označovat časové úseky, jim následně přiřadit patřičný text a zobrazit jej jako tlačítko pod vlnou se stejnou šířkou, jakou má úsek. Všechna vytvořená tlačítka se posunují a zvětšují stejně jako graf, aby byla vždy přesně určena již hotová část. Přehrávač dovoluje různé rychlosti přehrávání, počínaje od poloviční až po dvojnásobnou. Nakonec je implementováno i automatické ukládání a možnost exportu a importu projektu z/do aplikace.

V programu není omezena vstupní velikost zvukových dat, zvládá zpracovat jakoukoliv a přitom nijak výrazně nevzrůstá jeho spotřeba paměti. Jediné, co výkon trochu brzdí, je úvodní načítání zvukové vlny. Soubor o velikosti okolo 100 MB zvládne hravě celý zpracovat do jedné sekundy.

Mezi rozšířeními projektu by zcela jistě mohla být podpora MP3 formátu. Pokud by nastala potřeba pracovat s wav daty většími než 500 MB, vyšší rychlosti načítání je možné dosáhnout paralelizací výpočtu hodnot grafu vedoucímu k vyššímu výkonu na moderních vícejádrových procesorech.

Práce na tomto projektu pro mě byla obrovským přínosem a zkušeností. Blíže jsem se seznámil s principem zpracování a přehrávání zvukových dat v Javě a především strukturou uložení vzorků ve WAV formátu. Rozšířil jsem si znalosti o vytváření okenních aplikací, které je opravdu oproti například jazyku C# mnohem složitější. I když mám pocit, že odvětví informačních technologií postupně směřuje především k možnosti ovládat automatizované roboty, které za nás budou dělat manuální práci, jsem rád, že jsem do něho mohl možná i jen kouskem své práce přispět. Snad toho nebudu muset někdy v budoucnu litovat.

Použitá literatura

1. **BOUDAHMANE, Karim, a jiní.** *Transcriber.* [počítačový program] Ver. 1.5.1. Francie, 2008 [Citace: 16. duben 2012.]. Program pro transkripci zvuku na text. <http://trans.sourceforge.net>.

2. **DHIT, Amit.** *The Digital Consumer Technology Handbook.* 1. edition. Oxford : Newnes, 2004. str. 656. ISBN 0-7506-7815-1.

3. **KREJČÍ, Richard.** *Jemný úvod do jazyka XML.* [Online] 14. září 2001. [Citace: 16. Duben 2012.] <http://www.grafika.cz/art/webdesign/uvodxml_1.html>.

4. **MLÝNKOVÁ, Irena, a jiní.** *XML technologie: Principy a aplikace v praxi.* První vydání. Praha : Grada Publishing, a.s., 2008. str. 272. ISBN 978-80-247-2725-7.

5. **SIMON, Jonathan.** *Build an Audio Waveform Display.* [Online] 27. Květen 2007. [Citace: 16. duben 2012.] <http://codeidol.com/java/swing/Audio/Build-an-Audio-Waveform-Display/>.

6. **WILSON, Scott.** *WAVE PCM soundfile format.* [Online] 20. Leden 2003. [Citace: 16. duben 2012.] <https://ccrma.stanford.edu/courses/422/projects/WaveFormat/>.

7. **YADAV, Abhishek.** *Digital Communication.* 1. edition. New Delhi : University Science Press, 2008. str. 164. ISBN 8131803333.

Seznam obrázků

Uživatelská příručka:

Přílohy

Transcriber

Uživatelská příručka

Plzeň, 2012 Tomáš Kubový

Obsah

Instalace běhového prostředí

Java je sice jazyk platformně nezávislý, pro svoji práci ovšem potřebuje speciální běhové prostředí, jímž je právě již dříve (v kapitole 3 v prvním odstavci) zmíněná *Java Virtual Machine* (dále jen JVM). Ta je samozřejmě různá pro každý operační systém. Abychom mohli tedy na svém počítači program spustit, je nutné si takové prostředí obstarat. Pokud jej nemáme, můžeme ho získat například na internetových stránkách výrobce (`www.java.com`). Minimální verze pro práci programu je `Java SE 1.6`. Po instalaci kamkoliv na disk bychom měli mít možnost spouštět programy vytvořené v jazyce Java.

