VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

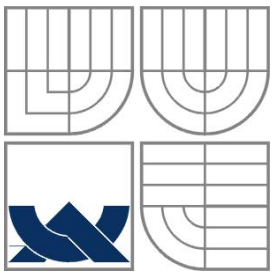
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

IMPLEMENTACE DETEKTORU KLÍČOVÝCH SLOV DO MOBILNÍHO TELEFONU (SYMBIAN 60)

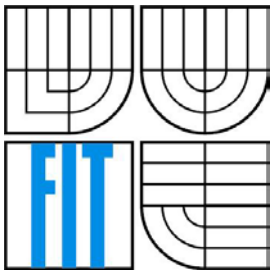
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. TOMÁŠ CIPR
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

IMPLEMENTACE DETEKTORU KLÍČOVÝCH SLOV DO MOBILNÍHO TELEFONU (SYMBIAN 60)

KEYWORD SPOTTING IMPLEMENTATION FOR MOBILE PHONE (SYMBIAN 60)

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. TOMÁŠ CIPR

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. IGOR SZÖKE

BRNO 2007

Zadání diplomové práce

Implementace detektoru klíčových slov do mobilního telefonu (Symbian 60)

Keyword Spotting Implementation for Mobile Phone (Symbian 60)

Vedoucí:
Szöke Igor, Ing., UPGM FIT VUT
Přihlášen:
Cipr Tomáš, Bc.
Zadání:

1. Seznamte se s operačním systémem Symbian 60 a vývojovými prostředky pod tento operační systém.
2. Seznamte se s implementací jednoduchého detektoru klíčových slov.
3. Naimplementujte extrakci příznaků, algoritmus Forward pass neuronové sítě a dekodér.
4. Ověřte funkčnost aplikace, proveďte testy zaměřené na přesnost a rychlost algoritmu na mobilním telefonu.
5. Shrňte dosažené výsledky a navrhněte směry dalšího vývoje.

Část požadovaná pro obhajobu SP:
Body 1,2 a část bodu 3 ze zadání.
Kategorie:
Umělá inteligence
Operační systém:
MS Windows nebo Linux

Licenční smlouva

Licenční smlouva je uložena v archívu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Detektor klíčových slov je jednou z aplikací automatického rozpoznávání řeči. Úkolem detektoru je určit, ve kterých místech souvislého toku řeči se vyskytují slova ze zadaného seznamu. Detektor klíčových slov najde řadu uplatnění mimo jiné i v mobilních telefonech, např. pro jejich ovládání hlasem. S nástupem OS Symbian se otevřela možnost jak detektor implementovat i pro tato zařízení.

Zpráva popisuje jednak teoretická a odborná východiska realizace detektoru a také jeho následnou implementaci. Nejdříve je uveden operační systém Symbian s ohledem na praktické řešení úkolu. Dále je popsán způsob detekce klíčových slov od vstupního řečového signálu až po výstup, zda a která slova byla nalezena. Následně je prezentován objektový návrh detektoru a podrobněji popsána jeho implementace. Závěrem jsou shrnuty dosažené výsledky a nastíněn další vývoj.

Klíčová slova

Symbian, EPOC, Series 60, S60, mobilní telefon, smartphone, zpracování řeči, rozpoznávání řeči, detektor klíčových slov, extrakce příznaků, mel spektrum, banky filtrů, perceptuální lineární predikce, PLP, DCT, neuronová síť, perceptron, Viterbi decoder.

Abstract

Keyword spotting is one of the many applications of automatic speech recognition. Its purpose is determining spots in given utterance in which some of the specified words were spoken. Keyword spotting has a great potential to enhance performance of new applications as well as the existing ones. An example could be a mobile phone voice control. Due to OS Symbian's coming to the market it is even possible for end user to implement a keyword spotting for a mobile phone on his or her own.

The thesis describes theoretical prerequisites for keyword spotting and its implementation as well. Firstly the OS Symbian is presented with respect to the given task. Secondly each step of keyword spotting process is described. Finally the object design of keyword spotter is presented followed by implementation description. The thesis concludes with results review and notes on possible improvements.

Keywords

Symbian, EPOC, Series 60, S60, mobile phone, cellular phone, smartphone, speech processing, speech recognition, keyword spotting, feature extraction, mel spectrum, filter banks, perceptual linear prediction, PLP, DCT, neural network, perceptron, Viterbi decoder.

Citace

Tomáš Cipr: Implementace detektoru klíčových slov do mobilního telefonu (Symbian 60), diplomová práce, Brno, FIT VUT v Brně, 2007

Implementace detektoru klíčových slov do mobilního telefonu (Symbian 60)

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Igora Szökeho. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Cipr
22.5.2007

Poděkování

Děkuji Ing. Igoru Szökemu za odborné vedení, užitečné podněty a připomínky, čas věnovaný konzultacím a v neposlední řadě také za pomoc při trénování neuronové sítě a při vyhodnocování výsledků.

Obsah

1 Úvod

Obor automatického rozpoznávání řeči si klade za cíl naučit stroj (počítač), aby porozuměl mluvenému slovu. Ačkoliv je tento úkol nelehký, mnoha problémům spjatým s jeho řešením již bylo porozuměno. Praktické aplikace jsou již v dnešní době za jistých podmínek v rozpoznávání řeči velmi úspěšné. I když ve spolehlivosti a přesnosti rozpoznávání se schopnostem člověka zatím nevyrovnají.

Praktickou aplikací automatického rozpoznávání řeči, kde bylo dosaženo poměrně uspokojivých výsledků, je mimo jiné detekce klíčových slov v souvislé řeči. Její využití je například při vyhledávání v řečových datech nebo pro ovládání různých zařízení hlasem. Není tedy překvapením, že se detektory klíčových slov postupně začaly stále více uplatňovat i v mobilních telefonech spolu s tím jak se zdokonalovaly jejich výpočetní prostředky. Zdokonalování hardware mobilních telefonů také umožnilo vznik otevřených operačních systémů pro tato zařízení, které poskytly nové možnosti pro vývojáře aplikací. Tyto skutečnosti byly původní myšlenkou, která určila téma tohoto projektu.

Práce využívá především poznatky publikované k tématu skupinou Speech@FIT*. Práce má implementační charakter a neklade si tedy za cíl výzkum v zájmu zdokonalení algoritmu detekce klíčových slov. Implementace vychází z ověřeného návrhu, který byl zjednodušen pro realizaci pod OS Symbian. Výsledná aplikace byla implementována jako knihovna. Ta poskytne programům, které ji budou používat, kompletní funkčnost online detektoru klíčových slov pro češtinu. Následující zpráva shrnuje teoretická a odborná východiska řešené problematiky a popisuje návrh a implementaci detektoru klíčových slov pod OS Symbian.

Zpráva je rozdělena do několika částí. V první z nich jsou popsány základní principy architektury OS Symbian. Je zde uveden přehled jednotlivých verzí s ohledem na dostupnost aplikačního rozhraní, které je pro implementaci detektoru zapotřebí. V samostatné kapitole je prezentována platforma Series 60 (S60) a její verze, které jsou úzce spjaty vždy s konkrétní verzí OS Symbian a mají vliv na implementaci a přenositelnost. V první části je dále uveden výčet vývojových prostředí pro platformu S60 se zmínkou o jejich dostupnosti. V závěrečné kapitole této části je zvolena verze platformy, která byla použita pro implementaci.

Druhá část technické zprávy se zabývá problematikou detekce klíčových slov. Nejdříve je uvedeno obecné schéma detektoru klíčových slov. Kapitoly této části jsou pak věnovány jednotlivým stavebním prvkům detektoru, který byl vybrán pro implementaci. V první kapitole jsou popsány dvě varianty extrakce příznaků. Experimentálně pak byla vybrána pro finální detektor jedna z nich. Další kapitola se věnuje klasifikátorům pro mapování příznaků na fonémy, konkrétně neuronovým sítím, které byly pro tento účel zvoleny. Poslední kapitola druhé části je určena dekodérům, které slouží pro vyhledání klíčových slov ve výstupu klasifikátoru. I zde jsou uvedeny dvě varianty, ze kterých bylo možno vybírat.

Ve třetí části je popsána implementace detektoru. První kapitola této části představuje specifika programování v jazyce C++ pod Symbian OS vzhledem k zajištění funkčnosti detektoru. Další kapitola popisuje prototyp detektoru, který byl implementován v jazyce C v rámci semestrálního projektu, na něž tato práce navazuje. Prototyp posloužil k optimalizaci některých částí aplikace a k zjednodušení počáteční fáze implementace. Poslední kapitola této části uvádí objektový návrh

* Skupina zpracování řeči na Ústavu počítačové grafiky a multimédií FIT VUT v Brně. http://www.fit.vutbr.cz/speech/

knihovny detektoru klíčových slov. V jednotlivých podkapitolách je prezentována implementace všech komponent knihovny od nahrávání audio vstupu až po aplikační rozhraní.

Předposlední část popisuje testování knihovny a dosažené výsledky. Jsou zde porovnány oba navržené a implementované dekodéry a diskutovány jejich výhody a nevýhody. Popsána je zde také dosažená přesnost rozpoznávání a faktory, které ji ovlivňují. V závěru této části jsou uvedeny problémy, které se vyskytly během implementace a jejich řešení.

V závěru zprávy je shrnut postup realizace a prezentován stav výsledného produktu. Nechybí zde také náměty na možné rozšíření knihovny v budoucnosti.

Tato práce nenavazuje na žádnou jinou bakalářskou nebo diplomovou práci.

2 OS Symbian

Operační systém Symbian je určen a stále více se prosazuje v zařízeních jako jsou mobilní telefony a komunikátory, které mají omezené hardwarové prostředky. Variabilita zařízení, na kterých Symbian běží je poměrně velká co se týče podporovaných technologií a hardwarových součástí. Způsob ovládání jednotlivých zařízení může být také značně odlišný (např. klávesnice vs. dotykový displej). Architektura OS Symbian je s ohledem na tyto skutečnosti navržena ve dvou vrstvách oddělením uživatelského rozhraní od operačního systému. OS Symbian poskytuje základní rozhraní a služby a výrobce zařízení následně zvolí platformu uživatelského rozhraní, která splňuje nejlépe jeho požadavky.

V současnosti existuje několik platforem uživatelského rozhraní i verzí samotného OS Symbian. Vzájemná kompatibilita mezi různými platformami i jejich jednotlivými verzemi je více či méně omezená. Pro implementaci byla vybrána platforma uživatelského rozhraní S60, která je v Evropě především díky velkému počtu telefonů značky Nokia, nejrozšířenější. Bylo však zapotřebí ještě zvolit její konkrétní verzi a tím současně verzi OS Symbian s ohledem na dostupnost API pro úspěšný vývoj aplikace. Implementačním jazykem bude C++.

2.1 Verze OS Symbian a multimedia API

OS Symbian je flexibilní operační systém. Výrobce zařízení není příliš omezován jaké komponenty využije, či jaké komponenty upraví pro své potřeby. Naštěstí pro vývojáře aplikací existuje klasifikace komponent OS Symbian, která je dělí do skupin na povinné a volitelné a nahraditelné a nenahraditelné. Níže bude přehledově vysvětlen význam jednotlivých tříd komponent API a následně bude uvedena komponenta pro multimedia a její dostupnost na jednotlivých verzích OS Symbian. Tato komponenta umožňuje načítání z audio vstupu, což je zásadní funkce pro detektor klíčových slov.

2.1.1 Klasifikace komponent

Dostupné API OS Symbian nezávisí jen na jeho verzi, ale také na tom jaké komponenty API podporuje konkrétní zařízení. Existují celkem čtyři skupiny komponent (viz. [1]):

- **Společné Symbian** (Common Symbian)
  Komponenty v této skupině tvoří jádro systému. Komponenty a jejich API jsou přítomny v každém zařízení s odpovídající verzí OS Symbian.

- **Společné nahraditelné** (Common replaceable)
  Tyto komponenty jsou na nejnižší úrovni OS a jejich implementace je závislá na použitém hardware.

- **Volitelné Symbian** (Optional Symbian)
  Tyto komponenty jsou volitelné pro všechna zařízení. V praxi však bývají téměř vždy implementovány. Pokud jsou tedy přítomné, jejich API musí odpovídat specifikaci dané verze OS Symbian.

- **Volitelné nahraditelné** (Optional replaceable)
  Zahrnutí těchto komponent je nepovinné. Na rozdíl od předchozí skupiny nemusí jejich API odpovídat společné specifikaci a tudíž může být odlišné i na zařízeních se shodnou verzí OS Symbian.

Pro účely detektoru klíčových slov je zásadní klasifikace komponenty pro multimedia. Ta patří do kategorie volitelných nahraditelných komponent. Naštěstí však podle [1] je její rozhraní implementováno na všech platformách (ale na příslušné verzi OS Symbian) stejně podle doporučení Symbian.

2.1.2 Multimedia Framework

Komponenta pro multimédia se (podobně jako řada dalších komponent) skládá z klientské části a serverové části. Klientská část obsahuje API pro vývoj aplikací pro práci s obrázky, nahrávání a spouštění zvukových a video souborů.

Serverová část zajišťuje funkčnost komponenty – podporu pro audio, video a obrázky, různé formáty souborů, kodeky a hardware [2]. Architektura serverové části umožňuje pomocí pluginů přidávat podporu dalších funkcí. Tyto pluginy se vytvářejí s využitím API serverové části (referenční API pro multimedia má název DevSound). Případná implementace chybějící části komponenty v podobě pluginu však není jednoduchou záležitostí a vyžaduje vývojový kit nižší úrovně (DevKit).

Komponenta pro podporu multimédií procházela vývojem, tak jak přicházely nové verze OS Symbian. S ohledem na přímočarou implementaci aplikace pro detekci klíčových slov je důležitá především podpora pro snadné zpracování vstupu z mikrofonu. Ta však nebyla v OS Symbian od první verze. Vývoj komponenty pro multimédia byl následující:

- **Symbian OS v6.x, v7.0**
  - *Multimedia server*

    Server poskytující platformě závislé rozhraní umožňující implementaci různých typů pluginů pro různé typy úloh. Jednotné API pro zpracování audio vstupu není dostupné.

- **Symbian OS v7.0s (a pozdější)**
  - *Multimedia Framework (MMF)* [3,4]

    Komponenta (Middleware) nahrazující Multimedia server podporující mimo jiné nahrávání a spouštění audio a video souborů (včetně zpracování přímého vstupu z mikrofonu/výstupu na reproduktor v případě audia). Komponenta je implementována jako ECOM plugin. Někdy je označována také jako Symbian Audio Controller.
  - *Image Conversion Library (ICL)*

    Knihovna nahrazující část Multimedia serveru určenou pro zpracování obrazu.

- **Symbian OS v8.0s**
  - *Media Device Framework (MDF)*

    Systém určený pro výrobce hardware. Slouží pro podporu implementace hardware pluginů. Ve skutečnosti se tedy jedná o definici nízkoúrovňového API pod systémem MMF.

Nejnižší verze OS Symbian, která podporuje přímé zpracování vstupu z mikrofonu, je tedy verze 7.0s. Implementace aplikace pro zařízení minimálně s touto verzí by tedy byla nejjednodušší. Nicméně řešení (zdá se) existuje i pro předchozí verze OS Symbian. Pro verzi 6.1 a 7.0 poskytuje přímo Symbian balík, který doplňuje chybějící funkčnost. Balík by měl být dopředně kompatibilní s API MMF ve verzi 7.0s. Zdrojový kód napsaný pro tento balík by tedy mělo být možné bez problému zkompilovat ve verzi 7.0s a naopak.

Naopak ve verzi OS Symbian 9.0 a vyšší bylo odstraněno API pro tvorbu MMF pluginů z důvodu nového systému zabezpečení, který byl uveden poprvé právě ve verzi 9.0. Zda v tomto

případě chybí v API i funkce samotného Symbian Audio Controlleru je otázkou. V každém případě chybějící část rozhraní je opět k dispozici ke stažení jako samostatný plugin.

2.2 Verze uživatelského rozhraní

Jak již bylo řečeno v úvodu kapitoly, uživatelské rozhraní je odděleno od operačního systému. Výrobce zařízení tak má svobodu ve volbě vhodné platformy UI. Vývojové kity včetně kompletního API pro OS jsou však vždy dostupné pro aplikační vývojáře pro danou platformu UI. Verze platformy UI je pak často přímo spjata s konkrétní verzí OS Symbian. Výběrem platformy UI pro implementaci aplikace je tedy současně zvolena verze OS Symbian. V současnosti existují celkem čtyři platformy uživatelských rozhraní pro Symbian – FOMA™ SW, Series 60, Series 80 a UIQ.

Platforma FOMA je navržena výhradně pro telefony v NTT DoCoMo 3G FOMA síti v Japonsku. Platforma Series 80 (vyvinuta firmou Nokia) je navržena pro komunikátory s kompletní QWERTY klávesnicí a velkými displeji. Níže je popsána platforma Series 60, která byla zvolena pro implementaci detektoru klíčových slov. Dále následuje pro srovnání stručný popis platformy UIQ, která se zaměřuje na v současnosti ne tolik rozšířené typy mobilních telefonů.

2.2.1 Platforma Series 60

Platforma Series 60 (S60) byla vyvinuta firmou Nokia. Je navržena pro snadné ovládání zařízení jednou rukou pomocí tlačítek a obsahuje řadu funkcí nejen přímo spojených s uživatelským rozhraním. Platformu S60 má licencovánu kromě Nokie řada výrobců mobilních telefonů: LG Electronics, Lenovo, Panasonic, Samsung, Sendo a Siemens. Prvním telefonem na této platformě byla Nokia 7650, která byla uvedena na trh v roce 2002. Počet zařízení na této platformě je v současnosti v porovnání s ostatními platformami především díky velkému počtu mobilních telefonů Nokia mnohem větší.

Platforma S60 je otevřená. Výrobci zařízení, kteří mají licenci na její používání si ji mohou upravit podle specifických funkcí daného zařízení. Především by se mělo jednat o rozšíření o další funkce, ale je také možné vynechání určitých funkcí, které nejsou z nějakého důvodu pro dané zařízení důležité.

Platforma S60 se neustále vyvíjí. Do současnosti byly vydány již tři verze (edice). Mezi hlavní změny S60 2nd Edition oproti S60 1st Edition patří podpora několika rozlišení displejů (176 x 208, 240 x 320 (QVGA) and 352 x 416). Ve verzi S60 3rd Edition je pak důležitou změnou oproti S60 2nd Edition přechod na verzi OS Symbian 9.0, která přináší nový binární formát a zdokonalenou úroveň zabezpečení.

V rámci edice navíc vznikají podverze (Feature Packs), které představují drobnější změny. V první edici byl uveden jeden, ve druhé tři a ve třetí zatím jeden Feature Pack. Verze Symbianu je pak spjatá nejčastěji nejen s danou edicí, ale může se lišit i mezi po sobě jdoucími Feature Packy.

Zpětná kompatibilita (i na binární úrovni) by měla být zaručena v dané edici platformy. S odlišnými verzemi OS Symbianu by však teoreticky mohly nastat menší problémy. Mezi jednotlivými edicemi pak zpětná kompatibilita zaručena není. Nemalé úpravy zdrojového kódu jsou nutné především při přechodu na S60 3rd Edition [3]. Problémy, které mohou s přesunem na novější edici vyvstat jsou alespoň dobře zdokumentovány [5].

V současnosti je nejvíce zařízení na platformě S60 2nd Edition. V budoucnosti lze ale očekávat nárůst počtu zařízení na platformě S60 3rd Edition. Téměř jisté však je, že i tato edice bude dříve či později překonána a zpětná kompatibilita s ní nebude zaručena.

Obecně se dá říci, že jednotlivé platformy nejsou vzájemně kompatibilní, tedy že aplikaci určenou pro jednu platformu není možné spustit na platformě jiné. Pro automatický převod aplikace z platformy S60 na platformu UIQ však existuje volně dostupné řešení v podobě nástroje, který umožňuje kompilování a spouštění aplikací pro S60 jakoby to byly nativní UIQ aplikace.

2.2.2 Platforma UIQ

UIQ je otevřená platforma vytvořená společností UIQ Technology, která je přímou odnoží a ve vlastnictví Symbian Ltd. Platforma je licencována následujícími výrobci mobilních zařízení: Arima, BenQ, Motorola a Sony Ericsson. Platforma UIQ byla původně vyvinutá pro zařízení ovládaná prostřednictvím dotykového displeje. Možná i díky tomu je platforma přítomna v mnohem méně zařízeních než je tomu v případě S60. Změnu by mohla přinést nová verze UIQ 3.0, která podporuje jak ovládání tlačítky tak přes dotykový displej.

V současných zařízeních je přítomna platforma UIQ ve verzi 2.0 (2002) a 2.1 (2003) nad OS Symbian v7.0 a 7.0s (viz. tab. B). SDK pro platformu je v těchto verzích distribuován přímo výrobci zařízení. Většina mobilních telefonů s těmito verzemi platformy již není v prodeji.

Platforma UIQ ve verzi 3.0 byla uvedena v roce 2005 a je primárně určena pro OS Symbian v9. Je vyvinuta tak, že by měla dát výrobcům mobilních zařízení možnost vytvořit celou škálu telefonů rozličných stylů bez toho, aniž by musela být platforma složitě přizpůsobována rozdílným požadavkům. Současně s touto verzí platformy vznikl UIQ Developer Program, který by měl nadále převzít podporu a služby pro vývojáře aplikací od výrobců zařízení. První telefon s UIQ 3 – Sony Ericsson M600 byl bohužel uveden teprve nedávno a stěží lze očekávat masové rozšíření této platformy v nejbližší době.