Spuštění programu

Pokud jsme úspěšně nainstalovali běhové prostředí, neměl by být problém program spustit. Po rozbalení archívu s aplikací obsaženém na přiloženém CD jednoduše otevřeme soubor `Transcriber.jar` a počkáme, co se stane. Mělo by se zobrazit hlavní okno, ovšem bohužel takové chování není pravidlem. Nestane-li se tak, je nutné spustit program z příkazové řádky.

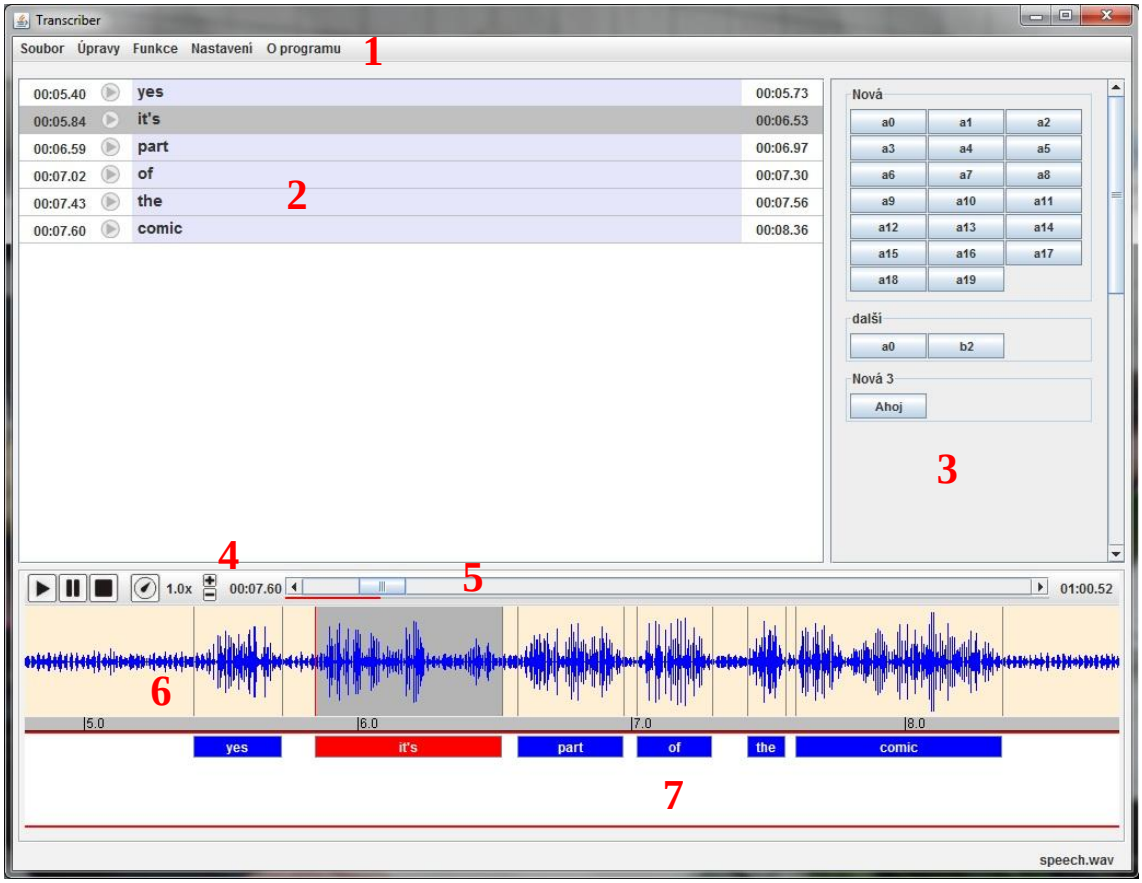
Příkazovou řádku vyvoláme například stisknutím tlačítka `Start` a napsáním příkazu `cmd` (v systémech Windows 7 a Windows Vista), stisknutím `Start`, vybráním programu `Spustit` a zadáním `cmd` (v systému Windows XP) nebo vyvoláním programu `Terminál` (v systému Linux). Po načtení se pomocí příkazu `cd` přesuneme do složky s naším programem a spustíme jej příkazem:

```
java –jar Transcriber.jar
```

V tuto chvíli by už neměl být problém s načtením aplikace a jejím během.