2.3 Vývojová prostředí pro S60

Aplikace na platformě S60 můžou být vytvářeny v jazyku C++ s využitím vývojového prostředí CodeWarrior® Development Studio for Symbian OS, Borland® C++BuilderX™ Mobile Edition, Microsoft Visual C++ 6.0, a Microsoft Visual Studio .NET 2003 [5].

Z uvedených prostředí je volně dostupný Borland C++ Mobile po vyplnění dotazníku na webu. Na vyzkoušení je také možné získat časově omezenou verzi CodeWarrior.

Nokia postupně nahrazuje doposud preferované a podporované prostředí CodeWarrior sadou nových nástrojů založených na prostředí Eclipse nazvaných Carbide.c++. K dispozici jsou celkem tři verze:

- **Carbide.c++ Express**
  Tato verze je dostupná zdarma a obsahuje všechny nezbytné nástroje pro vytváření aplikací na platformě S60 (2nd a 3rd Edition) a i na dalších platformách.

- **Carbide.c++ Developer Edition**
  Tato placená verze obsahuje nástroj pro grafický návrh aplikací a podporuje ladění aplikací přímo na cílovém zařízení.

- **Carbide.c++ Professional**
  Placená verze určená pro výrobce mobilních telefonů, která obsahuje všechny nástroje potřebné pro vývoj zařízení s OS Symbian.

2.4 Výběr verze S60

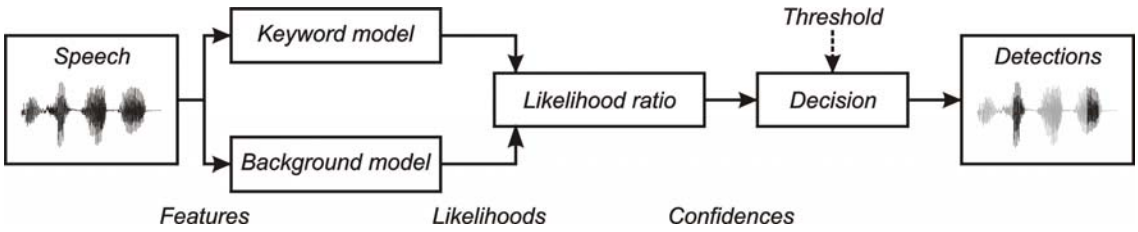
Vzhledem k uvedeným skutečnostem byla pro implementaci detektoru klíčových slov zvolena jako nejvhodnější verze S60 2nd Edition, která je v současnosti dostupná na největším počtu zařízení. Ty využívají OS Symbian minimálně ve verzi 7.0s a implementace zpracování vstupu z mikrofonu je tedy bez potíží možná prostřednictvím MMF. K dispozici je také zdarma vývojové prostředí Carbide.c++, které je vyvíjeno tvůrcem platformy.

Přes konkrétní stanovenou verzi platformy byla snaha, aby případný přenos na jinou verzi byl co možná nejjednodušší. Vzhledem k tomu, že je výsledná aplikace ve formě knihovny, nebylo nutné řešit uživatelské rozhraní, které je typicky oblast, která vyžaduje při přenosu na jiné platformy největší změny. Jediná část knihovny, která je závislá na platformě, je tak streaming audio vstupu z mikrofonu (viz. kap. 4.1.5).

3 Detektor klíčových slov

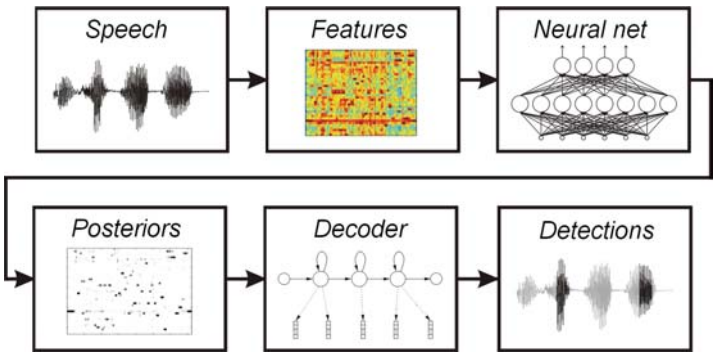
Detektor klíčových slov má za úkol vyhledat v řečovém signálu výskyt zadaných slov. Jakkoliv je specifikace problému jednoduchá, vlastní řešení již zdaleka tak přímočaré není. Na rozdíl od vyhledávání slov v textu, kde je řešení až na případné optimalizace triviální, musíme se při vyhledávání slov v mluvené řeči potýkat s mnohými problémy. Ty jsou spojené především s nejednoznačností a nepřesností. Signál odpovídající jednomu a témuž slovu se liší v závislosti na mnoha faktorech počínaje různými řečníky a konče vlivem okolního prostředí.

Úloha detekce klíčových slov je řešena v několika fázích, které postupně adresují dílčí problémy zpracování řeči (Obr. 3.1). Prvním krokem zpracování vstupního signálu (navzorkovaného a rozděleného na rámce) je extrakce příznaků (features), které reprezentují rysy signálu, které jsou významné pro rozpoznávání. Vektory příznaků odpovídající jednotlivým rámcům se dále klasifikují a určuje se pro ně, s jakou věrohodností (likelihood) odpovídají zadaným modelům. Věrohodnost se určí pro model úseku řeči obsahující klíčové slovo a pro model stejného úseku řeči neobsahující klíčové slovo. Výstupem je poměr těchto dvou věrohodností. Pokud tento poměr překročí stanovený práh, je odpovídající klíčové slovo detekováno.



Obr. 3.1: Obecné schéma detektoru klíčových slov (převzato z [7])

Popsané obecné schéma detektoru zachycuje především jeho poslední část – dekodér. Konkrétních přístupů řešení problému detekce klíčových slov, které se vzájemně liší nejen dekodérem ale i jinými částmi, bylo do současnosti navrženo již několik [7]. Způsob řešení zvolený pro implementaci je založený na extrakci příznaků pomocí kepstrální analýzy resp. perceptuální lineární predikce, klasifikaci příznaků na jednotlivé fonémy s určitou mírou pravděpodobnosti a detekci klíčových slov pomocí dekodéru a rozpoznávací sítě. Základní schéma implementovaného detektoru je níže na obr. 3.2.



Obr. 3.2: Schéma implementovaného detektoru klíčových slov

V následujících podkapitolách jsou podrobněji popsány jednotlivé části detektoru od vstupního řečového signálu až po detekci klíčového slova.

3.1 Extrakce příznaků

První část detektoru slouží pro extrakci příznaků ze vstupního řečového signálu. Tato část přizpůsobuje signál pro další automatické zpracování.

Příznaky by měly reprezentovat význačné rysy řečového signálu. Měly by pokud možno zachycovat takové charakteristiky, které jsou důležité pro vnímání a pochopení významu projeveného. Naopak by měly potlačovat šum a individuální charakteristiky mluvčího. Typicky např. požadujeme, aby příznaky nebyly závislé na frekvenci základního tónu. Na příznaky také klademe požadavek, aby byly dekorelované. Základním prostředkem pro zajištění výše uvedeného je výpočet spektra.

Metod pro získávání příznaků existuje hned několik [8]. Pro implementaci byly vybrány dvě z nich – metoda založená na kepstrální analýze s delším časovým kontextem a metoda perceptuální lineární predikce (PLP). Obě metody byly testovány a do výsledného detektoru byla nakonec zvolena ta, která se ukázala být vhodnější pro praktickou realizaci (viz. kap. 4.2.2). Dále je podrobněji popsán postup získání příznaků u obou metod. Nejdříve jsou však uvedeny jednoduché transformace a operace nad řečovým signálem, které slouží pro základní předzpracování signálu a můžou být volitelně použity u libovolné extrakce příznaků tedy i u obou zmíněných metod.

3.1.1 Předzpracování signálu

Základem pro digitální zpracování signálu je jeho vzorkování, tedy sběr hodnot spojité veličiny v daných diskrétních okamžicích. Vzhledem k tomu, že zpracováváme signál na číslicových strojích je tento předpoklad samozřejmý a nemusíme se jím přímo zabývat. Řečový signál je nejčastěji vzorkován o frekvenci 8000 nebo 16000 Hz. Veškeré metody popsané dále zpracovávají signál po rámcích. Rámec je vektor vzorků dané délky. Často se volí rámec s počtem vzorků odpovídajícím 25 ms promluvy. (Tedy s 200 vzorky při vzorkovací frekvenci 8 kHz.) Pro účely rozpoznávání řeči se navíc rámce překrývají. Běžně používaným intervalem mezi dvěma po sobě jdoucími rámci je 10 ms.

Jednotlivé transformace signálu jsou níže uvedeny v pořadí, v jakém jsou obyčejně prováděny. Transformace nejsou pro další zpracování nezbytně nutné a kterákoliv z nich (případně i všechny) může být vynechána. Popsané transformace však často pozitivně ovlivňují kvalitu výsledných příznaků a tak je jejich použití doporučeníhodné.

První transformace spočívá v odečtení průměrné hodnoty vzorku v rámci od každého jednotlivého vzorku (1). (Konstanta N v rovnici označuje délku rámce.) Tímto je odstraněna stejnosměrná složka, která bývá přidána k signálu při jeho konverzi z analogového na číslicový.

$$s'_n = s_n - \frac{1}{N}\sum_{i=0}^{N-1} s_i \tag{1}$$

Následně se často taktéž na každý vzorek rámce aplikuje preemfáze. Ta je popsána diferenční rovnicí prvního řádu (2). Koeficient k se volí v intervalu od 0 do 1.

$$s'_n = s_n - k s_{n-1} \tag{2}$$

Další transformací vstupního řečového rámce je jeho vážení Hammingovým oknem (3), což utlumí signál na okrajích. Ponechání původního signálu beze změny by později vedlo ke zkreslení spektra, jehož výpočet tvoří základ pro většinu metod extrakce příznaků.

$$w_n = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) \tag{3}$$

Poslední rovnice, která bude nyní uvedena, nepředstavuje transformaci signálu, ale její výsledek slouží jako jednoduchá charakteristika signálu. Jedná se o výpočet energie (4), která se často přidává do vektoru příznaků. Energie se počítá buďto přímo ze vstupního rámce nebo po předchozí preemfázi. Někdy se také bere v úvahu její logaritmus.

$$E = \sum_{i=0}^{N-1} s_i^{\,2} \tag{4}$$

3.1.2 Kepstrální příznaky

Metoda byla inspirovaná lidským sluchovým ústrojím. Modeluje ho pomocí sady filtrů typu pásmová propust. Níže jsou popsány jednotlivé kroky zpracování od vstupního signálu (na který byly aplikovány výše uvedené transformace) až po výsledný vektor příznaků.

Prvním krokem je výpočet spektra pomocí diskrétní Fourierové transformace (DFT). DFT pracuje v oboru komplexních čísel a tak převedeme do tohoto oboru i hodnoty vzorků rámce nastavením jejich komplexní složky na nulu. Během DFT je tedy postupně převedeno N komplexních vzorků $x_0, \ldots, x_{N-1}$ vstupního signálu na N komplexních vzorků $X_0, \ldots, X_{N-1}$ spektra (5). DFT slouží pro transformaci signálu z časové oblasti do frekvenční.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \tag{5}$$

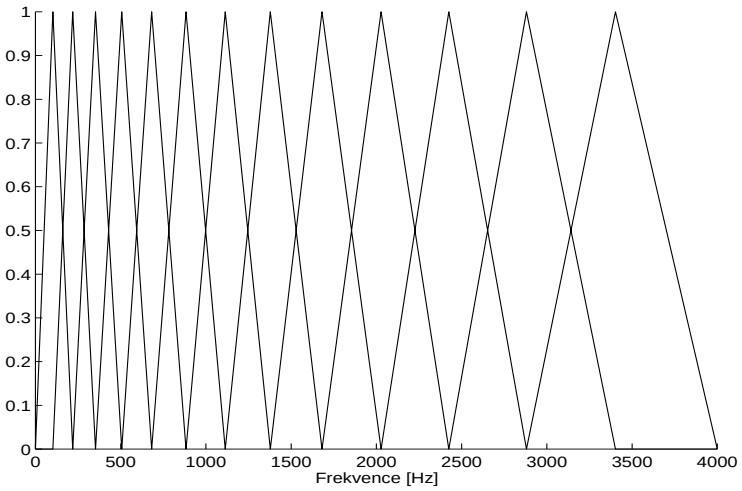
Z komplexních hodnot výsledného spektra následně spočítáme druhé mocniny jejich absolutních hodnot (power spectrum) (6).

$$S_k = (real(X_k))^2 + (imag(X_k))^2 \tag{6}$$

Nyní můžeme přistoupit ke konstrukci filtrů. Filtry vychází z experimentů, jak lidé vnímají frekvence v různé části spektra. Filtry trojúhelníkového tvaru se rovnoměrně umístí po frekvenční ose. Jednotkou však není hertz, ale mel. Způsobů vyjádření jejich závislosti se s ohledem na to, že se jedná o experimentálně získané údaje, používá několik. V implementaci detektoru je pro převod hertzů na mely použit vztah (7). Závislost, kterou vyjadřuje, je do 1 kHz zhruba lineární, nad tuto hodnotu pak logaritmická.

$$mel(f) = 2595\log\left(1 + \frac{f}{100}\right) \tag{7}$$

Sousední filtry se vždy překrývají o polovinu. Zobrazíme-li filtry na frekvenční ose v hertzích, uvidíme, že se jejich šířky se vzrůstající frekvencí zvětšují (Obr. 3.3). Tím je odstraněna jemná struktura spektra odpovídající základnímu tónu. Počet filtrů a interval, na kterém jsou rozmístěny je vhodně zvolen podle potřeby (viz. kap. 4). Výška trojúhelníkových filtrů je jednotně nastavena na jedna.



Obr. 3.3: Rozmístění filtrů na lineární frekvenční ose

Nyní, když máme připravené spektrum a sadu filtrů, můžeme vypočítat mel spektrum. To se spočítá postupným násobením spektra všemi filtry a integrováním výsledku (8). *N* je počet hodnot spektra, *L* je počet filtrů, M^l je l-tý filtr. Výsledné koeficienty mel spektra jsou poté ještě logaritmovány. To je v souladu s logaritmickým vnímáním hlasitosti lidmi.

$$\widetilde{S}_l = \sum_{k=0}^{N/2} S_k M_k^l \qquad l = 0,1,...,L-1 \tag{8}$$

Posledním krokem je výpočet DCT podle vztahu (9), kde m_j je j-tý koeficient mel spektra a *N* je počet koeficientů (filtrů). Použitím DCT dojde k dekorelaci koeficientů. DCT není aplikována pouze na jeden vektor mel spektra, ale přes delší časový kontext (viz. kap. 4.2.1). Takto vzniknuvší vektory jsou před výpočtem DCT opět váhovány Hammingovým oknem.

$$c_i = \sum_{j=1}^{N} m_j \cos\left(\frac{\pi i}{N}\left(j-0.5\right)\right) \tag{9}$$

Počet koeficientů výsledného vektoru je ještě možné redukovat oproti délce časového kontextu. Výsledný vektor příznaků, který pak vznikne spojením vektorů koeficientů vypočítaných v rámci jednoho časového kontextu, má pak méně dimenzí a zjednodušuje následnou klasifikaci.

3.1.3 Perceptuální lineární predikce

Metoda vychází z lineární predikce (LP) [8]. Koeficienty LP jsou váženy rovnoměrně přes celé spektrum. Tento přístup není v souladu s funkcí lidského sluchového ústrojí. Metoda perceptuální lineární predikce (PLP) [12,13] proto zahrnuje do výpočtu koeficientů poznatky o vnímání (odtud perceptuální) frekvence v různé části spektra lidmi, podobně jako v předchozí metodě. Níže jsou popsány jednotlivé kroky zpracování signálu pro získání PLP koeficientů.

Prvním krokem je výpočet spektra pro daný řečový rámec. Postup je totožný jako v minulé kapitole. Nejdříve je signál váhován Hammingovým oknem (3) a poté je spočítána DFT. Výsledkem tohoto kroku je power spectrum (6).

Dalším krokem je násobení spektra filtry a integrování podobně jako v případě předchozí metody. U perceptuální lineární predikce se však spíše používají lichoběžníkové filtry než trojúhleníkové. Filtry jsou rozmístěny rovnoměrně po frekvenční ose po intervalech o délce zhruba

1 Bark. Jednotka Bark je opět vztažena k Hz nelineárně. V tomto případě je převodní funkce popsána vztahem (10), kde ω je frekvence v rad/s.

$$\Omega(\omega) = 6\ln\left\{\frac{\omega}{1200\pi} + \left[\left(\frac{\omega}{1200\pi}\right)^2 + 1\right]^{0.5}\right\} \tag{10}$$

Dále následuje preemfáze spektra, která vyváží nerovnoměrné vnímaní hlasitosti na různých frekvencích. Tímto už se PLP liší od předchozí metody, kde se sice preemfáze také používá, ale již před spektrální analýzou a podle vztahu (2). Preemfáze spektra u PLP je provedena jeho násobením s koeficienty danými předpisem (11) (equal-loudness curve).

$$E(\omega) = \frac{(\omega^2 + 1.44\times10^6)\omega^4}{(\omega^2 + 9.61\times10^6)(\omega^2 + 1.6\times10^5)^2} \tag{11}$$

Po integraci a preemfázi se koeficienty v předchozí metodě logaritmovaly. V případě PLP je použita místo logaritmu třetí odmocnina, která vyjadřuje vztah mezi vnímáním hlasitosti lidmi a intenzitou signálu (12). Tento vztah sice příliš neplatí pro velmi hlasité nebo naopak velmi tiché zvuky, ale jako aproximace v případě řeči je vyhovující.

$$L(\omega) = \sqrt[3]{I(\omega)} \tag{12}$$

Dalším krokem výpočtu PLP koeficientů je aplikování inverzní diskrétní Fourierové transformace (IDFT) (13).

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N}kn} \tag{13}$$

Výsledek je ekvivalentní autokorelačním koeficientům. Z autokorelačních koeficientů jsou následně pomocí řešení soustavy lineárních rovnic spočítány PLP koeficienty. Pro řešení může být využita efektivní metoda Levinson-Durbin. PLP koeficienty můžou být poté volitelně transformovány na kepstrální koeficienty. Kepstrum signálu může být získáno pomocí Fourierovy transformace PLP spektra. To obdržíme z Fourierové transformace koeficientů filtrů. Tento postup je poměrně složitý, naštěstí existuje mnohem efektivnější způsob výpočtu kepstra [16]. Ten spočívá v použití jednoduché rekurze (14). V rovnici a_i značí vstupní i-tý PLP koeficient.

$$c_n = -a_n - \frac{1}{n}\sum_{i=1}^{n-1}(n-i)a_i c_{n-i} \tag{14}$$

Počet takto získaných koeficientů nemusí být nutně shodný s počtem PLP koeficientů. Výhodou kepstrálních koeficientů je, že jsou obecně dekorelovány. Menší problém však může činit fakt, že koeficienty vyššího řádu bývají značně malé a rozdíly mezi jednotlivými koeficienty jsou tak poměrně velké. Z důvodů přesnosti při dalším zpracování a dalších praktických důvodů jsou proto koeficienty váženy (liftrovány) pomocí okna (15), aby byly přibližně stejného řádu.

$$c'_n = \left(1 + \frac{L}{2}\sin\frac{\pi n}{L}\right)c_n \tag{15}$$

Perceptuální lineární predikce je založena na krátkodobém spektru řeči. Počet PLP koeficientů se často volí poměrně malý (např. 12). Pro zvětšení kvality rozpoznávání řeči se mohou do vektoru

příznaků přidat derivace původních koeficientů podle času. Navíc je možné přidávat koeficienty derivací vyšších řádů a díky tomu zvolit výsledný vektor příznaků tak, aby nejlépe vyhovoval daným účelům. Koeficienty získané první derivací se nazývají delta koeficienty, koeficienty získané druhou derivací se nazývají double delta (nebo také akcelerační) koeficienty. Někdy se ještě používají také koeficienty získané třetí derivací. Vždy platí, že pokud jsou ve vektoru příznaků použity koeficienty derivace řádu k, jsou také použity koeficienty derivace řádu $k–1$. Pro testovací účely byla implementována podpora pro výpočet delta a double delta koeficientů (volba příznaků, které byly nakonec vybrány pro finální detektor, je uvedena v kap. 4.2.2).

Delta koeficienty se vypočítají z PLP koeficientů podle vzorce (16). Delta koeficient d pro čas t závisí na Θ statických koeficientech c, které předchází času t a Θ statických koeficientech, které následují po čase t. Volitelná hodnota parametru Θ (také nazývaná velikost okna) tedy určí kolik rámců do minulosti resp. do budoucnosti se pro výpočet použije.

$$d_t = \frac{\sum_{\theta=1}^{\Theta} \theta\left(c_{t+\theta} - c_{t-\theta}\right)}{2\sum_{\theta=1}^{\Theta} \theta^2} \tag{16}$$

Double delta koeficienty se spočítají podle též formule. Jediným rozdílem je, že se nepočítají ze statických koeficientů nýbrž z delta koeficientů. Mírným problémem výpočtu delta resp. akceleračních koeficientů je neexistence statických resp. delta rámců pro výpočet v čase $t = 0$. Tento problém startu metody se řeší zopakováním prvního rámce tak, aby se naplnila levá část okna.

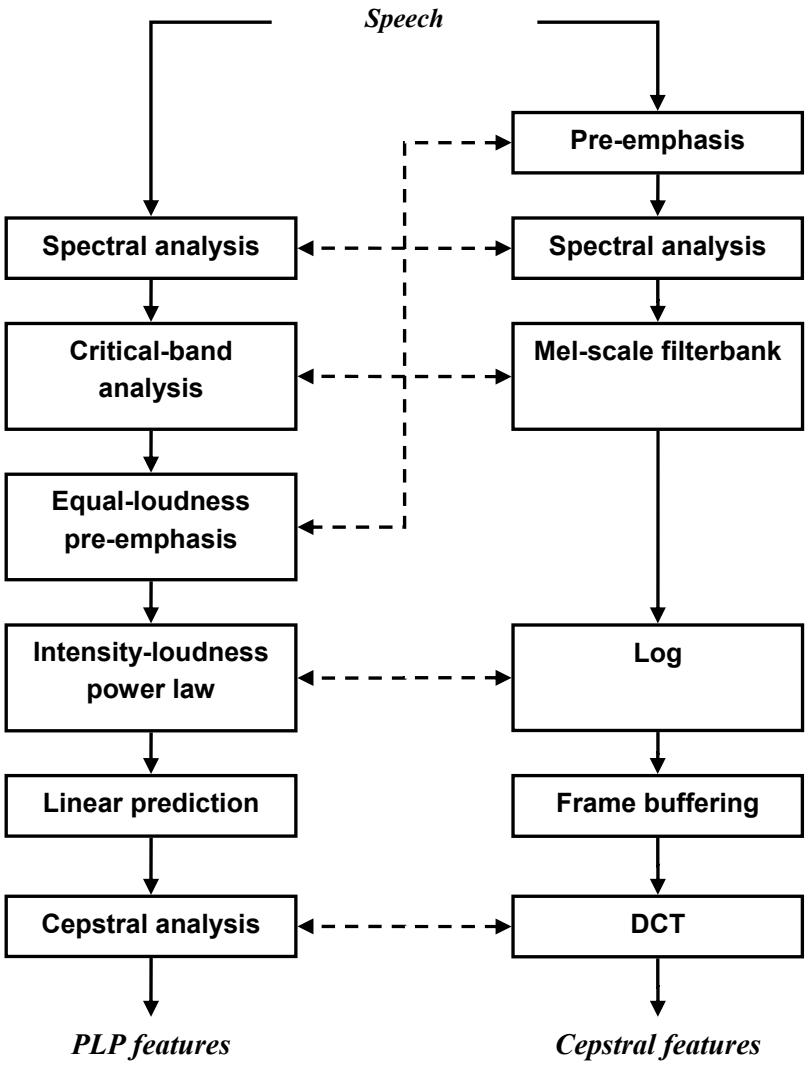
Pro některé účely může být vhodné použít pro výpočet dynamických koeficientů pouze koncové rámce okna. V takovém případě se použije vztah (17).

$$d_t = \frac{\left(c_{t+\Theta} - c_{t-\Theta}\right)}{2\Theta} \tag{17}$$

Závěrem této kapitoly budou shrnuty rozdíly mezi oběma popsanými metodami extrakce příznaků. Jak je vidět z obrázku 3.4, v obou metodách se dají najít části zpracování signálu, které si vzájemně odpovídají. Obě metody převádí vstupní signál na spektrum pomocí DFT. Obě metody také využívají poznatky o sluchovém ústrojí člověka a modelují ho pomocí filtrů. Metoda PLP jde v tomto ohledu dál a uplatňuje tyto poznatky i v případě preemfáze a aplikace experimentálně zjištěného vztahu mezi intenzitou a hlasitostí. V obou případech však můžeme identifikovat části zpracování u kepstrálních příznaků, které mají podobný účel.

Zvláštností popisovaných kepstrálních příznaků je aplikace DCT přes delší časový kontext (ilustrováno polem frame buffering v obrázku). Toto rozšíření zvyšuje přesnost rozpoznávání ne nepodobně jako je tomu při použití delta a akceleračních koeficientů u metody PLP. Toto rozšíření má však za následek značný nárůst velikosti výsledného vektoru příznaků a ovlivňuje zásadním způsobem časovou složitost celého procesu rozpoznávání. To může být kritické zvláště s ohledem na implementaci do mobilního telefonu s omezenými výpočetními prostředky.

Co se týče kvality příznaků pro rozpoznávání, dají se vzhledem k použití delšího časového kontextu očekávat lepší výsledky u metody kepstrálních příznaků. Skutečný rozdíl by však bylo nutné vyhodnotit statisticky na daném vzorku dat.

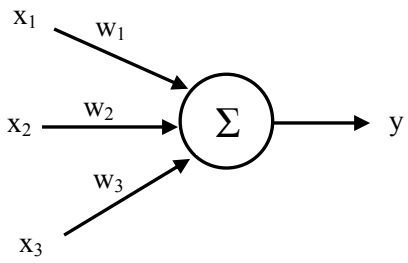


Obr. 3.4: Vztah výpočtu PLP a kepstrálních příznaků, převzato a upraveno z [8]

3.2 Klasifikátor – neuronová síť

Další částí detektoru klíčových slov, která následuje po extrakci příznaků, je klasifikátor. Klasifikátor slouží pro mapování příznaků na fonémy. Klasifikátorů, které můžou být použity pro účely detektoru, existuje hned několik. Jsou to např. neuronové sítě, skryté Markovovy modely (HMM), Support Vector Machines (SVM) nebo rozhodovací stromy. Pro implementaci detektoru byly vybrány neuronové sítě. Níže bude stručně popsán základní princip neuronových sítí. Detailně se problematice věnuje např. [10,11].

Neuronová síť je výpočetní model, jehož činnost je inspirována funkcí lidského mozku. Síť je tvořena určitým množstvím vzájemně propojených atomických jednotek – perceptronů. Perceptron je velice zjednodušeným matematickým modelem lidského neuronu. Perceptron má vždy vektor vstupů a právě jeden výstup (viz. obr. 3.5).

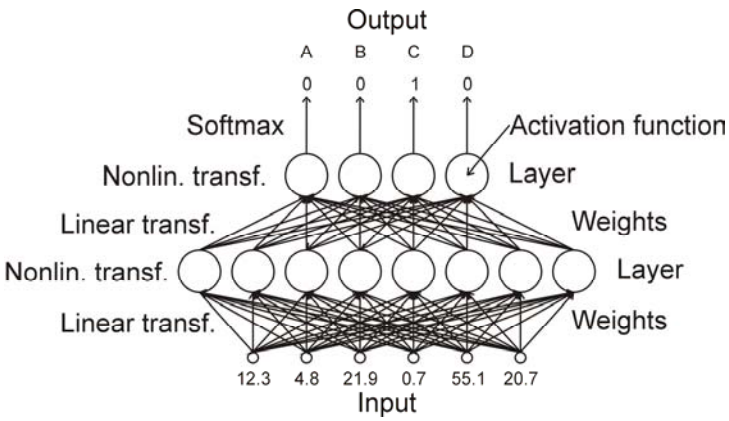


Obr. 3.5: Perceptron – umělý neuron

Činnost perceptronu je popsána funkcí (18), která transformuje vstupní vektor na výstupní hodnotu. Proměnné ve vztahu mají následující význam: x_i je hodnota i-tého vstupu, w_i je odpovídající váha, N je počet vstupů, b je práh a f je aktivační funkce (viz. dále).

$$y = f\left(\sum_{i=1}^{N} w_i x_i + b\right) \tag{18}$$

Perceptron sám o sobě má schopnost klasifikovat vstupní vektor do dvou tříd. Může řešit pouze lineárně separabilní problém. Ve dvourozměrném prostoru je tedy hranicí mezi třídami přímka. Klasifikátor detektoru však musí rozlišovat do tříd daných počtem fonémů, tedy více než do dvou. Abychom zvýšili rozlišovací schopnost perceptronu, vytváříme jejich spojením neuronové sítě. Typů neuronových sítí je celá řada. Dělí se podle architektury, způsobu učení, aplikace i dalších kritérií. Pro účely klasifikace je jednou z nejpoužívanějších sítí třívrstvá (resp. dvouvrstvá pokud mezi vrstvy nepočítáme vstup) dopředná neuronová síť (Obr. 3.6), která je využita i v implementaci detektoru. Tato síť je již schopná dělit prostor libovolnou nelineární funkcí.

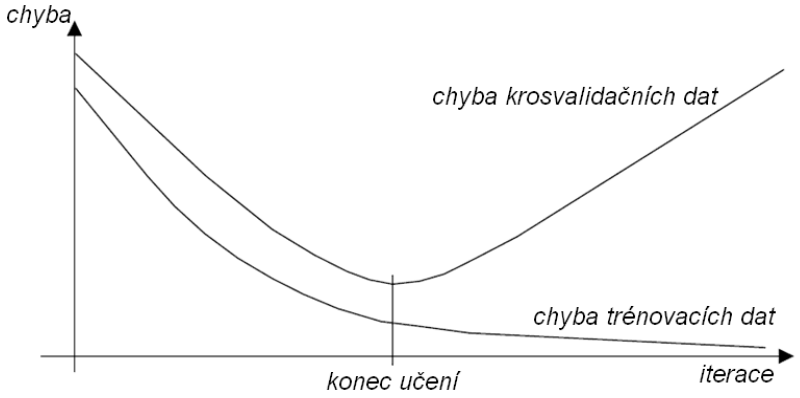


Obr. 3.6: Třívrstvá dopředná neuronová síť (převzato z [15])

Aby síť řešila požadovaný problém, je nutné ji nejdříve natrénovat. V průběhu trénování se nastaví hodnoty vah a prahů jednotlivých perceptronů. K tomu je zapotřebí trénovací sada vstupních vektorů, ke kterým známe odpovídající výstupní vektory. Pro trénování dopředných neuronových sítí se používá algoritmus Backpropagation. Jedná se o iterativní vyhodnocování odezvy sítě a úpravy vah na základě zpětného šíření chyby. Algoritmus by se dal zjednodušeně popsat následujícími kroky:

1. Inicializuj váhy a prahy sítě náhodnými hodnotami
2. Proveď dopředný průchod sítí
3. Vypočti chybu sítě (rozdíl požadované hodnoty od obdržené)
4. Propaguj chybu zpět sítí a upravuj jednotlivé váhy a prahy

V průběhu trénování se neustále snižuje chyba odezvy sítě na trénovací vektory. Proces trénování je však nutné ve správnou chvíli ukončit, jinak hrozí přetrénování sítě. Síť je přetrénovaná ve chvíli, kdy ztrácí schopnost žádané generalizace a umí klasifikovat jen vektory z trénovací množiny (byť třeba dokonale). K detekci vhodného okamžiku pro ukončení učení se používá krosvalidační sada vektorů. Jedná se o vektory, které jsou z procesu učení vyjmuty a slouží pouze pro srovnání chyby odezvy sítě. Jakmile se chyba sítě pro krosvalidační vektory začne zvětšovat, je potřeba trénování ukončit (viz. obr. 3.7).



Obr. 3.7: Vývoj chyby sítě v průběhu trénování

Algoritmus trénování sítě nebylo potřeba implementovat. K natrénování sítě pro detektor byl použit dostupný specializovaný nástroj (viz. kap. 4.3.3). Pro úspěšné natrénování sítě a pro zajištění její úspěšné klasifikace je však potřeba v nástroji pro trénování nastavit její rozměry, tedy počet neuronů první (skryté) vrstvy. Počet neuronů by neměl být příliš malý, aby byla síť schopna klasifikovat všechny požadované třídy. Naopak počet neuronů by neměl být ani příliš velký, aby nedošlo ke špatné generalizaci. Správný počet je potřeba experimentálně odhadnout nebo využít zkušeností jiných při řešení daného problému klasifikace fonémů na základě vektoru příznaků jisté délky.

Implementace detektoru zajišťuje pouze vyhodnocení sítě pro dané vstupy. Jedná se o algoritmus Forward pass popsaný níže.

Vstupem neuronové sítě detektoru klíčových slov bude vektor příznaků. V našem případě to bude buď vektor PLP koeficientů nebo vektor kepstrálních koeficientů.

Vstupní vektor příznaků se v případě kepstrálních koeficientů nejdříve normalizuje podle vztahu (19). Vektor středních hodnot μ a vektor směrodatných odchylek σ ve vztahu se vypočítá z trénovací sady pro učení neuronové sítě. Normalizací se transformuje střední hodnota přibližně na hodnotu 0 a odchylka zhruba na hodnotu 1.

$$y = \frac{x - \mu}{\sigma} \tag{19}$$

Účelem normalizace je modifikovat rozsah vstupních dat tak, aby lépe odpovídal výstupním hodnotám. Ty jsou v případě pravděpodobností fonémů v rozmezí od 0 do 1. Tato nenáročná operace může často zdokonalit výsledky predikce.

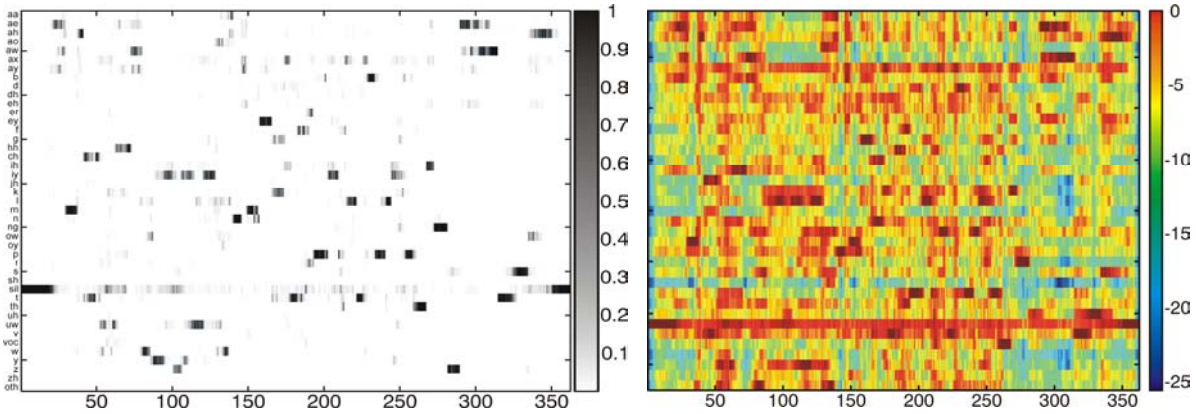
Dalším krokem vyhodnocení sítě je výpočet výstupních hodnot perceptronů v první vrstvě podle vztahu (18). Aktivační funkcí je v tomto případě sigmoida (20). Ta má podobný průběh jako skoková funkce, ale na rozdíl od ní je derivovatelná, což je potřeba pro trénovací algoritmus Backpropagation.

$$y = \frac{1}{1 + e^{-x}} \tag{20}$$

Výstup perceptronů první vrstvy je vstupem perceptronů druhé vrstvy. Jejich počet je roven počtu fonémů. Na této vrstvě se již nepoužije jako aktivační funkce sigmoida. Místo toho se použije funkce softmax (21).

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{M} e^{x_j}} \tag{21}$$

Softmax upraví rozsah výstupních hodnot od 0 do 1 tak, aby byl jejich součet roven jedné. Výstupem sítě i algoritmu Forward pass pak bude vektor pravděpodobností jednotlivých fonémů. Příklad výstupních vektorů pro několik desítek po sobě jdoucích rámců je uveden na obrázku 3.8.

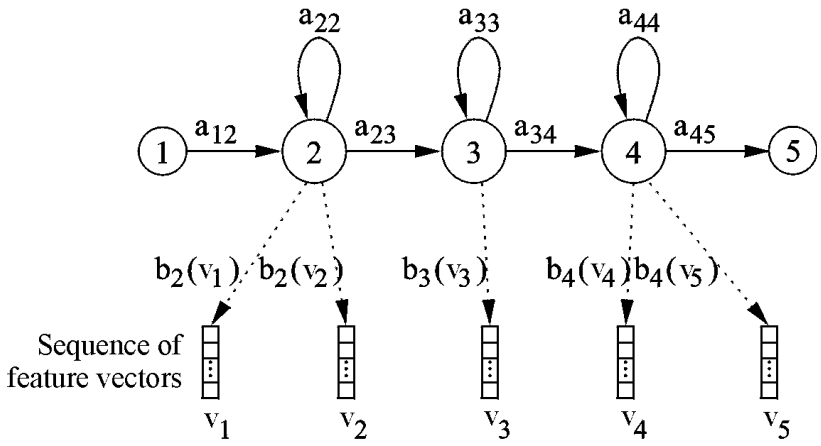


Obr. 3.8: Výstup klasifikátoru - matice posteriorních pravděpodobností (vlevo výstup neuronové sítě, vpravo výstup po zlogaritmování) (převzato z [15])

3.3 Dekodér

Mohlo by se zdát, že výstup neuronové sítě je již vlastně výstupem detektoru. Naivní řešení detekce by spočívalo v pokusu o vyhledání klíčových slov v řetězci konstruovaném z nejpravděpodobnějších fonémů tak, jak je určila neuronová síť. Problémem tohoto řešení je již fakt, že jednomu fonému nemusí odpovídat jeden výstupní vektor sítě. Počet těchto vektorů navíc není vždy konstantní, jedno a totéž slovo není nikdy řečeno za stejnou dobu. Další problém by vznikal také díky nepřesnostem rozpoznávání a tudíž striktní porovnávání řetězců není možné. Je potřeba uplatnit pravděpodobnostní model, který je robustní vůči různým délkám týchž slov. Těmto podmínkám vyhovuje tzv. skrytý Markovův model (HMM).

Příklad HMM je zobrazen na obrázku 3.9. Nejdříve bude popsáno, jak se model vytváří a následně, jak se s ním pracuje. Model se skládá ze stavů (označené čísly), které jsou propojovány různým (ale ne libovolným) způsobem orientovanými hranami.



Obr. 3.9: Příklad skrytého Markovova modelu

Se stavy je asociována funkce rozložení pravděpodobnosti. Ta obecně určuje věrohodnost (likelihood) vyslání daného vektoru příznaků stavem. Tato funkce je tedy závislá na čase, protože pro každý rámec máme nový vektor příznaků. V našem případě vysílací pravděpodobnosti stavů odpovídají posterior pravděpodobnostem, které nám produkuje neuronová síť. Stavy v HMM reprezentují obecně určitý úsek promluvy. V implementaci detektoru odpovídá jeden stav vždy jednomu fonému, ale obecně by to mohly být i větší či menší jednotky. Model znázorněný na obrázku 3.9 by tedy v našem případě odpovídal slovu složenému ze tří fonémů (stav označený 1 resp. 5 je počáteční resp. koncový a slouží jen pro spojování modelů).

Orientované hrany v HMM určují možné přechody mezi stavy. Na obrázku 3.9 jsou stavy propojeny způsobem, který umožňuje pro daný stav *k* v tomto stavu setrvat nebo přejít do stavu *k+1* nebo do stavu *k+2*, pokud tyto existují. V praxi se běžně používají jen zpětné smyčky a přechody do bezprostředně následujících stavů. Ne jinak je tomu i v implementovaném detektoru, tam jinými slovy platí, že je možné zůstat „v“ daném fonému nebo se posunout na další. Hrany jdoucí zpět nejsou přípustné. Hrany jsou ohodnocené tzv. přechodovými pravděpodobnostmi. Ty určují pravděpodobnost přechodu z jednoho stavu do druhého. Vždy musí platit, že součet přechodových pravděpodobností všech hran vycházejících z daného uzlu je roven jedné. V detektoru je navíc pravděpodobnost přechodu do následujícího stavu stejná jako pravděpodobnost setrvání v aktuálním. Nerovnoměrným rozložením přechodových pravděpodobností lze docílit v rozpoznávačích např. uplatnění jazykového modelu. Ten však není v implementaci detektoru uvažován.

HMM nám umožní pro každý vektor posteriorních pravděpodobností (výstup neuronové sítě) určit aktuální hodnotu věrohodnosti výskytu slova, které modeluje. Věrohodnost slova odpovídá optimální cestě v modelu. Každá cesta je ohodnocena kumulovanou věrohodností. Ta se získá násobením přechodové pravděpodobnosti s vysílacími pravděpodobnostmi, které se po cestě vyskytují. Optimální cesta je cesta s maximální koncovou hodnotou věrohodnosti. Zjišťování věrohodností všech cest, abychom mohli vybrat optimální, je však časově náročné ne-li nemožné. Proto se používá algoritmus, který nalezne optimální cestu mnohem efektivněji. Jedná se o Viterbiho dekodér.

Viterbiho dekodér je založen na definici částečné likelihood, jako věrohodnosti nejlepší cesty končící v daném stavu pro daný čas. Rozhodnutí o nejlepší cestě se aktualizuje pro každý čas. Zjednodušeně řečeno se pro každý stav vyhodnotí nová částečná likelihood jako maximum částečných věrohodností pro předchozí čas, které jsou na stav napojeny. Přesněji je jádro algoritmu posáno vztahem (22). Ve vztahu Φ_i označuje částečnou Viterbiho likelihood stavu *i*, a_{ij} je přechodová pravděpodobnost ze stavu *i* do stavu *j*, b_j je vysílací pravděpodobnost stavu *j* a *t* je čas.