Je vhodné, aby ve stejné složce spolu se spouštěcím souborem byl i soubor `startConfig.xml`, který obsahuje nastavení okna aplikace (například rozměr nebo aktuálně používané barvy okna) a soubor `defaultBtns.xml` nesoucí informaci o tlačítkových zkratkách. Oba tyto soubory se nedoporučuje měnit ručně v textovém editoru, neboť při špatně zadaných hodnotách by mohlo dojít k nechtěnému chování aplikace. Jejich editace je umožněna přímo v programu.

Hlavní okno aplikace



Obrázek 15: Ukázka hlavního okna aplikace

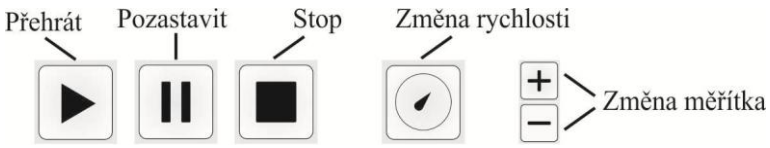
1. Menu
2. Seznam vytvořených položek
3. Panel rychlých zkratek
4. Tlačítka pro přehrávání, změnu rychlosti a změnu měřítka
5. Rolovací lišta pro posouvání zobrazovaného úseku zvukové vlny
6. Vykreslená zvuková vlna
7. Panel s řečníky a jejich příslušnými položkami

Načtení zvukové stopy

Než začneme pracovat, je nutné načíst zvuková data. To provedeme stisknutím tlačítek `Soubor → Otevřít`. Zobrazí se nám dialogové okno pro výběr souboru typu wav. Po zvolení námi požadované stopy proběhne příprava celého hlavního okna, smažou se data z předchozí transkripce a vypočítá se tvar zvukové vlny. V této chvíli je program připraven pro práci.

Přehrávání a posouvání zvukové stopy

Po dokončení fáze načítání můžeme začít přehrávat. Celý tento proces ovládáme buď pomocí tří tlačítek (viz Obrázek 16) umístěných nad zvukovou vizualizací, nebo pomocí klávesnice. Po stisknutí tlačítka `Přehrát`, jak již název napoví, začne přehrávání zvuku od místa označeného ukazatelem pozice. Tím je právě červená svislá linka pohybující se v grafu při přehrávání. Po načtení je samozřejmě na začátku. Běh přerušíme tlačítky `Pozastavit` nebo `Stop`. Při pozastavení zůstane ukazatel



Obrázek 16: Tlačítka pro přehrávání zvuku a změnu měřítka

na aktuálním místě, odkud bude přehrávání opět pokračovat. Tlačítkem `Stop` se přesune příští čas spuštění buď na začátek vybraného úseku, nebo není-li úsek vybrán, tak na úplný začátek zvukové stopy.

Pro pohyb v grafu slouží posuvník umístěný nad ním. Celá délka zvukové stopy je transformována do velikosti panelu s posuvníkem, aby přesně kopíroval pozici stejně jako zobrazený úsek. Tažením posuvníku tedy posunujeme právě zobrazenou část v čase dopředu nebo dozadu. Stejné funkce, avšak přesnější, docílíme, když nad zvukovou vlnou stiskneme pravé tlačítko myši a táhneme požadovaným směrem. V tu chvíli je pozice, nad kterou bylo tlačítko stisknuto, vždy pod ukazatelem myši a dynamicky se posunuje s jeho polohou.