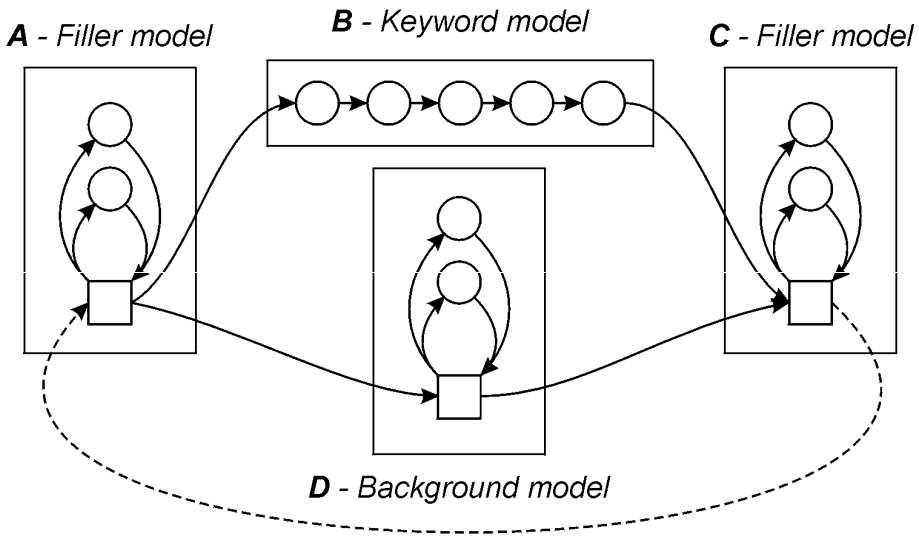
$$\Phi_j(t) = \max_i \{\Phi_i(t-1)a_{ij}\}b_j(t) \tag{22}$$

Částečná Viterbiho likelihood v posledním stavu pak odpovídá věrohodnosti optimální cesty a tedy i celého slova. Pro praktickou realizaci je účelné reprezentovat vysílací a přechodové pravděpodobnosti jejich logaritmy. To je především z důvodů zachování přesnosti a také díky tomu operace násobení v algoritmu přejde na operaci sčítání, která je rychlejší.

Poslední problém, který je potřeba v souvislosti s dekódováním řešit je nastavení prahu detekce klíčových slov. Tento problém není až tak triviální protože likelihood, která je výstupem Viterbiho dekodéru závisí jednak na délce klíčového slova (tedy počtu stavů) a jednak na konkrétním rozložení pravděpodobností jednotlivých fonémů v dané promluvě. Abychom mohli nastavovat práh jednotným způsobem pro všechna slova nezávisle na zmíněných proměnných je potřeba do procesu dekódování zavést určitou normalizaci. V následujících podkapitolách jsou popsány dva různé přístupy k řešení tohoto problému. Implementovány byly oba dva, po experimentech s nimi byl do výsledného detektoru vybrán jeden z nich (viz. kap. 4.3.4).

3.3.1 Síť modelů slov a modelu pozadí

Klasickým přístupem pro řešení detekce klíčových slov je použití Viterbiho dekodéru uvedeného výše pracujícího nad sítí modelů. Obecná síť je zobrazena na obrázku 3.10. Skládá se z několika vzájemně propojených skrytých Markovových modelů, které slouží k různému účelu.



Obr. 3.10: Síť modelů pro detekci klíčového slova (převzato a upraveno z [15])

Část označená v obrázku *A* resp. *C* slouží pro modelování části promluvy bezprostředně před klíčovým slovem resp. po klíčovém slovu. Tyto modely představují smyčky všech fonémů. Část *B* modeluje klíčové slovo a tvoří ji stavy odpovídající jednotlivým fonémům slova. Konečně část *D* je model pozadí pro odpovídající část promluvy jako model klíčového slova. Model pozadí ale stejně jako *A* a *C* neklade na obsah promluvy žádná omezení a je tvořen fonémovou smyčkou. Znamená to tedy, že jeho likelihood bude maximální.

Obrázek zachycuje pro ilustraci síť pouze s jedním klíčovým slovem. Modely dalších klíčových slov by byly analogicky přidány mezi části *A* a *C*. Popsaná síť může být bez modifikací použita v offline režimu detekce klíčových slov. Pro použití v online režimu, se kterým počítá implementace, je zapotřebí ji mírně upravit (viz. kap. 4.3.4). Princip ale zůstává nezměněn.

Výsledná věrohodnost výskytu klíčového slova v promluvě L_{KW} je dána podílem likelihood cesty modelem obsahujícím klíčové slovo L_{ABC} a likelihood cesty modelem neobsahujícím klíčové slovo L_{ADC} (23).

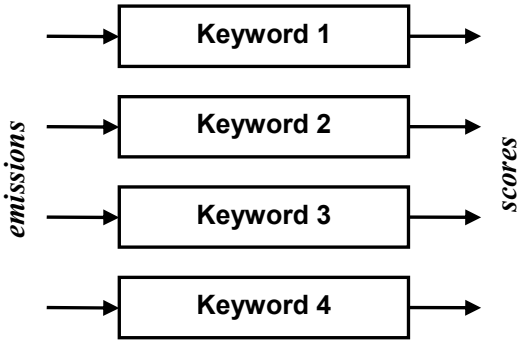
$$L_{KW} = \frac{L_{ABC}}{L_{ADC}} \tag{23}$$

Tímto vztahem je vyjádřena normalizace zmiňovaná v úvodu kapitoly. Pro detekci libovolného klíčového slova stačí nastavit jedinou hodnotu prahu. Samotné nastavení prahu ovšem také není jednoduché, jedná se v podstatě o optimalizační problém.

Práh by měl být pokud možno nastaven tak, aby maximalizoval počet správných případů (přijme klíčové slovo, pokud mělo být detekováno nebo zamítne klíčové slovo, pokud nemělo být detekováno) a tím minimalizoval počet špatných případů (zamítne klíčové slovo, přestože mělo být detekováno nebo přijme klíčové slovo, přestože nemělo být detekováno).

3.3.2 Modifikovaný Viterbiho dekodér

Alternativním přístupem pro detekci klíčových slov je použití modifikovaného Viterbiho dekodéru [14, 20]. Tento způsob detekce klíčových slov narozdíl od minulého přístupu nevyužívá síť modelů, ale používá jednotlivé modely zcela odděleně (viz. obr. 3.11).



Obr. 3.11: Organizace modelů pro modifikovaný Viterbiho dekodér

Každý model klíčového slova je tvořen opět lineárním propojením stavů, které odpovídají jednotlivým fonémům. Žádný model pozadí ani jiné modely kromě klíčových slov tento dekodér nepoužívá. První stav modelu klíčového slova tedy nemá žádného předchůdce a poslední stav nemá žádného následníka. Věrohodnost výskytu klíčového slova se určí pro každý čas na základě hodnoty likelihood (u tohoto algoritmu nazývané skóre) posledního stavu odpovídajícího modelu.

Aby nebylo nutné používat model pozadí, jsou explicitně do původního Viterbiho algoritmu zakomponovány dvě normalizace. První normalizace se týká vysílací pravděpodobnosti. Normalizovaná vysílací pravděpodobnost $b_j(o_t)$ se spočítá podle vztahu (24).

$$lsc_j(o_t) = -\log b_j(o_t) - \min_i\left(-\log b_i(o_t)\right) \tag{24}$$

Výsledná hodnota (nazývaná lokální skóre) je podle něj rovna rozdílu opačné hodnoty logaritmu vysílací pravděpodobnosti daného stavu a minimální opačné hodnoty vysílací pravděpodobnosti ze všech stavů. Čím bude vysílací pravděpodobnost větší, tím bude lokální skóre menší. Pro nejpravděpodobnější stav bude lokální skóre rovno nule.

Druhá normalizace se týká délky cesty v modelu klíčového slova. Skóre je vyděleno počtem přechodů mezi jednotlivými stavy. Tato normalizace probíhá průběžně při každém přechodu mezi dvěma stavy podle vztahu (25).

$$nsc_j(t) = \min_i \frac{nsc_i(t-1)L_i(t-1) + lsc_j(o_t)}{1 + L_i(t-1)} \quad L_j(t) = L_k(t-1) + 1 \tag{25}$$

Výsledná hodnota (nazývaná normalizované skóre) nahrazuje částečnou Viterbiho likelihood v původním algoritmu. Ze vztahu vyplývá, že algoritmus vyžaduje, aby v paměti byla uchovávána kromě normalizovaného skóre také odpovídající aktuální délka. Normalizované skóre umožňuje začít novou cestu pro každý vstupní rámec a udržovat pro každý stav pouze jednu (nejlepší a tedy minimální) jeho hodnotu. Nová cesta v modelu může začít v kterémkoliv čase *t* s parametry (26).

$$nsc_1(t) = lsc_1(o_t) \quad L_1(t) = 1 \tag{26}$$

Normalizované skóre zároveň umožňuje nastavit jednu hodnotu prahu pro detekci klíčových slov. Pokud se v promluvě vyskytne klíčové slovo, hodnota jeho normalizovaného skóre by měla být blízká nule.

4 Implementace

Detektor klíčových slov byl implementován podle popisu uvedeného v předchozí kapitole.

V první části této kapitoly jsou popsány specifika implementace vzhledem k použitému programovacímu jazyku C++. Počátky vývoje systému Symbian se datují do doby před uvolněním standardu tohoto jazyka a tak jsou některé jeho konstrukty řešeny odlišně než ve standardní verzi. Rozdíly v samotném jazyce jsou však minimální, spíše se jedná o rozdíly v dostupných knihovních funkcích. Některé odlišnosti a omezení ve způsobu programování také vyplývají z omezených výpočetních prostředků cílových zařízení.

V další části kapitoly uvádím prototyp detektoru, který byl vyvinut v jazyce C pro PC v rámci semestrálního projektu. Prototyp sloužil jednak k natrénování neuronové sítě a jednak k posouzení, které části detektoru by bylo vhodné ve finální verzi optimalizovat. V rámci optimalizace byla také vybrána konečná varianta detektoru pokud jde o části, ve kterých bylo navrhnuto více možných řešení (viz. kap. 3). Myšlenka realizovat nejdříve prototyp detektoru pro PC byla také motivována usnadněním počáteční fáze implementace pod OS Symbian.

Poslední část se věnuje detailům implementace finálního detektoru pod OS Symbian. Detektor nebyl implementován jako samostatná aplikace, ale místo toho jako dynamická knihovna, kterou budou moci ostatní aplikace využívat pro konkrétní zvolený účel. Prezentován je objektový návrh knihovny a jednotlivé třídy, které implementují její funkčnost. Rozhraní knihovny tvoří třída, která umožňuje spustit a zastavit detekci, nastavit výchozí práh pro příjem slova a přidávat či odebírat klíčová slova. Detekovaná klíčová slova jsou uživateli předávána pomocí asynchronní callback metody. Knihovna byla otestována ve dvou jednoduchých aplikacích.

4.1 Symbian C++

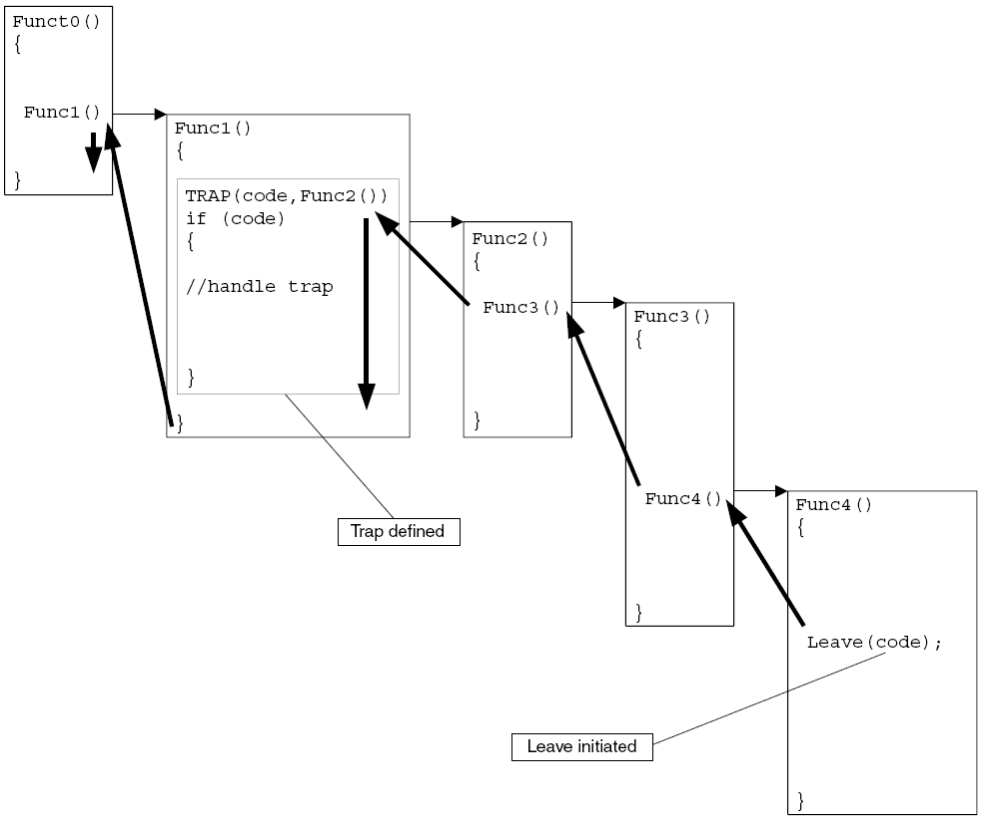
Jak již bylo naznačeno v úvodu kapitoly, programování v C++ pod Symbian OS má některá specifika, která se promítla do implementace knihovny. Příkladem na úvod může být neexistence STL pod Symbian. V této podkapitole budou popsány základní odlišnosti od programování ve standardním C++ pro běžné operační systémy. Popsána bude správa výjimek, způsob vytváření a práce s dynamickými knihovnami, konvence pojmenování identifikátorů, aktivní objekty a využití MMF pro audio streaming z mikrofonu.

4.1.1 Správa výjimek

Jednou z vlastností C++, která je ve verzi pro Symbian OS řešena odlišným způsobem, je správa výjimek. Místo mechanismu throw/catch ve standardním C++ je v Symbian mechanismus leave/trap. Zjednodušeně řečeno je tento mechanismus odlehčenou a efektivnější metodou než je standardní řešení. Princip generování výjimek a jejich ošetření je znázorněn na obrázku 4.1. Výjimka je generována metodou `Leave()` a odchycena pomocí makra `TRAP()`. Metodě `Leave()` je předán celočíselný kód chyby, který může být při odchycení výjimky makrem `TRAP()` načten a zpracován. V tomto je mechanismus úspornější, protože výjimka není jako v C++ parametrizována libovolným objektem.

Knihovna je řešena tak, že veškeré výjimky, které nastanou v libovolné z funkcí, ať už díky nedostatku paměti nebo díky chybě v nějaké z použitých API funkcí, jsou propagovány uživateli. Knihovna nedeklaruje vlastní chybové kódy výjimek a používá pouze standardní návratové kódy

definované v SDK pro Symbian. U každé funkce by měl uživatel vědět, zda může generovat výjimku. To je jednak z důvodu, aby ji případně mohl ošetřit nebo aby zajistil uvolnění dynamicky alokovaných proměnných. Symbian z tohoto důvodu definuje pro pojmenování funkcí, které mohou skončit díky leave, konvenci, která spočívá v přidání písmene *L* na konec jejich názvu. Implementace knihovny tuto konvenci také důsledně dodržuje.



Obr. 4.1: Leave-trap mechanismus výjimek (převzato z [19])

S mechanismem leave/trap je spojen speciální cleanup mechanismus, který slouží k uvolnění alokovaných zdrojů při předčasném opuštění funkce v důsledku leave. Tento mechanismus je založen na zásobníku ukazatelů na dynamicky alokované proměnné. Pokud je v rámci nějaké funkce alokována paměť pomocí `new`, která musí být při generování výjimky uvolněna, stačí ukazatel na danou proměnnou po alokaci přidat na cleanup zásobník. Při vzniku leave systém automaticky ukazatel odstraní a zavolá na něj `delete`. Pokud funkce proběhne bez generování výjimky, uživatel odstraní ukazatel ze zásobníku manuálně.

Problémem, který v souvislosti s cleanup mechanismem vyvstává, je použití konstruktorů, které můžou generovat výjimku. Mezi alokací paměti voláním `new` a prováděním těla konstruktoru totiž nemá uživatel možnost přidat ukazatel na cleanup zásobník. Tento problém je v Symbian vyřešen pomocí tzv. dvoufázového konstruktoru. Tento koncept zakazuje použití kódu, který může generovat výjimku v konstruktoru a pro jeho provedení definuje druhý konstruktor. Tento konstruktor je implementován jako běžná metoda a je standardně pojmenován `ConstructL()`. Instanciace objektu tedy spočívá nejdříve ve volání běžného konstruktoru, po němž následuje přidání ukazatele na objekt do cleanup zásobníku a nakonec je dokončena inicializace objektu voláním `ConstructL()`. Aby bylo zaručeno, že uživatel zavolá i druhou fázi konstrukce, poskytují třídy SDK statickou metodu `NewL()`, která uvedenou inicializační posloupnost vykoná. Knihovna

implementující detektor tento koncept také splňuje a definuje statické konstruktory pro snadnou inicializaci objektů.

4.1.2 Dynamické knihovny

Podobně jako na jiných operačních systémech i v Symbian OS existují dva typy knihoven – statické a dynamické. Statické knihovny jsou linkovány k programu během kompilace, dynamické knihovny za běhu. Knihovna pro detektor klíčových slov byla vytvořena jako dynamická knihovna. Přestože je princip dynamických knihoven stejný jako v běžných OS, některé rozdíly tu přece existují.

Výhodou dynamických knihoven je, že jejich kód je natažen do paměti pouze jednou a je sdílen všemi procesy, které knihovnu používají. Tato vlastnost však v případě detektoru není podstatná, protože se nepřipouští jeho využití více než jedním procesem současně. Důvod je čistě technický a spočívá ve výlučném přístupu k API pro audio streaming z mikrofonu.

Dynamické knihovny pod Symbian OS mohou obsahovat třídy jazyka C++ se zachováním všech jejich vlastností. Třídy, které tvoří rozhraní knihovny mohou být v uživatelských programech využívány jak běžnou instanciací, tak prostřednictvím dědičnosti. Rozhraní knihovny detektoru je tvořeno dvěma třídami. Hlavní třída, která slouží pro ovládání detektoru, je zpřístupněna instanciací. Další třída definuje rozhraní, které musí implementovat uživatelská třída pro příjem zpráv o detekci klíčového slova. Tato třída je tedy dostupná děděním.

Rozhraní knihovny se v Symbian OS nespecifikuje pro celou třídu, ale pro jednotlivé metody. Znamená to tedy, že metody, které nejsou určeny pro export z knihovny, zůstanou uživateli nepřístupné a to i přesto, že mohou být definovány jako `public`. Knihovna tohoto nijak nezneužívá a exportuje všechny veřejné metody v třídě pro aplikační rozhraní.

Pro přilinkování dynamické knihovny slouží odpovídající statická knihovna. Ta obaluje exportované funkce a zajišťuje nahrání dynamické knihovny do paměti, pokud je to zapotřebí. Vzhledem k tomu, že jsou programy linkovány ke statické knihovně, můžou vzniknout problémy, pokud verze dynamické knihovny statické knihovně neodpovídá. Pokud je totiž překládána nová verze dynamické knihovny, běžně není zajištěno, že původní funkce budou na stejných adresách, přestože jinak může být nová verze knihovny kompatibilní s předchozí. Symbian tento problém řeší pomocí tzv. DLL interface freezing. Díky němu se při překladu vytvoří definiční soubor rozhraní knihovny. Tento definiční soubor se použije při překladu novější verze a původní funkce tak budou přístupné i pro programy, které byly překládány se starší verzí knihovny. Knihovna pro detektor tento přístup taky využívá s ohledem na její možné bezproblémové rozšíření do budoucna.

Symbian OS klade na dynamické knihovny jedno zásadnější omezení. To zakazuje použití globálních proměnných uvnitř dynamických knihoven. Důvodem je snaha o minimalizaci paměťových nároků. Vzhledem k tomu, že by pro každý proces musely být vlastní globální proměnné a vzhledem k počtu systémových knihoven, by mělo povolení globálních proměnných velký vliv na výkon systému. Konstantní globální data určená jen pro čtení však nečiní problémy a jsou povolená. Takto jsou v knihovně detektoru například uloženy parametry neuronové sítě, tedy její váhy a prahy.

4.1.3 Konvence pojmenování tříd

Symbian OS definuje čtyři typy tříd podle účelu, kterému slouží. Pro jejich vzájemné rozlišení slouží konvence pojmenování, kterou Symbian rovněž zavádí. Tuto konvenci je vhodné dodržovat i v uživatelských programech a knihovna pro detektor ji také následuje. Třídy se rozlišují podle počátečního písmene jejich názvu na:

T – třídy datových typů (Type)
C – dynamicky alokované třídy (CBase)
R – třídy pro přístup ke zdrojům (Resource)
M – třídy definující rozhraní (Mixin)

Třídy datových typů jsou jednoduché třídy zapouzdřující většinou nějaký primitivní typ, který doplňují o operace nad ním.

Dynamicky alokované třídy jsou odvozené od třídy `CBase`. Jejich objekty by nikdy neměly být deklarovány staticky. (Jeden z důvodů je, že třídy spoléhají na vynulování členských proměnných při inicializaci, což je zajištěno právě operátorem `new` díky nadřazené třídě `CBase`.) Mezi tyto třídy patří i hlavní třída definující rozhraní detektoru klíčových slov.

Třídy pro přístup ke zdrojům se naopak typicky instanciují na zásobníku. Slouží pro ovládání vzdálených objektů. Implementuje se pomocí nich klientská část aplikace v modelu klient/server. Tyto třídy také slouží pro přístup k funkcím jádra systému.

Posledním typem tříd jsou třídy rozhraní. Slouží k podobnému účelu jako např. `interface` v jazyce Java. Často definují pouze čistě virtuální metody, jež implementují třídy, které rozhraní rozšiřují. V knihovně detektoru třída tohoto typu slouží pro definování abstraktní callback metody pro příjem události detekce klíčového slova.

4.1.4 Aktivní objekty

Symbian OS je operační systém, který poskytuje v mnohém ohledu tytéž služby, které jsou běžné na standardních operačních systémech pro osobní počítače. Jednou z jeho funkcí je také podpora preemptivního multitaskingu na úrovni vláken. Přestože nám tedy nic nebrání v tom, abychom vytvářeli v rámci procesu další vlákna, v praktických aplikacích se bez nich velmi často dokážeme obejít. Symbian OS totiž poskytuje alternativu ve formě efektivnějšího kooperativního multitaskingu, který využívá pouze jedno vlákno. Základními prostředky pro práci s kooperativním multitaskingem jsou asynchronní funkce, aktivní objekty a plánovač událostí.

Asynchronní funkce jsou funkce, které bezprostředně po jejich zavolání vrátí řízení zpět a pokračují v činnosti v pozadí paralelně s vláknem, které je vyvolalo. Na rozdíl od běžných synchronních funkcí tedy neblokují provádění programu do doby než skončí. Neblokující asynchronní funkce jsou tedy jistě v mnoha případech žádoucí. Princip práce s asynchronními funkcemi spočívá v jejich zavolání, následném provádění jiné činnosti a nakonec ve zpracování jejich dokončení. Řada systémových funkcí v Symbian OS je implementována právě jako asynchronní funkce. V implementaci detektoru se používají asynchronní funkce pro audio streaming z mikrofonu a samotná detekce klíčového slova je zpracována v události dokončení jedné z nich.

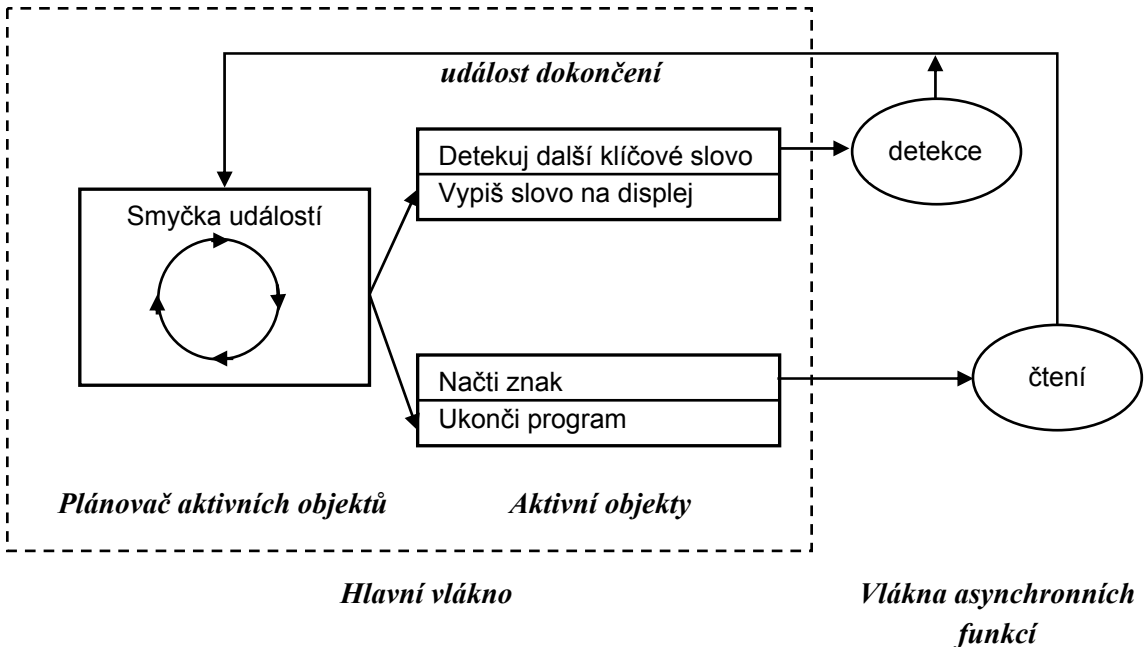
Problémem při práci s asynchronními funkcemi je však právě detekce jejich dokončení. Nejméně efektivním řešením by bylo aktivní čekání ve formě neustálého dotazování se, zda již funkce neskončila. Mnohem lepším řešením je pasivní čekání na dokončení pomocí systémové funkce s využitím semaforu. Takovýto způsob zbytečně nezaměstnává procesor a v mnoha případech má praktické využití. Nejefektivnějším řešením pro práci s asynchronními funkcemi je však pod Symbian OS použití aktivních objektů.

Aktivní objekty slouží pro volání asynchronních funkcí a pro příjem události jejich dokončení. Aktivní objekty však nečekají na dokončení asynchronních funkcí přímo, ale definují speciální callback metody, které jsou po dokončení zavolány. V těchto metodách je zpracována dokončená operace a může být zavolána další asynchronní funkce. Callback metody jsou invokovány tzv.

plánovačem událostí. Všechny aktivní objekty mají společnou bázovou třídu, se kterou plánovač událostí pracuje.

Plánovač událostí je prostředek systému, který uchovává aktivní objekty. Jeho činnost by se dala popsat cyklem, ve kterém vždy čeká na dokončení libovolné operace a následně volá příslušnou callback metodu. Výhodou je, že jak plánovač tak callback metody běží jen v jednom vlákně. Z uvedeného principu vyplývá, že od chvíle, kdy je plánovač spuštěn, může být prováděn pouze kód callback metod aktivních objektů. Naopak, pokud plánovač událostí není spuštěn, aktivní objekty jsou nečinné.

Implementace detektoru klíčových slov, jak již bylo řečeno, využívá asynchronní funkce. Ty poskytuje API pro načítání dat z mikrofonu, které rovněž zapouzdřuje aktivní objekt pro zpracování události dokončeného čtení. Princip činnosti plánovače událostí a aktivních objektů je znázorněn na obrázku 4.2 na zjednodušeném příkladu použití knihovny pro detekci klíčových slov.



Obr. 4.2: Aktivní objekty – příklad použití detektoru klíčových slov

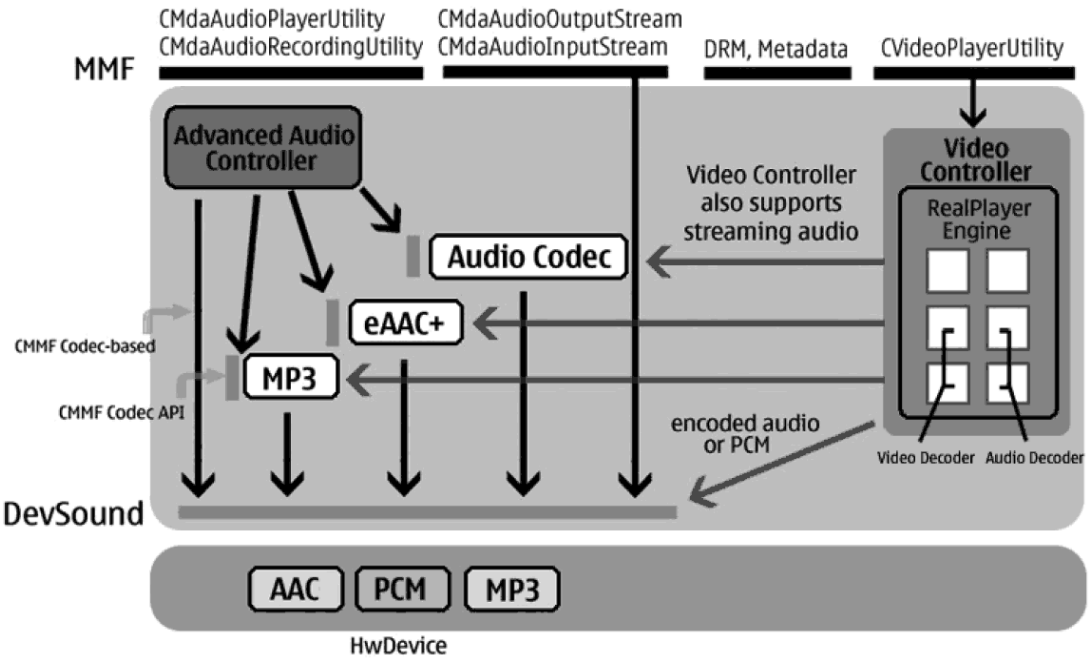
Zobrazené schéma reprezentuje jednoduchý program, který používá knihovnu pro výpis detekovaných slov na displej. Detekce klíčových slov i činnost celého programu je ukončena stiskem libovolné klávesy. Uvedená funkčnost programu je zajištěna pomocí dvou aktivních objektů, které jsou řízeny plánovačem událostí. Jeden aktivní objekt ve své calllback metodě přijímá detekovaná klíčová slova, vypisuje je na displej a vždy znovu spouští asynchronní funkci knihovny pro detekci dalšího klíčového slova. Druhý aktivní objekt spustí asynchronní funkci pro načtení znaku klávesy. Pokud dojde k vyvolání jeho callback metody plánovačem, klávesa byla stisknuta a tak metoda ukončí program.

Pro upřesnění budiž řečeno, že je uvedený příklad oproti implementaci mírně zjednodušený. V uživatelské metodě pro příjem události detekce klíčového slova pouze uživatel zpracuje detekované slovo a nemusí ručně volat asynchronní funkci pro detekci dalšího klíčového slova resp. pro načtení dalšího rámce z audio vstupu. Použití aktivního objektu v pozadí knihovny nicméně není pro uživatele zcela transparentní. Aby byl aktivní objekt uvnitř knihovny aktivní a aby tedy fungovalo asynchronní volání callback metody při detekovaném slovu, musí být spuštěn plánovač událostí. A z toho tedy vyplývá, že i uživatelský program musí fungovat jako aktivní objekt.

Pokud je uživatelský program ve formě GUI aplikace, nečiní výše uvedená podmínka na uživatele žádné nároky. GUI aplikace je totiž již z podstaty řízená událostmi a plánovač událostí je automaticky aktivní. Pokud se jedná o konzolovou aplikaci, která se však pod Symbian OS používá téměř výhradně k testovacím účelům při vývoji aplikací, musí být plánovač událostí explicitně spuštěn a program musí být řízen prostřednictvím aktivního objektu. (Příklad použití knihovny, jak v GUI tak v konzolové aplikaci, je na přiloženém CD.)

4.1.5 Audio streaming z mikrofonu

Pro funkčnost detektoru je nutné zajistit načítání vstupního proudu z mikrofonu. Podpora pro zpracování audia je v Symbian OS na poměrně dobré úrovni (především v novějších verzích systému) díky tomu, že multimediální aplikace tvoří nezanedbatelnou část aplikací pro mobilní telefony. Aplikační rozhraní systému poskytuje v zásadě dvě možnosti implementace načítání dat z mikrofonu. První možností je využití nízkoúrovňového API nazývaného DevSound, druhou možností je pak implementace založená na middleware nazývaném Multimedia Framework. Obecně se těmto komponentám věnovala již kapitola 2.1.2, nyní budou tyto komponenty popsány z implementačního hlediska.



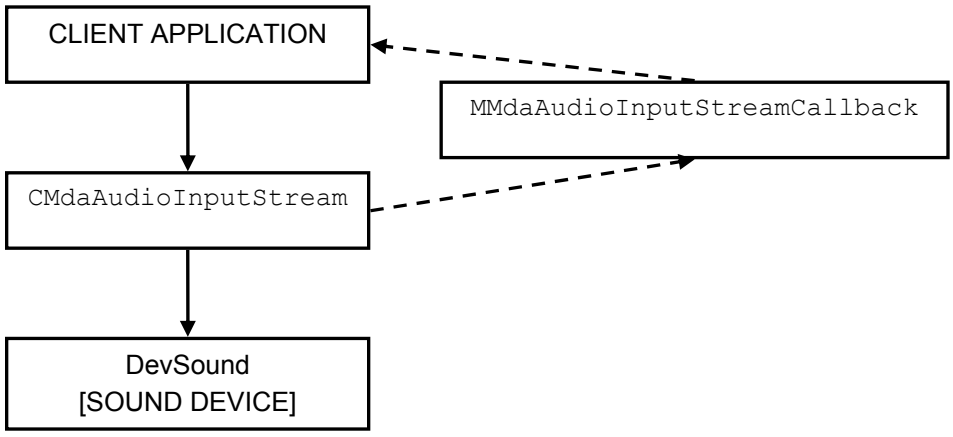
Obr. 4.3: Architektura Multimedia Framework (převzato z [22])

Komponenta DevSound tvoří nejnižší vrstvu abstrakce multimediálního hardware, která je dostupná uživateli. DevSound poskytuje potřebné rozhraní pro ovládání audio zařízení a zpřístupňuje dostupné hardwarové i softwarové kodeky. Dá se říci, že veškeré aplikace založené na zpracování audia jsou plně realizovatelné pouze s využitím této komponenty. Díky možnosti detailní kontroly činnosti hardware je tato komponenta užitečná např. v aplikacích, které vyžadují dokonalou synchronizaci přehrávání zvuku s vykreslováním údajů na displej, jako je tomu např. u her. Práce s DevSound nicméně není jednoduchou záležitostí a proto je k dispozici MMF, které má podstatně jednodušší rozhraní se zachováním funkčnosti požadované většinou multimediálních aplikací.

Multimedia Framework je API vyšší úrovně pro práci s audiem, videem a obrázky dostupné v OS Symbian od verze 7.0s. Framework je implementován jako plugin a je vystavěn nad vrstvou

DevSound. MMF poskytuje přístupnější formou část rozhraní DevSound a také implementuje některé další kodeky a rozšíření. Architektura MMF je znázorněna na obrázku 4.3. Rozhraní pro zpracování vstupu z mikrofonu je zajištěno třídou `CMdaAudioInputStream`. Vzhledem k tomu, že MMF je pro naše účely plně postačující, byl zvolen pro implementaci detektoru klíčových slov. Drobným omezením je jeho dostupnost od verze 7.0s, ale jak již bylo uvedeno v kapitole 2.1.2, teoretická možnost jeho doinstalování na nižší verze systému existuje.

Data z mikrofonu jsou načítána pomocí již zmíněné třídy `CMdaAudioInputStream`. Tato třída poskytuje potřebné metody pro spuštění a zastavení nahrávání a pro příjem načtených dat. Zatímco prvně zmíněné metody jsou volány přímo klientskou částí aplikace, načtená data jsou předávána asynchronní metodou, kterou klient implementuje, ale volá ji při dokončení čtení samotná třída `CMdaAudioInputStream`. Druhá a poslední třída, která bude v této kapitole zmíněna je třída `MMdaAudioInputStreamCallback`. Tato třída slouží jako rozhraní, které definuje všechny asynchronní callback metody, které jsou při nahrávání z mikrofonu zapotřebí. Kromě zmíněné metody pro zpracování načtených dat jsou to ještě metody pro notifikaci uživatele o dokončeném otevření a uzavření streamu. Vztah a komunikace jednotlivých tříd s klientskou částí aplikace na jedné straně a s rozhraním DevSound na straně druhé jsou zobrazeny na obrázku 4.4. Dále bude popsán princip načítání dat z mikrofonu.



Obr. 4.4: Diagram tříd při použití MMF pro audio streaming (převzato a upraveno z [23])

Nahrávání z mikrofonu se jednoduše spustí metodou pro otevření audio streamu. Zvláštností, která není zřejmá, je fakt, že instance pro nahrávání musí být před otevřením čerstvě vytvořená. Pokus o otevření instance, která již byla používána a která již byla uzavřena by totiž na některých telefonech mohl selhat. Před otevřením proudu je ještě možné nastavit typ kódování, počet kanálů, vzorkovací frekvenci atp. Konkrétní nastavení je uvedeno v kapitole 4.3.1. Po spuštění nahrávání se již většina událostí odehrává asynchronně.

Po otevření vstupního proudu je nejdříve zavolána speciální callback metoda, která slouží výhradně pro ověření úspěšného otevření streamu a pro první zavolání metody pro čtení dat z mikrofonu. Tato metoda je opět asynchronní a její dokončení je signalizováno zavoláním další callback metody, která slouží pro příjem načtených dat.

Načtená data se v příslušné callback metodě zpracují a zavolá se další asynchronní čtení. V našem případě budou načtená data reprezentovat jeden řečový rámec. V události načtení dat tedy proběhne celé zpracování daného rámce od extrakce příznaků až po případnou detekci klíčového slova.

Nahrávání se ukončí opět jednoduše voláním příslušné metody. V závislosti na spuštění této metody jsou invokovány následující události. Bezprostředně po zavolání metody je ukončeno čtení, které případně probíhá a naposledy je vyvolána událost pro zpracování zbytku dat, které již byly načteny. Poslední událost, která je při ukončení nahrávání spuštěna, pak slouží pro případné uvolnění alokovaných zdrojů.

4.2 Prototyp detektoru v C

V rámci semestrálního projektu byla implementována část detektoru zajišťující extrakci kepstrálních příznaků a vyhodnocení neuronové sítě algoritmem Forward pass. Program byl napsaný v jazyce C a je určen pro Linux a Windows. V diplomové práci byl využit jednak jako základ pro implementaci kompletního detektoru v jazyce C++ pro OS Symbian a jednak pro počáteční testování a optimalizaci. Pro referenční účely implementace byl použit tutoriál [15] a toolkit HTK [16] a STK [17].

Program zpracovává vstupní zvukový soubor ve formátu WAV a výstup ukládá do souboru ve formátu HTK. Program byl implementován ve dvou základních variantách. První slouží pouze pro extrakci příznaků a druhá implementuje navíc algoritmus Forward pass nad již natrénovanou sítí. V první variantě jsou tedy pro každý vstupní řečový rámec výstupem vektory příznaků a ve druhé variantě vektory posterior pravděpodobností jednotlivých fonémů.

Program byl vytvořen jako konzolová aplikace. Ovládá se předáním dvou argumentů při spuštění. Prvním z nich je název vstupního souboru WAV, druhým je název výstupního souboru HTK. Obsah výstupního souboru může být zobrazen utilitou HList z toolkitu HTK. Správnost činnosti byla ověřena nástrojem HCopy z téhož toolkitu a SFeaCat z toolkitu STK.

Extrakce příznaků resp. algoritmus Forward pass byl implementován podle postupu uvedeného v kap. 3.1.1 resp. 3.2. V následující části jsou představeny detaily provedení, které nebyly v uvedených kapitolách popsány.

4.2.1 Detaily provedení

Program zpracovává vstupní vzorkovaný signál po rámcích. Délka rámce je nastavena na 25 ms, posunutí rámců je 10 ms. Pro každý rámec (s drobnou výjimkou viz. dále) program spočítá výstupní vektor příznaků resp. pravděpodobností.

Nejdříve je spočítáno spektrum a z něho následně pomocí bank filtrů mel spektrum. Prvním krokem po vážení Hammingovým oknem (3) je úprava délky vstupního rámce tak, aby byla mocninou dvojky. Rámec za tímto účelem jednoduše rozšíříme o nuly. V našem případě doplníme rámec na 512 vzorků (v případě vzorkovací frekvence 16 kHz má původní rámec 400 vzorků). To nám umožní pro výpočet spektra použít efektivní algoritmus Fourierovy transformace (FFT). Spektrum dále násobíme s filtry v odpovídajících vzorcích, které pak sčítáme (8). Filtry jsou rozmístěny po frekvenční ose v rozmezí od 0 do 8000 Hz. V programu je nastaven počet filtrů na 23. Tímto způsobem tedy obdržíme 23 koeficientů mel spektra.

Na koeficienty mel spektra je následně aplikována DCT (9). Transformace se však neprovádí nad jednotlivými vektory koeficientů, ale přes delší časový kontext. Ten je nastaven na 31 vektorů. První transformace se tedy provede nad prvními koeficienty prvních 31 vektorů, druhá nad druhými koeficienty těchto vektorů, atd. až 23. transformace se provede nad 23. koeficienty prvních 31 vektorů. Takto získané výsledky DCT pro 23 koeficientů a 31 vektorů spojíme v jeden výstupní vektor. Vzhledem k tomu, že v programu používáme jen prvních 16 koeficientů DCT bude mít výstupní vektor $16 \times 23 = 368$ dimenzí. V dalším kroku odstraníme první vektor koeficientů, přidáme

na konec nový a počítáme výstup stejným způsobem. Z uvedeného také vyplývá, že pro prvních 30 rámců program neprodukuje žádný výstup.