Změna měřítka zvukové vlny

Pro změnu měřítka zobrazovaného úseku zvukové vlny můžeme použít buď tlačítka + a − umístěna napravo od tlačítek pro přehrávání, nebo rychlejší způsob, jímž je

otočení kolečka myši nahoru nebo dolu nad grafem. Velikost úseku se v každém kroku zvětší nebo zmenší na délku v rozmezí od 2 do 32 sekund. Pokud použijeme myš, čas, který je pod ukazatelem před změnou, je stejný jako po změně měřítka (nemusí platit na začátku a konci stopy, kdy je při zvětšování délky úseku graf zarovnán ke kraji).

Změna rychlosti přehrávání

Rychlost můžeme měnit přímo během přehrávání či v zastaveném stavu. Učiníme tak buď pomocí kláves + a – na klávesnici, nebo pomocí tlačítka `Změna rychlosti`. Na výběr máme z možností od poloviční až po dvojnásobnou rychlost.

Vytvoření prvku transkripce

Abychom mohli vytvořit prvek transkripce, musíme mít načtený nějaký zvukový soubor. Pokud jsme tak učinili, označíme námi požadovaný úsek v grafu zvukové vlny stejným způsobem, jako běžně označujeme například slovo v textu, tedy tažením myši za současného držení levého tlačítka. Po dokončení výběru je pod grafem vytvořen panel o velikosti a pozici shodné jako má vybraný úsek. Stejně tak přibyde v seznamu všech položek záznam odkazující právě na výše zmíněný panel. Do této položky můžeme hned po dokončení výběru začít psát odpovídající slovo.

Jestli ještě chceme úsek přehrát, pro spuštění použijeme například mezerník. Po dosažení konce můžeme opětovně spustit podle potřeby, kolikrát požadujeme. Textové pole pro editaci položky zůstane stále aktivní. Hned, jak si budeme jisti překladem, můžeme začít psát a nakonec potvrdit klávesou Enter.

Po vytvoření prvku máme stále možnost změnit jeho hraniční časy. Kliknutím myší na položku pod zvukovou vlnou nebo v seznamu provedeme její výběr a označení úseku v grafu. Pokud najedeme myší k levé nebo pravé straně úseku, dojde ke změně ukazatele šipky na symbol ↔. V tu chvíli po stisku levého tlačítka hranicí posouváme.

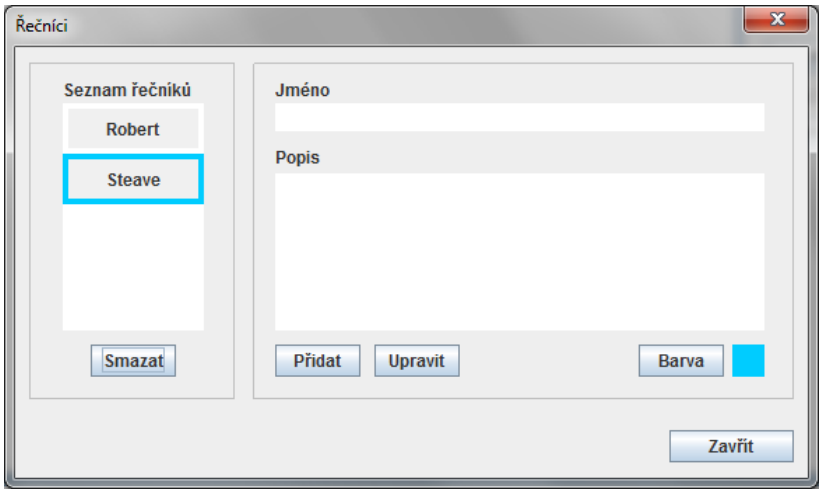
V grafu můžeme aktuální pozici přehrávače posouvat i pomocí šipek klávesnice doleva a doprava, kdy každým jejich stisknutím přemístíme ukazatel asi o tři pixely. Časový skok tedy záleží na aktuálním měřítku. Při současném držení klávesy Shift a šipky bude probíhat vytváření nového úseku s hranicemi určenými až po uvolnění klávesy Shift.

Smazání prvku transkripce

Smazání prvku provedeme jeho označením v seznamu nebo na jeho umístění pod grafem a stisknutím klávesy Delete.