Vektor koeficientů je výstupem v první verzi programu. Ve druhé je následně normalizován (19) a poté přiveden na vstup neuronové sítě. Parametry pro normalizaci – odchylky a střední hodnoty a parametry pro neuronovou síť – váhy a prahy jsou uloženy v programu jako konstantní data. Pro testovací účely byla použita data ze zmíněného tutoriálu [15]. Připomeňme, že program pouze zjišťuje odezvu již natrénované neuronové sítě na zadaný vektor. Výstupem sítě i druhé varianty programu pro jeden rámec je vektor pravděpodobností výskytu jednotlivých fonémů. Celkový počet fonémů a velikost výstupního vektoru je ve shodě s tutoriálem 39.

4.2.2 Optimalizace

Během implementace a následného testování programu byly odhaleny jeho části, které bylo vhodné (nutné) optimalizovat pro akceleraci výpočtu. Akcelerace je klíčová vzhledem k omezeným výpočetním prostředkům zařízení, pro která je výsledný detektor určen.

Implementace části detektoru v prototypu např. opakovaně pro každý rámec počítala tytéž tvary filtrů pro výpočet mel spektra. Obecně byla identifikována velká část kódu, jehož efekt byl invariantní vůči aktuálně zpracovávanému rámci. Zefektivnění v tomto případě spočívalo v předpočítání pokud možno všech těchto neměnných částí výpočtu ještě před zpracováním prvního rámce na počátku běhu programu. Výsledné hodnoty pak byly uloženy a přímo použity pro zpracování jednotlivých rámců. Kromě toho bylo možné zoptimalizovat i některé části prováděné pro každý rámec s jedinečnými hodnotami. Příkladem může být výpočet mel spektra, který byl inspirován řešením použitým v HTK (viz. kap. 4.3.2).

Profilováním se zjistilo, že největší potíže s rychlostí výpočtu i nároky na paměť má neuronová síť. Ta obsahuje ve výše popsané implementaci 500 neuronů ve skryté vrstvě. Uložení parametrů sítě v operační paměti představuje stovky kilobajtů, což je v případě mobilních telefonů nezanedbatelná velikost. Vyhodnocení takto velké sítě je vzhledem k počtu operací, které se musí provést, rovněž náročné na výpočetní zdroje a má velký vliv na celkovou rychlost zpracování.

Běžně se řeší problém optimalizace algoritmu Forward pass tak, že se převede na násobení a sčítání matic, pro které existují optimalizované knihovny (např. BLAS). Tyto knihovny jsou však implementovány pro standardní OS a procesory osobních počítačů. Pro OS Symbian podobná knihovna bohužel neexistuje. Možné řešení tedy spočívalo ve zmenšení velikosti vstupního vektoru sítě a snížení počtu neuronů skryté vrstvy byť za cenu horší přesnosti detekce klíčových slov. Toto řešení bylo realizovatelné buď zkrácením časového kontextu, ze kterého se vstup sítě vypočítává a nebo použitím druhé navrhované metody extrakce příznaků – PLP. Vzhledem k tomu, že samotná práce s časovým kontextem generuje navíc jisté nároky na paměť, protože je potřeba uchovávat buffer aktuálních vektorů koeficientů a protože bylo zapotřebí zmenšit velikost sítě podstatným způsobem, byla pro výsledný detektor zvolena varianta PLP příznaků (viz. kap. 4.3.2).

Poslední optimalizace, která byla provedena, se opět týkala neuronové sítě. Obě použité aktivační funkce neuronů, sigmoida (20) i softmax (21) obsahují výpočet exponenciální funkce. Vzhledem k počtu neuronů bylo vhodné výpočet exponenciál urychlit. Nabízelo se pro to efektivní řešení. Jeho implementace v podobě makra `FAST_EXP()` v jazyce ANSI C je uvedena níže, podrobně je pak popsána v [18].

```
static union {
      double d;
      struct {
#if BYTE_ORDER == BIG_ENDIAN
      int i, j;
#else
      int j, i;
#endif
      } n;
} d2i;

#define EXP_A (1048576/M_LN2)
#define EXP_C 60801
#define FAST_EXP(y) \
      (qn_d2i.n.i = (int)(EXP_A*(y))+(1072693248-EXP_C),d2i.d)
```

Vzhledem k tomu, že Symbian podporuje ANSI C, je tato optimalizace použitelná.

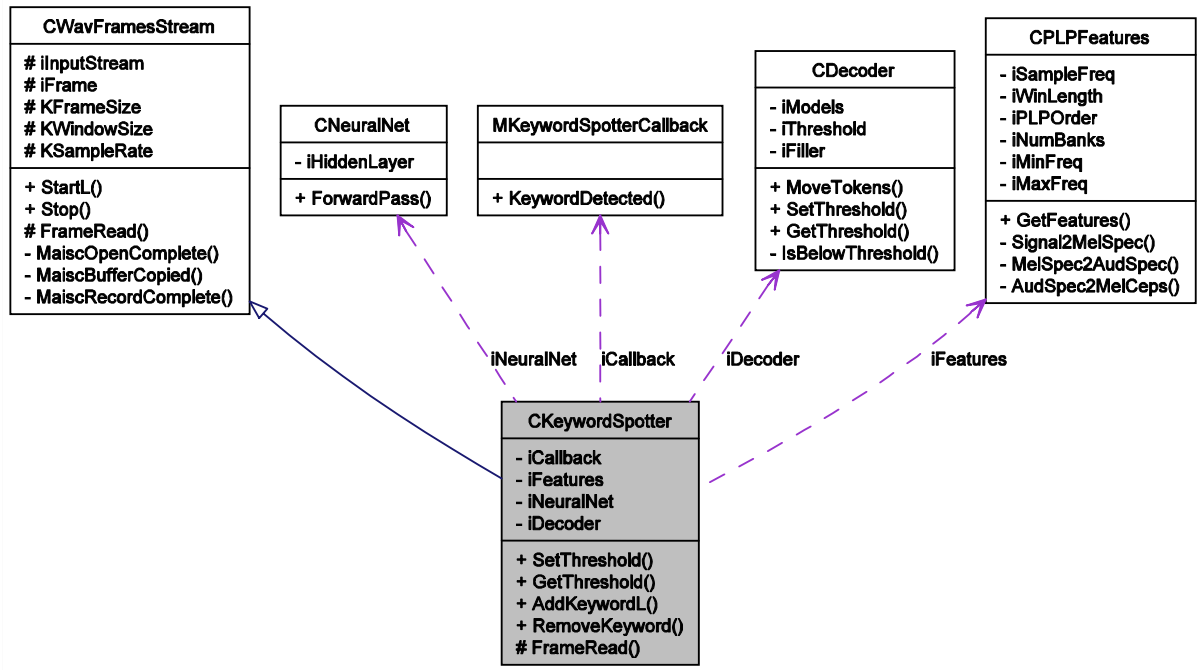
4.3 Objektový návrh a implementace

Na základě postupu uvedeného v kapitole 3 a zkušeností získaných implementací prototypu byl navržen objektový model finálního detektoru klíčových slov a byla provedena jeho implementace pro Symbian OS v jazyce C++. Výsledná aplikace je ve formě dynamické knihovny (DLL). Tu můžou využívat libovolné další programy pro zajištění detekce klíčových slov pro požadovaný účel. V této kapitole budou popsány jednotlivé části knihovny tak, jak byly navrženy a implementovány. Uvedeny budou především detaily provedení, které nebyly doposud ozřejměny v předchozích kapitolách. Popsáno bude také konkrétní nastavení a konfigurace jednotlivých parametrů detektoru.

Knihovna detektoru klíčových slov byla navržena s ohledem na implementační jazyk objektově. Je to jednak z důvodu snazší udržovatelnosti a také z důvodu konzistence s aplikačním rozhraním OS Symbian, které je také navrženo objektově. Diagram tříd knihovny je znázorněn na obrázku 4.5, vynechány jsou pouze některé vlastnosti a metody, které nejsou pro vysvětlení problematiky podstatné.

Rozhraní knihovny tvoří třída `CKeywordSpotter`. Ta obsahuje veškeré metody potřebné pro ovládání detektoru. Umožňuje spustit a zastavit detekci klíčových slov, nastavit implicitní hodnotu prahu pro přijetí slova a přidávat resp. odebírat klíčová slova ze seznamu a to i za běhu detektoru. Zmíněná třída využívá pro zajištění funkčnosti detektoru pět dalších tříd. Třída `MKeywordSpotter` je součástí API knihovny. Je to abstraktní třída definující rozhraní, které musí implementovat uživatelská třída pro příjem události detekce klíčového slova. Třída `CWavFramesStream` slouží pro nahrávání vstupu z mikrofonu po rámcích. Tato třída je bázovou třídou pro `CKeywordSpotter`. Zbývající třídy `CNeuralNet`, `CDecoder` a `CPLPFeatures` implementují odpovídající části detektoru. Přestože tyto třídy slouží jednomu konkrétnímu účelu a pracují vždy se stejným nastavením, byla snaha, aby byly vytvořeny pokud možno univerzálně. Jedná se především o umožnění konfigurace některých parametrů (např. počet bank filtrů u extrakce příznaků), kde obecné řešení neovlivňuje výkon knihovny.

V následujících podkapitolách budou jednotlivé komponenty knihovny popsány podrobněji. Programová dokumentace aplikačního rozhraní knihovny je uvedena v příloze, úplná programová dokumentace všech částí knihovny je na přiloženém CD.



Obr. 4.5: Diagram tříd knihovny detektoru klíčových slov

4.3.1 Načítání řečových rámců

Načítání řečových rámců z mikrofonu je implementováno pomocí prostředků operačního systému, které byly popsány v kapitole 4.1.5. Odpovídající komponenta knihovny slouží pro řízení nahrávání a předávání řečových rámců pro další zpracování. Komponenta je tvořena abstraktní třídou, která je bázová pro rozhraní knihovny. Veřejné operace pro spouštění a zastavení nahrávání jsou tedy dostupné i uživateli knihovny. Ten je volá pro spuštění a zastavení detekce klíčových slov. Předávání načtených rámců do odvozené třídy je zajištěno pomocí abstraktní metody, která slouží jako callback metoda.

Komponenta je optimalizovaná pro daný účel a většina její konfigurace je tedy implementovaná napevno bez možnosti změny za běhu programu. Důvody, které k tomu vedly jsou popsány dále. Nastavení, které je neměnné, se týká velikosti řečového rámce, posunutí (překrytí) rámců, vzorkovací frekvence a kódování.

Multimedia framework použitý pro nahrávání vstupu z mikrofonu umožňuje různé typy kódování audio streamu. Podpora pro kódování je většinou dostupná ve formě kodeků, které jsou implementovány jako pluginy MFF. Ne všechny kodeky jsou však dostupné na všech zařízeních a proto je volba kódování důležitá s ohledem na přenositelnost. Mezi základní a nejrozšířenější typy kódování na Symbian OS patří AMR a PCM.

AMR (Adaptive Multi-Rate) je kódovaní vyvinuté Evropským ústavem pro telekomunikační normy (ETSI). Jak již název napovídá kódovaní podporuje několik režimů s různým bitrate. AMR existuje ve dvou verzích AMR-NB (narrow band) a AMR-WB (wide band). AMR je vhodné především pro kódování řeči.

PCM (Pulse-Code Modulation) je pod Symbian OS lineární kódování. Jeden vzorek signálu je kódován na 16 bitech. Jeho výhodou je, že jeho podpora je přímo součástí operačního systému a že je tedy dostupné na všech zařízeních. Toto kódování neposkytuje žádnou kompresi, ale to je v našem případě spíše výhoda s ohledem na další zpracování. Vzhledem k výše uvedenému tedy bylo pro načítání řečových rámců použito kódování PCM.

Délka řečového rámce byla nastavena na 25 ms, překrytí jednotlivých rámců pak na 10 ms. Při zvolené vzorkovací frekvenci 8000 Hz odpovídá 10 ms 80 vzorkům. Vzhledem k použitému PCM, které kóduje vzorek na 16 bitech pak dostáváme pro posuv rámce velikost 160 bajtů. API pro načítání dat z mikrofonu umožňuje najednou načíst maximálně 320 bajtů streamu (to platí pro S60 1st a 2nd Edition, ve verzi 3rd Edition je maximální velikost bufferu 4 kB). Tato velikost je tedy v uvedeném nastavení přesně dvojnásobná proti počtu bajtů, které musíme načíst pro každý nový rámec. Tohoto bylo využito a při každém čtení vstupu jsou tedy odeslány k dalšímu zpracování dva rámce. Budiž řečeno, že celý proces by se v případě načítání méně bajtů než je maximum zkomplikoval. MMF totiž používá interně buffer o zmíněné maximální velikosti, který před dalším naplněním nejdříve celý vyprázdní a při čtení menšího počtu bajtů bychom tedy nemuseli vždy obdržet požadovanou velikost.

Operace čtení audio vstupu probíhá, jak již bylo řečeno v kapitole 4.1.5, asynchronně. Při dokončení čtení daného bufferu je generována událost, která slouží pro zpracování načtených dat a pro vyvolání další operace čtení. Vzhledem k tomu, že zpracování jednoho řečového rámce je časově náročné, je vhodné využít asynchronního provádění čtení k tomu, aby obě operace probíhaly paralelně. V implementaci je toho docíleno pomocí dvou bufferů, které slouží střídavě pro čtení dat a jejich zpracování. Data nejdříve tedy načítáme do prvního bufferu. Po jeho načtení ihned zavoláme čtení pro druhý buffer a zpracováváme buffer první. Při dokončení čtení druhého bufferu se situace obrátí a probíhá analogickým způsobem dál.

Poslední poznámka, která bude zmíněna v souvislosti s načítáním řečových rámců, se týká výjimek. Výjimky, které nastanou v některé z asynchronních událostí, totiž není vhodné propagovat dál standardním způsobem. Výjimky by se totiž propagovaly do plánovače událostí (viz. kap. 4.1.4), jehož implicitní chování spočívá pouze v jejich odchycení bez žádné další akce. Výjimky by se tedy nedostaly do části aplikace, která detektor používá. Implementované řešení tedy spočívá v odchycení všech výjimek, které mohou v událostech nastat a předání jejich chybových kódů prostřednictvím callback metody, která slouží pro zpracování načtených dat. Z toho vyplývá, že při načtení každého rámce resp. při vyvolání události detekce klíčového slova je nutné nejdříve zkontrolovat, zda při čtení resp. detekci nenastala výjimka.

4.3.2 PLP příznaky

Komponenta pro výpočet PLP příznaků byla implementována na základě popisu v kapitole 3.1.3. Nicméně oproti uvedenému popisu se implementace v některých ohledech mírně liší. Je to především z důvodu zajištění kompatibility výpočtu příznaků s toolkitem HTK. Do implementace se také promítly optimalizace uvedené v kapitole 4.2.2, které se týkají předpočítání hodnot, které jsou neměnné v průběhu zpracování jednotlivých rámců. Níže budou popsány tyto implementační detaily s uvedením číselných hodnot parametrů, které jsou pro výpočet příznaků nastaveny.

Činnost komponenty by se dala pro účely popisu rozdělit na inicializační část a operaci pro výpočet vektoru příznaků pro vstupní řečový rámec. Inicializace nastaví parametry výpočtu a předpočítá hodnoty, u kterých je to možné. Komponenta byla navržena obecně, takže lze při inicializaci specifikovat hodnoty některých parametrů. Jsou to vzorkovací frekvence, velikost rámce v počtu vzorků a počet výstupních PLP koeficientů. Ostatní parametry mají přednastavené hodnoty, které nelze za běhu měnit. Nicméně tyto parametry jsou specifikovány symbolickými konstantami, které můžou být případně změněny, pokud by to bylo z nějakého důvodu nutné. Parametry jsou v knihovně nastaveny tak, aby odpovídaly nastavení podle konfigurace HTK uvedené níže (seznam obsahuje jen nejdůležitější parametry týkající se PLP). Pro tuto konfiguraci byla také samozřejmě trénovaná neuronová síť. Z konfigurace mj. vyplývá, že se ze vstupního rámce počítá 13 PLP

příznaků a pro výpočet mel spektra se používá 24 filtrů. Delta a akcelerační koeficienty nejsou použity.

```
TARGETKIND    = PLP_0
WINDOWSIZE    = 250000.0
TARGETRATE    = 100000.0
PREEMCOEF     = 0.97
USEHAMMING    = T
ZMEANSOURCE   = T
USEPOWER      = T
NUMCHANS      = 24
LOFREQ        = 125
HIFREQ        = 3800
LPCORDER      = 12
COMPRESSFACT  = 0.3333333
NUMCEPS       = 12
CEPLIFTER     = 22
CEPSCALE      = 10
```

V inicializační části se předpočítají filtry pro výpočet mel spektra, koeficienty equal-loudness křivky (11), matice kosinových bází pro výpočet inverzní DFT a koeficienty liftrovacího okna (15). Za zmínku stojí především předpočítání filtrů. Filtry jsou v implementaci řešeny stejným způsobem jako u kepstrálních příznaků (viz. kap. 3.1.2). Jsou tedy trojúhelníkové a jsou rovnoměrně rozmístěny po mel-frekvenční ose. Předpočítány jsou indexy filtrů pro všechny frekvence v daném rozmezí a jejich odpovídající hodnoty. Protože se filtry překrývají, jsou zmíněné hodnoty počítány pouze pro jejich pravé poloviny. Hodnoty filtru pro druhou polovinu se pak dopočítají jako doplněk do jedné. Tento přístup, který je používán i v HTK, velice zjednoduší samotný výpočet mel spektra (8). To je spočítáno pouze v jediném průchodu přes všechny frekvence. Pro každou frekvenci se akumuluje do výsledných koeficientů přírůstek od sousedních filtrů, jež mají na dané frekvenci nenulovou hodnotu.

Operace pro výpočet příznaků spočítá hodnoty výstupního vektoru na základě předpočítaných údajů a aktuálních vzorků vstupního rámce. Implementace se liší od základního postupu v následujících ohledech.

Místo DFT (5) je pro výpočet Fourierovy transformace použit efektivní algoritmus FFT. Ten je optimalizován pro vektory o délce, která je mocninou dvojky. Vstupní rámec je proto doplněn do potřebné velikosti nulami.

Výpočet PLP koeficientů z autokorelačních koeficientů je implementován rychlým algoritmem Levinson-Durbin. Inicializace algoritmu je popsána rovnicí (28). Jednotlivé kroky algoritmu jsou popsány výrazy (29) až (32). Ve výrazu *R(i)* značí i-tý autokorelační koeficient, $a_j^{(i)}$ zastupuje j-tý PLP koeficient vypočítaný v kroku *i*, *E* je nultý PLP koeficient a k_i je pomocná proměnná. Algoritmus se provádí iterativně tak dlouho kolik PLP koeficientů chceme spočítat (v našem případě 12).

$$E^{(0)} = R(0) \tag{28}$$

$$k_i = -\left[R(i) + \sum_{j=1}^{i-1} a_j^{(i-1)} R(i-j) \right] / E^{(i-1)} \tag{29}$$

$$a_i^{(i)} = k_i \tag{30}$$

$$a_j^{(i)} = a_j^{(i-1)} + k_i a_{i-j}^{(i-1)} \quad 1 \le j \le i-1 \tag{31}$$

$$E^{(i)} = \left(1 - k_i^2\right) E^{(i-1)} \tag{32}$$

Poslední implementační detail, který nebyl popsán v předchozím popisu výpočtu PLP příznaků se týká vážení koeficientů. Výstupní koeficienty jsou násobeny konstantou `CEPSCALE`.

4.3.3 Neuronová síť

Komponenta pro klasifikaci vektorů příznaků na fonémy implementuje algoritmus Forward pass neuronové sítě (viz. kap. 3.2). Komponenta byla navržena tak, aby byla schopna pracovat nad třívrstvou neuronovou sítí libovolné velikosti. Parametry neuronové sítě se komponentě předají při její inicializaci. Komponenta pak poskytuje operaci pro klasifikaci zadaného vektoru.

Parametry neuronové sítě jsou v knihovně uloženy odděleně od komponenty pro její vyhodnocení. Jsou uloženy přímo v programu jako konstantní data a není tedy zapotřebí je při spuštění knihovny načítat z externího souboru.

Neuronová síť pro rozpoznávání fonémů byla natrénovaná na databázi českých promluv SpeechDat-E[*]. Trénování bylo provedeno pomocí nástroje SNet [11], který je součástí STK. Neuronová síť má 13 vstupů (PLP koeficienty včetně nultého nakonec) a 45 výstupů (české fonémy). Natrénovány byly experimentálně dvě sítě o různém počtu neuronů skryté vrstvy. První síť obsahovala v této vrstvě 100 neuronů, druhá pak 500 neuronů. Úspěšnost klasifikace větší sítě byla zhruba o 1 % větší než úspěšnost menší sítě (viz. kap. 5). Vzhledem k tomuto nevýraznému zlepšení a také proto, že velikost sítě je zásadním faktorem, který ovlivňuje rychlost celého detektoru, byla pro implementaci vybrána síť se 100 neurony ve skryté vrstvě.

Na trénovacích datech byly také spočítány střední hodnoty a směrodatné odchylky vstupních vektorů, které jsou zapotřebí pro normalizaci (19).

4.3.4 Implementace dekodéru

V kapitole 3.3 byly popsány dva dekodéry pro detekci klíčových slov v matici posteriorních pravděpodobností, která je produkována neuronovou sítí. První dekodér byl založený na tradičním přístupu s využitím rozpoznávací sítě modelů slov a modelu pozadí (viz. kap. 3.3.1), druhý dekodér modifikoval Viterbiho algoritmus za účelem průběžné normalizace likelihood a možnosti odstranění modelu pozadí (viz. kap. 3.3.2). Implementovány byly obě varianty dekodéru a byly porovnány jejich vlastnosti.

Ve finálním detektoru je nakonec použita běžnější verze dekodéru s modelem pozadí. Důvodem byla především lepší přesnost detekce (viz. kap. 5) a jednodušší nastavení prahu pro přijetí klíčového slova. První varianta dekodéru na testovacích datech dokázala výrazněji rozlišit úroveň likelihood pokud se klíčové slovo v promluvě vykytovalo od úrovně likelihood v místech promluvy, kde se klíčové slovo nevyskytovalo. U druhé varianty byly rozdíly mezi těmito úrovněmi v mnohých případech setřené, což stěžovalo nastavení prahu a zvyšovalo počet falešných detekcí.

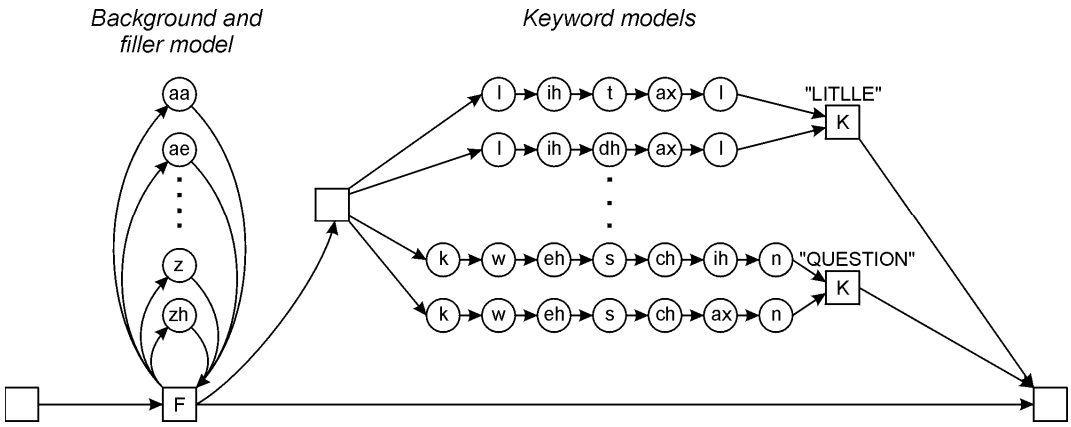
Komponenta implementující dekodér v knihovně umožňuje nastavení prahu, přidávání a rušení klíčových slov a online detekci klíčových slov na základě vektorů pravděpodobností fonémů. Komponenta se inicializuje implicitní hodnotou prahu pro detekci klíčových slov. Tuto hodnotu však nemusí sdílet všechna slova. Při vkládání slov je možné určit hodnotu prahu specifickou pro dané slovo. Tato možnost byla implementována vzhledem k faktu, že univerzální práh se při testech nejevil jako příliš vhodný. Výsledná likelihood různých slov se často pohybuje v různých úrovních.

[*] SpeechDat-E – Eastern European Speech Databases for Creation of Voice Driven Teleservices
`http://www.fee.vutbr.cz/SPEECHDAT-E/`

Automatické stanovení optimálních prahů pro klíčová slova však není jednoduché a tak bylo alespoň umožněno, aby individuální prahy mohl zadat uživatel, byť to jistě není ideální řešení.

Dekodér umožňuje, aby byla klíčová slova přidávána a odebírána kdykoliv za běhu detektoru. Při přidání slova se automaticky vytvoří jeho model a přidá se do rozpoznávací sítě (viz. dále). Maximální počet slov, které mohou být současně rozpoznávány, je teoreticky omezen jen velikostí dostupné paměti. V praxi však více slov vzhledem k přesnosti rozpoznávání zvětšuje míru falešných detekcí. Maximální délka slova je z implementačních důvodů v programu omezena na 32 fonémů, což by mělo být vyhovující pro naprostou většinu případů. Dekodér neřeší transkripci slov z grafického zápisu na fonetický. Klíčová slova jsou akceptována ve fonetickém přepisu v notaci SAMPA (viz. příloha 2).

Dekodér pracuje nad rozpoznávací sítí. Původní síť, která byla uvedena v kapitole 3.3.1, byla v implementaci upravena tak, aby umožňovala online režim detekce klíčových slov (viz. obr. 4.5). Model pozadí *D* a fonémová smyčka *A* v původním schématu (viz. obr. 3.10) byly sloučeny do jediného modelu, který plní funkce obou předchozích. Fonémová smyčka *C*, která modelovala úsek řeči za klíčovým slovem, byla vypuštěna. Síť je implementována tak, že pro každý její stav je uchovávána odpovídající hodnota částečné Viterbiho likelihood pro aktuální čas. S přechody mezi každými dvěma stavy je asociována přechodová pravděpodobnost. Všechny možné přechody z libovolného stavu jsou však v našem případě stejně pravděpodobné a tak přechodová pravděpodobnost nemusí být uvažována. Pro přechod mezi dvěma různými stavy (fonémy) je však zavedena tzv. phoneme insertion penalty (PIP). Je to konstantní hodnota, která se vždy při přechodu do dalšího stavu přičte k částečné likelihood. PIP v podstatě ovlivňuje rychlost průchodu modelem a její hodnotu je potřeba na trénovacích datech optimalizovat tak, aby doba setrvání modelu při detekci klíčového slova odpovídala délce slova v promluvě. PIP byla v knihovně nastavena na hodnotu -7 (viz. kap. 5).



Obr. 4.5: Síť pro online detekci klíčových slov (převzato z [15])

Činnost dekodéru spočívá pro každý vektor pravděpodobností fonémů v aktualizaci částečných Viterbiho věrohodností jednotlivých stavů modelů a na jejich základě v určení zda bylo nějaké klíčové slovo detekováno. Viterbiho věrohodnosti jsou aktualizovány podle vztahu (22), PIP není v tomto vztahu explicitně vyjádřena, ale je v zásadě součástí přechodové pravděpodobnosti. Vzhledem k tomu, že všechny částečné likelihood v síti odpovídají stejně dlouhým cestám, je skutečně možné je přímo porovnávat mezi sebou a v každém stavu uchovávat jen jednu maximální. Věrohodnosti jsou navíc v implementaci aktualizovány od posledních stavů modelu k prvním, takže není nutné mít v paměti uložené zvlášť věrohodnosti pro aktuální a předchozí čas. Při přidání nového

modelu slova do sítě jsou hodnoty částečných Viterbiho věrohodností odpovídající jednotlivým stavům modelu inicializovány na velkou zápornou hodnotu.

První stavy modelů slov jsou napojeny na výstup modelu pozadí. Aktualizace modelu pozadí se provádí stejně jako u modelů slov. Výstup (a současně vstup) modelu pozadí představuje v zásadě věrohodnost nejpravděpodobnější cesty mezi všemi fonémy.

Věrohodnost výskytu klíčového slova je spočítána jako poměr věrohodností posledního stavu jeho modelu a modelu pozadí (23). Z důvodu normalizace je ještě tato věrohodnost dělena dobou strávenou v modelu slova. Klíčové slovo je detekováno, pokud jeho věrohodnost překročí zvolený práh. Detekce by přitom neměla být signalizována hned když dojde k překonání prahové hodnoty, ale až ve chvíli, kdy hodnota věrohodnosti dosáhne lokálního maxima. Tento okamžik by měl odpovídat konci slova v promluvě. Vzhledem k tomu, že detekce probíhá online, není možné maximum zachytit hned jak nastane, ale s drobným odstupem. V implementaci knihovny je proto použit krátký buffer posledních hodnot věrohodností. Maximum nastane, pokud je prostřední hodnota bufferu maximální. Velikostí bufferu je zároveň nastaven minimální interval, po kterém může být též slovo znovu detekováno. Optimální velikost zabrání vícenásobným signalizacím detekce téhož výskytu klíčového slova v promluvě.

Zbývá dodat, že pro jeden čas může dojít k detekci více slov současně. Vzhledem k tomu, že tato situace nastává velice zřídka a nelze jednoduše určit, která detekce je správná a která falešná (za předpokladu, že v jeden moment může být vysloveno pouze jedno slovo, které není příponou jiného klíčového slova), je signalizováno poslední detekované slovo.

4.3.5 Rozhraní knihovny

Komponenta tvořící rozhraní knihovny propojuje komponenty popsané v předchozích podkapitolách a poskytuje uživateli operace pro řízení detektoru klíčových slov. Dostupné jsou operace pro spuštění a zastavení detekce, nastavení prahu pro přijetí slova a přidání či odebrání klíčového slova. Detekce je signalizována uživateli pomocí callback metody.

Knihovna se inicializuje objektem implementujícím událost detekce klíčového slova a hodnotou implicitního prahu. Spuštěním detektoru je aktivováno nahrávání rámců a proces detekce. Detekce klíčových slov probíhá asynchronně do doby než je zastavena příslušnou metodou. Nezávisle na stavu detektoru je možné měnit hodnotu prahu a přidávat příp. odebírat klíčová slova.

Pro manipulaci s klíčovými slovy volá komponenta rozhraní příslušné operace dekodéru. Rozhraní v tomto ohledu funkčnost dekodéru nerozšiřuje a je proto zapotřebí přidávat klíčová slova zapsaná pomocí fonetické abecedy SAMPA (viz. příloha 2). Automatický převod pravopisu slova na jeho výslovnostní podobu, který by byl jistě vhodný pro snadnější ovládání knihovny, byl ponechán jako námět na případné rozšíření v budoucnu. Jednoduché řešení by v tomto případě spočívalo ve využití slovníku. Ten je však poměrně dost velký a tak nepříliš vhodný pro telefony se značně omezenou pamětí. Vhodnější řešení by bylo implementovat fonetickou transkripci na základě fonologických pravidel. V češtině, kde se většinou slova čtou tak, jak se píší, by tento přístup mohl být postačující. Výjimky z pravidel by mohly být ošetřeny malým slovníkem.

Detekované klíčové slovo je předáno uživateli pomocí již zmíněné callback metody. Tento způsob odpovídá návrhovému vzoru Observer, který je pro podobné účely hojně používán i v OS Symbian. Kromě klíčového slova je do metody také předána hodnota jeho likelihood. Před zpracováním slova musí navíc nejdřív uživatel zkontrolovat, jestli nedošlo v asynchronní operaci k výjimce (viz. kap. 4.3.1). Pro připomenutí budiž řečeno, že knihovna vyžaduje pro svou činnost spuštěný plánovač událostí (viz. kap. 4.1.4). Ten je nezbytný pro běh asynchronních operací.

Pro ověření funkce knihovny byly implementovány dvě jednoduché aplikace, které ji používají. První aplikace byla vytvořena jako konzolová. Její funkce spočívá ve spuštění detektoru s několika předdefinovanými klíčovými slovy. Detekovaná slova jsou následně vypisována na displej spolu s hodnotami jejich likelihood. Druhá aplikace má GUI a na rozdíl od první již umožňuje uživateli měnit parametry detektoru. Aplikace je tvořena dvěma okny. Jedno zobrazuje seznam klíčových slov pro detekci a umožňuje ho editovat. Druhé okno zobrazuje poslední detekované slovo. Aplikace také umožňuje upravovat prahy pro přijetí klíčových slov.

5 Testování a výsledky

Vývoj knihovny byl započat v rámci semestrálního projektu implementací prototypu detektoru v jazyce C (viz. kap. 4.2). Vývoj finální verze spočíval nejdříve v doplnění chybějící funkčnosti prototypu a teprve následně v převodu pod Symbian OS. Tento přístup umožnil základní otestování detektoru s využitím nástrojů, které nejsou pod Symbian OS dostupné. Správnost některých výpočtů byla ověřena pomocí toolkitu HTK a STK (viz. dále). Části programu, ve kterých procesor tráví nejvíce času a tedy části vhodné pro optimalizaci pak byly vytipovány nástrojem `gprof`. Detektor pro testování byl implementován ve dvou verzích – offline a online. Offline verze, která nebyla určena pro implementaci ve finální verzi, umožnila značnou část testování, které by v online verzi bylo jen obtížně realizovatelné. Jednotlivé komponenty byly testovány postupně tak, jak byly vytvářeny od komponenty pro extrakci příznaků přes algoritmus Forward pass až po dekodér.

Extrakce příznaků byla testována s využitím HTK. Výpočet příznaků byl implementován tak, aby byl kompatibilní s tím, jak je počítá právě tento toolkit. Na testovacích datech bylo ověřeno, že výstup HTK a implementovaného programu je prakticky totožný. Drobné odchylky v řádu desetitisícin, které se v datech vyskytovaly, byly způsobeny zaokrouhlovacími chybami v aritmetice s plovoucí řádovou čárkou. Po otestování extrakce příznaků mohla být otestována další část detektoru a sice vyhodnocení neuronové sítě.

Neuronová síť byla trénována pomocí STK. Výsledné hodnoty vah a prahů mohou být v STK zapsány pomocí matic do speciálního souboru transformací. Pomocí dalšího nástroje z tohoto toolkitu je možné tyto transformace aplikovat na vstupní vektory, v našem případě vektory příznaků. Pomocí maticového počtu se v našem případě provede algoritmus Forward pass a obdržíme výstupní vektory pravděpodobností fonémů. Tímto způsobem byly získány pro testovací data referenční výsledky na jejichž základě byla ověřena funkčnost vyhodnocení neuronové sítě v detektoru.

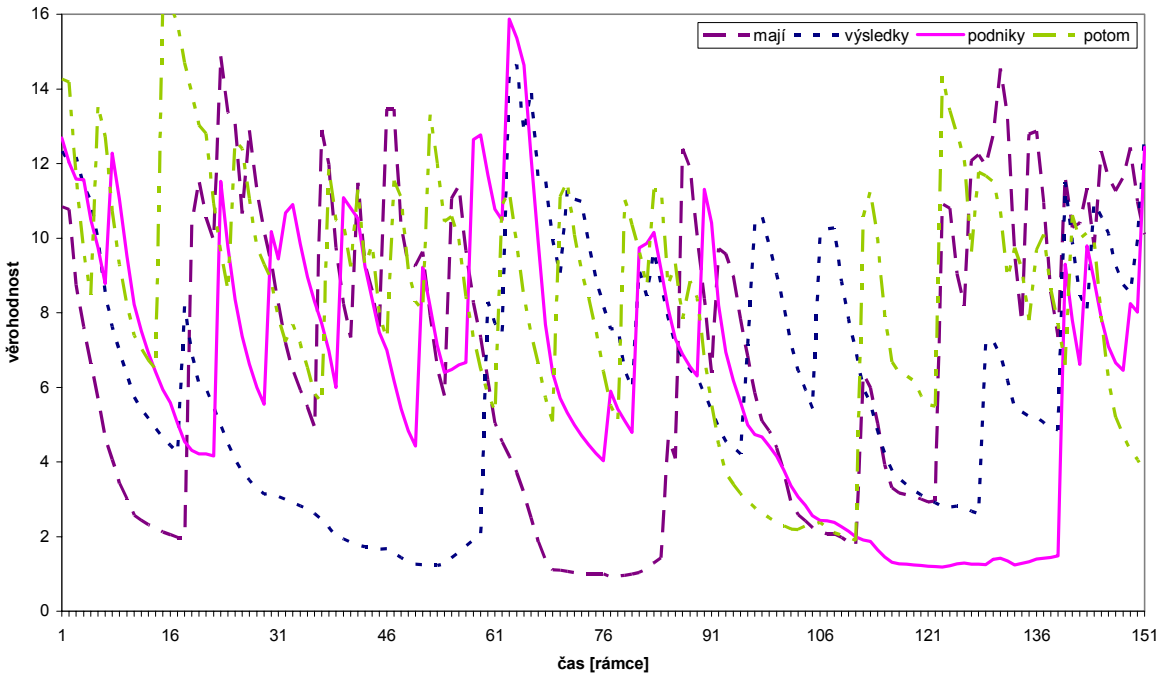
Poslední částí detektoru je dekodér. Implementovány byly pro testování dvě verze dekodéru (viz. kap. 4.3.4). Verze dekodéru s rozpoznávací sítí byla porovnávána s dekodérem dostupným v STK. Modifikovaný Viterbiho dekodér dává díky odlišnému přístupu jiné výsledky a tak tímto způsobem nemohl být otestován. Místo toho byl otestován na vzorcích z databáze SpeechDat, která obsahuje pro každou promluvu tzv. label file s časováním jednotlivých fonémů. Na datech, které byly použity pro trénování sítě, tak bylo ověřeno, že začátky a konce jednotlivých slov v promluvě a odpovídajících detekovaných slov se přibližně kryjí.

Obě verze dekodéru byly v rámci testování porovnány a byly zjištěny jejich vlastnosti. Průběh likelihood pro oba dekodéry pro jednu promluvu a totožná klíčová slova je na obrázku 5.1 resp. 5.2. Vliv použitého dekodéru na celkovou přesnost rozpoznávání je popsán dále.

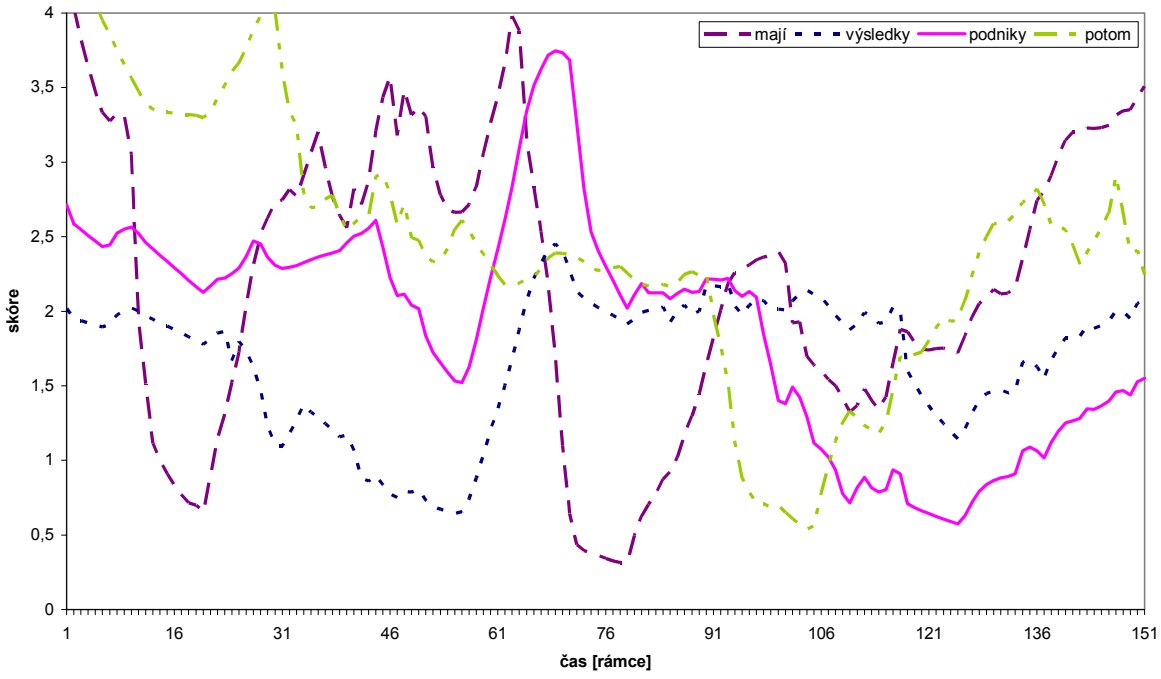
Výhodou modifikovaného Viterbiho dekodéru bylo, že pro něj nebylo zapotřebí optimalizovat hodnotu PIP. Jinými slovy vyhovovala nulová PIP, aby byla klíčová slova ideálně zarovnaná vůči promluvě. Oproti tomu klasický dekodér bez vhodně nastavené PIP nepracoval správně. V dalších hlediscích se však už klasický dekodér s rozpoznávací sítí jevil jako výhodnější.

Klasický dekodér dokázal na rozdíl od modifikovaného dekodéru lépe rozlišit věrohodnost pro případ detekce slova a průměrnou věrohodnost v částech promluvy, kde se klíčové slovo nevyskytovalo.

Porovnání implementovaných dekodérů pro promluvu „nejhorší výsledky mají podniky“



Obr. 5.1: Dekodér s rozpoznávací sítí (zobrazena je opačná věrohodnost)



Obr. 5.2: Modifikovaný Viterbiho dekodér

Pro klasický dekodér bylo při testování také jednodušší pro danou promluvu a klíčová slova nastavit optimální univerzální práh pro jejich přijetí. Průměrná hodnota věrohodnosti byla v tomto případě podobnější pro všechna slova. Průměrné hodnoty skóre se při použití modifikovaného dekodéru vzájemně více lišily, přestože tento dekodér používá dvojí normalizaci. Bylo tedy obtížnější nastavit práh, abychom detekovali všechny výskyty klíčových slov a zároveň odfiltrovali falešné detekce. Tento problém je patrný na obrázku 5.2, kde pro slova „podniky“ a „potom“ není možné nalézt práh, který by je rozlišil, zatímco pro tatáž slova v případě klasického dekodéru na obrázku 5.1 ideální hodnota prahu existuje.