Nastavení řečníků

Pokud se ve zvukové stopě vyskytuje více mluvících osob, je možné jejich slova přiřazovat do vlastních panelů a tím je od sebe odlišovat. Jejich nastavení provedeme stisknutím tlačítka menu `Nastavení` → `Nastavení řečníků`, čímž vyvoláme dialogové okno se seznamem definovaných osob (viz Obrázek 17).



Obrázek 17: Okno pro nastavení řečníků

V tomto okně máme možnost přidávat, upravovat a mazat osoby. Spolu se jménem a popisem můžeme nastavit i barvu, kterou bude mít příslušný panel pod grafem zvukové vlny, díky kterému bude jednodušší orientace v již vytvořených slovech.

Nové přeložené úseky jsou vkládány do panelu právě vybraného řečníka. Aktuálně vybraný poznáme podle červeného ohraničení. Přepínat mezi nimi můžeme pouhým kliknutím myší. Je-li počet řečníků větší než dva, nevejdou se všichni najednou do zobrazovaného okna a musíme mezi nimi posouvat. Toho docílíme stejně jako při posouvání grafu, tedy pravým tlačítkem myši a tahem příslušným směrem nahoru nebo dolu pro požadovaného řečníka, nebo doleva a doprava pro pohyb po časové ose.

Pokud omylem přidáme slovo jinému řečníkovi, prvek vybereme a numerickými klávesami `1`, `2`, `3` a tak dále, podle počtu řečníků, jej můžeme mezi panely přehazovat.

V aplikaci musí být vždy alespoň jeden řečník, proto nás ani nenechá toho posledního smazat.

Seznam již vytvořených slov

Po vytvoření transkripce nějakého úseku je vytvořen jak příslušný panel pod grafem, tak i položka umístěná v přehledném seznamu všech již vytvořených přepisů. Položky jsou řazeny pod sebe podle počátečního času a každá z nich obsahuje text jí přiřazený spolu s ohraničujícími časy. Pokud potřebujeme nějakou z částí upravit, máme možnost i zde. Dvojklikem myší na požadovaný prvek v seznamu jej změníme na editovací panel, ve kterém můžeme změny provést. Druhou možností je pravým tlačítkem myši vyvolat menu, kde si patřičnou úpravu vybereme.

Při označení položky v seznamu dojde k posunutí grafu vlny tak, aby byla v tu chvíli viditelná.

Seznam rychlých zkratek

Pravý horní panel v hlavním okně aplikace slouží jako seznam rychlých zkratek (viz Obrázek 18). Je tvořen z tlačítek, která si můžeme nadefinovat podle vlastní potřeby a poté použít při editaci textu místo vypisování klávesnicí.

Obrázek 18: Ukázka rychlých zkratek

Pro povolení editace stačí nad panelem stisknout pravé tlačítko myši a vybrat v menu položku `Editovat`. Barva pozadí panelu se změní na červenou a můžeme začít upravovat kategorie a prvky dle libosti. Opět pomocí vyvolání menu pravým tlačítkem máme možnost změnit, přidávat či mazat jak zkratky, tak i jejich kategorie.

Editaci ukončíme stejně, jako jsme ji začali, tedy vyvoláním menu a stisknutím `Ukončit editaci`. Následně budeme dotázání, jestli chceme seznam uložit, nebo vrátit změny do původního stavu před úpravami. Pokud souhlasíme s uložením, seznam se zapíše na disk do souboru `defaultBtn.xml` a při příštím spuštění aplikace bude opět načten do stejného stavu.

Vkládání rychlých zkratek

Abychom mohli zkratky použít, musí být aktivní panel pro editaci textu v seznamu položek. V takovém stavu je právě po vybrání úseku v grafu vlny nebo po vyvolání úpravy v seznamu. Poté je po stisknutí zkratky vložen její příslušný text do editovacího panelu tam, kde se nachází textový kurzor.

Import / Export transkripce

Uložení transkripce do souboru se provede stisknutím tlačítka v menu `Soubor→Exportovat`. Zobrazí se standardní dialogové okno pro uložení souboru, kde vybereme požadované místo a stiskneme tlačítko `Save`. Následně budeme informováni zprávou o výsledku uložení.