Dále budou popsány dosažené výsledky, tedy především přesnost rozpoznávání. Nejdříve budou porovnány neuronové sítě, které byly testovány. Úspěšnost klasifikace vstupních vektorů pro jednotlivé fonémy (frame accuracy) byla v případě sítě s 500 neurony ve skryté vrstvě 51,38 % a pro síť se 100 neurony ve skryté vrstvě 50,99 %. Přesnost s jakou si odpovídají fonémy klasifikované sítí s fonémy odpovídající promluvy (phoneme accuracy) byla pro větší síť 32,42 % a pro menší síť 31,57 %. Dále bude zhodnocena celková úspěšnost detekce pro síť se 100 neurony ve skryté vrstvě, která byla zvolena pro implementaci.

Úspěšnost detekce byla vyhodnocena metrikou Figure-of-Merit (FOM) definovanou NIST institutem v USA[*]. FOM je založena na tzv. receiver operating characteristic (ROC) křivce [24]. Tato křivka vyjadřuje závislost mezi počtem falešných detekcí na klíčové slovo za hodinu promluvy a odpovídajícího podílu správných detekcí. Pro vyhodnocení nás zajímají především podíly správných detekcí pro míru falešných detekcí menší než 10. FOM se spočítá jako průměr správných detekcí přes prvních 0 až 10 falešných detekcí za hodinu. FOM se tedy dá přibližně interpretovat jako přesnost detekce za předpokladu 5 falešných detekcí za hodinu [6].

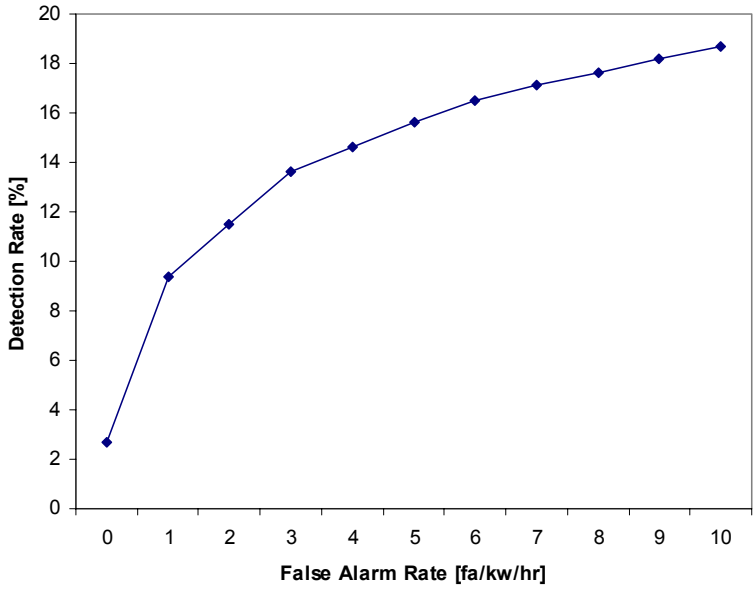
FOM byla vyhodnocena s použitím testovací sady více než 2000 promluv z databáze SpeechDat-E. Slovník klíčových slov obsahoval 1679 nejčastějších slov z databáze o délce alespoň 3 fonémů. Slovník byl profiltrován tak, aby se v něm nevyskytovala slova, která jsou již obsažena v jiných delších slovech. Výsledný slovník klíčových slov, který byl použit pro vyhodnocení, tedy obsahoval 1020 slov s celkovým počtem 3175 výskytů v testovací sadě. Celkem 268 klíčových slov se pak v testovací sadě nevyskytovalo vůbec. Výsledky pro oba testované dekodéry jsou shrnuty v tabulce 5.1.

	Správné detekce	**Falešné detekce**	**FOM**
Klasický dekodér	885	794266	14,59
Modifikovaný dekodér	439	680692	4,27

Tab. 5.1: Figure-of-Merit testovaných detektorů

Z tabulky vyplývá, že klasický dekodér dosáhl zhruba 3 krát lepších hodnot než modifikovaný. Nejlepší FOM u klasického dekodéru (14,59) byla dosažena pro PIP o hodnotě -7. ROC křivka pro klasický dekodér je uvedena na obrázku 5.3. Pro 10 falešných detekcí na klíčové slovo za hodinu tak například detekujeme správně 19 % všech slov, což není až tak špatné vzhledem k velikosti neuronové sítě. Podrobnějším vyhodnocením také bylo zjištěno, že lepší přesnosti bylo dosaženo u delších slov. To odpovídá skutečnosti, že krátká slova je obtížnější diskriminovat a způsobují tedy víc falešných detekcí.

[*] NIST - National Institute of Standards and Technology
http://www.nist.gov



Obr. 5.3: ROC křivka pro klasický dekodér

Na přesnost rozpoznávání má v implementaci největší vliv velikost neuronové sítě resp. použité příznaky. Pro urychlení výpočtu byly použity jen jednoduché PLP příznaky. Větší přesnosti by bylo možné docílit rozšířením vektoru příznaků o delta a akcelerační koeficienty.

Největší problém, který zásadním způsobem ovlivňuje přesnost detekce je už několikrát zmiňované nastavení prahu. Univerzální práh pro všechna slova se ukázal jako nevyhovující, přestože u klasického dekodéru byl použitelnější než u modifikovaného. Jediný práh byť optimálně nastavený způsoboval velké množství falešných detekcí resp. malý počet správných detekcí. S individuálními prahy bylo docíleno znatelně lepších výsledků. V implementaci je zadávání slovně specifických prahů umožněno. Optimální práh závisí na konkrétních fonémech, z kterých se slovo skládá. Metody pro automatické předpočítání prahů sice již byly publikovány [20], ale jejich implementace není jednoduchá a byla ponechána jako případné rozšíření.

Drobným problémem především u modifikovaného dekodéru byla občasná vícenásobná detekce jednoho výskytu klíčového slova. Skóre rostlo po detekci pomalu a v závislosti na promluvě se mohlo stát, že vytvořilo po kratší době další, tentokrát falešný lokální extrém vyhovující prahu. Tento problém byl řešen reinicializací modelu (nastavením výchozích hodnot skóre jeho stavů) po každé detekci odpovídajícího klíčového slova.

Kromě přesnosti rozpoznávání byla kritériem pro vyhodnocení dosažených výsledků také rychlost implementace. Vzájemně byly v tomto ohledu porovnány testované dekodéry. Na jednom PC trvalo klasickému dekodéru zpracování jednoho rámce průměrně 0,174 ms, týž úkol trval modifikovanému dekodéru průměrně 0,237 ms. Vychází nám tedy, že implementace klasického dekodéru je přibližně o 36 % rychlejší než implementace modifikovaného dekodéru.

Dále nás zajímalo, zda bude detektor funkční online tak, jak bylo zamýšleno. Toto kritérium však bohužel nebylo vyhodnoceno díky tomu, že v době dokončování práce byl pro testovací účely k dispozici pouze telefon se starší verzí Symbian OS a na platformě S60 1st Edition, kde chybí některé části MMF. Byla snaha o doinstalování chybějících částí, která však nebyla úspěšná. Rychlost nebylo bohužel možné vyhodnotit ani v emulátoru Symbian OS, kde byl jinak detektor testován, protože emulátor neumožňuje simulovat rychlost procesoru a periferií cílového zařízení.

6 Závěr

Cílem diplomové práce byla implementace detektoru klíčových slov pro mobilní telefon. Práce vycházela z postupu detekce klíčových slov, který již byl prověřený skupinou Speech@FIT. Účelem práce byla implementace detektoru pro operační systém Symbian a provedení případných optimalizací detektoru vzhledem k omezené výkonnosti hardware mobilních telefonů. Realizace práce spočívala nejdříve v teoretické přípravě a následně v praktické implementaci, testování, vyhodnocení a dokumentaci výsledného produktu.

Teoretická část se týkala především dvou oblastí. První z nich se vztahovala k OS Symbian (kap. 2). Bylo zapotřebí se seznámit s tímto OS jak z pohledu uživatele tak z pohledu vývojáře aplikací. Důležitým úkolem bylo vybrání verze systému a platformy uživatelského rozhraní, pro které bude detektor implementován. Vzhledem k jistým problémům v přenositelnosti aplikací mezi jednotlivými verzemi OS tento výběr určil počet zařízení, na kterých by měla být aplikace beze změny funkční. Po zvážení všech hledisek byla nakonec vybrána platforma S60 ve verzi 2nd Edition.

Druhou oblastí teoretické přípravy bylo seznámení se s implementací jednoduchého detektoru klíčových slov (kap. 3). Byl popsán postup, jakým způsobem detektor realizovat. Postup byl rozdělen na tři části podle hlavních komponent detektoru, kterými jsou extrakce příznaků, klasifikace a dekodér. V případě extrakce příznaků resp. dekodéru byly navrženy dvě varianty, z nichž byla vybrána varianta PLP příznaků resp. dekodéru s rozpoznávací sítí modelující zvlášť klíčová slova a pozadí. Výběr byl ovlivněn praktickými požadavky na implementaci a spolehlivost výsledného detektoru.

Praktická část spočívala nejdříve v implementaci prototypu detektoru, na jehož základě byl následně implementován finální detektor pro Symbian OS (kap. 4). Prototyp detektoru umožnil určit části aplikace vhodné pro optimalizaci a usnadnil jednak počáteční fázi implementace pro cílovou platformu a také testování základní funkčnosti detektoru. Detektor pro Symbian OS byl realizován jako dynamická knihovna, kterou můžou používat další programy pro zajištění detekce klíčových slov pro požadovaný účel. Knihovna byla navržena a implementována s využitím objektových rysů jazyka C++. Jednoduché aplikační rozhraní knihovny umožňuje řídit online detekci klíčových slov.

Knihovna byla testována na emulátoru OS Symbian a její spolehlivost odpovídá zvolenému postupu a implementačním možnostem (kap. 5). Použití knihovny je demonstrováno na dvou jednoduchých aplikacích. Knihovna by měla sloužit především jako základ pro rozšíření aplikací automatického zpracování řeči na přenosná zařízení s OS Symbian. Pro další využití v praxi by bylo vhodné knihovnu rozšířit o následující funkce a možnosti. Zadávání klíčových slov by bylo pro jednodušší práci s knihovnou vhodné umožnit tak, jak se píší a nikoliv v jejich fonetickém přepisu. Automatický převod by spočíval v implementaci fonologických pravidel. Práce s knihovnou by se také velice usnadnila, pokud by se podařilo implementovat automatické nastavování prahů specifických pro detekci jednotlivých klíčových slov. Posledním rozšířením by mohla být úprava knihovny za účelem dosažení kompatibility s nejnovější verzí platformy S60 3rd Edition.

7 Literatura

[1] Jode, M., Turfus, C.: Symbian OS System Definition. Symbian Developer Network, 2004. Dokument dostupný na URL http://www.symbian.com/developer/techlib/papers/SymbOS_def/symbian_os_sysdef.pdf (prosinec 2006).

[2] Willee, H., aj.: What Symbian OS Development Kit Do I Need? Symbian Developer Network, 2005. Dokument dostupný na URL http://www.symbian.com/files/rx/file6378.pdf (prosinec 2006)

[3] S60 Platform: Source and Binary Compatibility, Nokia Corporation. 2006. Dokument dostupný na URL http://sw.nokia.com/id/2ac24078-a0b3-4bd2-8b03-1707fa2e1405/ S60_Platform_Source_and_Binary_Compatibility_v1_4_en.pdf (prosinec 2006).

[4] Symbian OS: Creating Audio Applications In C++. Nokia Corporation, 2005. Dokument dostupný na URL http://sw.nokia.com/id/829fbb19-e7fb-4578-8365-6a66d74b2572/ Symbian_OS_Creating_Audio_Applications_In_Cpp_v1_0_en.pdf (prosinec 2006).

[5] S60 Platform: FAQ. Nokia Corporation, 2005. Dokument dostupný na URL http://sw.nokia.com/id/19397f28-e130-452a-8d0c-be67eb532cfe/ S60_Platform_FAQ_v1_6_en.pdf (prosinec 2006).

[6] Szöke, I. aj.: Phoneme Based Acoustic Keyword Spotting in Informal Continuous Speech. 2005. Dokument dostupný na URL https://www.fit.vutbr.cz/~szoke/papers/radioelektronika_2005.pdf (prosinec 2006).

[7] Szöke, I. aj.: Comparison of Keyword Spotting Approaches for Informal Continuous Speech. 2005. Dokument dostupný na URL https://www.fit.vutbr.cz/~szoke/papers/eurospeech_2005.pdf (prosinec 2006).

[8] Rosell, M.: An Introduction to Front-End Processing and Acoustic Features for Automatic Speech Recognition. 17. ledna 2006. Dokument dostupný na URL http://www.nada.kth.se/~rosell/courses/rosell_acoustic_features.pdf (prosinec 2006).

[9] Seltzer, M.: SPHINX III Signal Processing Front End Specification. CMU Speech group, 31. srpna 1999. Dokument dostupný na URL: http://cmusphinx.sourceforge.net/sphinx3/s3_fe_spec.pdf (prosinec 2006).

[10] Mehrotra, K., Mohan, C. K., Ranka, S.: Artificial Neural Networks. The MIT Press, 1997, ISBN 0-262-13328-8.

[11] Kontár, S.: Paralelní trénování neuronových sítí pro rozpoznávání řeči. Brno, 2006. 51 s.

[12] Gold, B., Morgan, N.: Speech and Audio Signal Processing. John Wiley & Sons, 2000, ISBN 0-471-35154-7

[13] Robinson, T.: Speech Analysis. 1998. Dokument dostupný na URL http://svr-www.eng.cam.ac.uk/~ajr/SA95/SpeechAnalysis.html (prosinec 2006).

[14] Schwarz, P.: Modifications of Viterbi algorithms for keyword detection, In: Proceedings of 8th Conference STUDENT EEICT 2002, Brno, CZ, FEKT VUT, 2002, s. 4, ISBN 80-214-2116-9.

[15] Szöke, I.: Phoneme posterior estimation and acoustic keyword-spotting. 2006. Tutoriál dostupný na URL http://tstmaster.let.uu.nl/EMaster-SS2006/tutorial10/tutorial10.html (prosinec 2006).

[16] Young, S. aj.: The HTK Book. Cambridge University Engineering Department, 2005. Dokument dostupný na URL http://htk.eng.cam.ac.uk/docs/docs.shtml (prosinec 2006).

[17] Burget, L., Schwarz, P., Karafiát, M., Černocký, J.: HMM Toolkit STK from Speech@FIT. 2004. Toolkit dostupný na URL http://www.fit.vutbr.cz/research/groups/speech/index.php?id=stk (prosinec 2006).

[18] Schraudolph, N. N.: A fast, compact approximation of the exponential function. 1999. Dokument dostupný na URL http://www.inf.ethz.ch/~schraudo/pubs/exp.pdf (prosinec 2006).

[19] Babin, S.: Developing Software for Symbian OS – An Introduction to Creating Smartphone Applications in C++, John Wiley & Sons Ltd, 2006.

[20] Junkawitsch, J. aj.: A new keyword spotting algorithm with pre-calculated optimal thresholds, Proc. Intl. Conference on spoken language processing ICSLP, 1996.

[21] Psutka, J.: Komunikace s počítačem mluvenou řečí, Academia, Praha, 1995, ISBN 80-200-0203-0.

[22] Multimedia Framework Architecture in S60 Devices, Nokia, 2007. Dokument dostupný na URL http://sw.nokia.com/id/1077c555-7179-4169-8626-f1a938bf78ff/ Multimedia_Framework_Architecture_in_S60_Devices_v1_0_en.pdf (květen 2007).

[23] Johnson, N.: Audio Streaming: How to successfully stream audio on Symbian OS v7.0s, Symbian Ltd., 2004. Dokument dostupný na URL http://developer.symbian.com/main/downloads/papers/Audiostream/Audi0StreamSymbianOSv1.1.pdf (květen 2007)

[24] Rohlicek, J. R., aj.: Continuous hidden Markov modeling for speaker-independent word spotting, BBN Systems and Technologies Corporation, Cambridge, 1989.

Seznam příloh

Příloha 1 – Dokumentace API knihovny

class CKeywordSpotter

extends CWavFramesStream as public

Třída pro detektor klíčových slov.

---

public **static** **CKeywordSpotter** * **NewL** (MKeywordSpotterCallback* aCallback,
TReal32 aThreshold)

Vytvoří novou instanci třídy. Ke spuštění detektoru dojde až po zavolání metody StartL().

Params:

aCallback	Objekt implementující callback metodu pro zpracování detekovaných klíčových slov.
aThreshold	Práh pro detekci/zamítnutí klíčového slova.

Returns:

Ukazatel na nově vytvořenou instanci.

---

public **static** **CKeywordSpotter** * **NewLC** (MKeywordSpotterCallback* aCallback,
TReal32 aThreshold)

Vytvoří novou instanci třídy. Ke spuštění detektoru dojde až po zavolání metody StartL(). Metoda ponechá na Cleanup zásobníku ukazatel na vytvořenou instanci.

Params:

aCallback	Objekt implementující callback metodu pro zpracování detekovaných klíčových slov.
aThreshold	Práh pro detekci/zamítnutí klíčového slova.

Returns:

Ukazatel na nově vytvořenou instanci.

---

public ~ **CKeywordSpotter** ()

Destruktor. Uvolní alokovanou paměť a odstraní objekt.

---

public **void** **AddKeywordL** (**const** TDesC& aKeyword)

Přidá další klíčové slovo do seznamu pro detekci. Pro detekci bude použit globální práh.

Param:

aKeyword	Klíčové slovo pro přidání.

---

public **void** **AddKeywordL** (**const** TDesC& aKeyword, TReal32 aThreshold)

Přidá další klíčové slovo do seznamu pro detekci. Pro detekci bude použit zadaný práh.

Param:

aKeyword	Klíčové slovo pro přidání.
aThreshold	Práh pro detekci slova.

---

public TInt **RemoveKeyword** (TInt aIndex)

Odstraní klíčové slovo ze seznamu pro detekci.

Param:

aIndex Index klíčového slova v seznamu.

Returns:

KErrNone pokud bylo klíčové slovo úspěšně odstraněno, jinak vrací KErrArgument (zadaný index není platný).

---

public TReal32 **GetThreshold** ()

Získá aktuální hodnotu globálního prahu používanou pro přijetí klíčových slov.

Returns:

Aktuální hodnota prahu.

---

public **void** **SetThreshold** (TReal32 aThreshold)

Nastaví novou hodnotu globálního prahu pro přijetí klíčových slov.

Param:

aThreshold Nová hodnota prahu.

---

public **void** **StartL**()

Spustí streaming audio vstupu z mikrofonu.

Inherited From:

CWavFramesStream

---

public **void** **Stop** ()

Ukončí streaming audio vstupu z mikrofonu.

Inherited From:

CWavFramesStream

class MKeywordSpotterCallback

Abstraktní třída definující rozhraní, které musí implementovat třídy, které chtějí používat CKeywordSpotter.

---

public **virtual void** **KeywordDetected** (TInt aError, **const** TDesC& aKeyword, TReal32 aScore) = 0

Callback metoda, kterou volá objekt třídy CKeywordSpotter, aby předal uživateli detekované klíčové slovo.

Params:

aError KErrNone pokud během detekce nenastala chyba, jinak jeden ze systémových chybových kódů.

aKeyword Detekované klíčové slovo.

aScore Likelihood detekovaného klíčového slova.

Příloha 2 – Fonetická abeceda SAMPA

SAMPA SYMBOL	ORTHOGRAPHIC TRANSCRIPTION	SAMPA TRANSCRIPTION	ENGLISH TRANSLATION
Vowels			
i	myš	miS	mouse
e	les	les	forest
a	pas	pas	passport
o	rok	rok	year
u	kus	kus	piece
i:	pít	pi:t	to drink
e:	lék	le:k	drug
a:	rád	ra:t	glad
o:	móda	mo:da	fashion
u:	půl	pu:l	half
diphthongs			
o_u	mouka	mo_uka	flour
a_u	auto	a_uto	car
Consonants			
plosives			
p	pes	pes	dog
b	bota	bota	shoe
t	tam	tam	there
d	dům	du:m	house
c	tito	cito	these
J\	děd	J\et	grandfather
k	krk	krk	neck
g	kde	gde	where
affricates			
t_s	cíl	t_si:l	aim
d_z	leckdy	led_zgdi	at times
t_S	čas	t_Sas	time
d_Z	léčba	le:d_Zba	medical treatment
fricatives			
f	forma	forma	form
v	vak	vak	bag
s	sen	sen	dream
z	zub	zup	tooth
Q\	tři	tQ\i	three
P\	řád	P\a:t	order
S	šaty	Sati	clothes
Z	žal	Zal	regret
j	jas	jas	brightness
x	chata	xata	cottage
h\	had	h\at	snake
liquids			
r	ret	ret	lip
l	led	let	ice
nasals			
m	mák	ma:k	poppy
n	noc	not_s	night
N	banka	baNka	bank
J	nic	Jit_s	nothing