Pro načtení přepsané práce postupujeme obdobným způsobem, opět tlačítkem `Soubor→Importovat`. V zobrazeném okně vybereme umístění XML souboru s transkripcí a potvrdíme tlačítkem `Open`. Pokud vše proběhne v pořádku, následuje načtení zvukového souboru a vložení načtených položek patřičným řečníkům. Není-li však zvuková stopa nalezena v umístění, které je obsaženo v XML souboru, budeme vyzváni pro její vyhledání opět dialogovým oknem pro otevírání souborů.

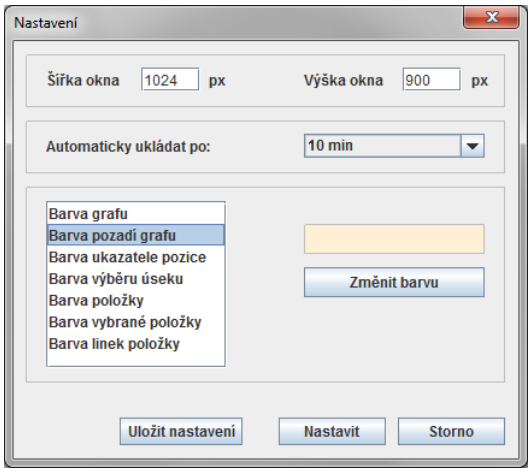
Formát vstupního a výstupního XML souboru je přesně daný schématem `export.xsd` přiloženým na CD. Jeho nedodržením by se pravděpodobně nepodařilo správně nebo vůbec data načíst. Obsahuje informaci o umístění zvukové stopy, seznamu řečníků s popisem a barvou panelu a seznam všech přeložených úseků spolu s informací o řečníkovi a ohraničujících časech seřazené vzestupně podle počátečních časů.

Automatické ukládání

Možnost automatického ukládání se nastavuje v menu pod položkou `Nastavení`. Je-li aktivní, je v definovaných intervalech ukládána naše práce do složky `autoSave`, nacházející se ve stejném umístění jako spouštěcí soubor aplikace. V této složce je udržováno posledních pět archivů, které jsou postupně přepisovány novými, aby nedocházelo ke zbytečnému zahlcování disku.

O automatickém uložení jsme během práce informováni v levém dolním rohu hlavního okna zprávou o dokončení.

Nastavení aplikace



Obrázek 19: Okno pro nastavení

Po stisknutí položek v menu `Nastaveni` a `Nastavení` se nám zobrazí okno (viz Obrázek 19), ve kterém můžeme měnit jak počáteční velikost hlavního okna, tak interval automatického ukládání nebo nejrůznější zbarvení aplikace.

Po dokončení změn tlačítkem `Uložit nastavení` zapíšeme konfiguraci na disk do stejné složky jako je spouštěcí soubor aplikace. Toto nastavení bude aplikováno a použito při dalším startu aplikace. Pokud by byl soubor poškozen nebo pokud bychom se chtěli vrátit k původním hodnotám, stačí jej smazat. Při dalším spuštění programu budeme informováni o nenalezení souboru, použijí se defaultní hodnoty a nový soubor bude vytvořen.

Seznam klávesových zkratek

mezerník	spustí nebo pozastaví přehrávání
šipka ← →	posun ukazatele aktuální pozice
šipka ↓	zahájení editace vybrané položky
kolečko myši	změna měřítka grafu
Shift + šipka ← →	výběr úseku pomocí klávesnice
Ctrl + šipka ← →	označení další nejbližší položky vlevo (vpravo)
NUM 1, 2, 3… 9	změna aktuálního řečníka
+ nebo -	změna rychlosti (rychleji, pomaleji)
Ctrl + O	zobrazí nabídku pro otevření WAV souboru
Ctrl + S	uloží práci do otevřeného XML souboru nebo zobrazí nabídku pro vybrání umístění
CTRL + I	zobrazí nabídku pro importování XML souboru do aplikace